2012 International Conference on Software and Computer Applications (ICSCA 2012)
IPCSIT vol. 41 (2012) © (2012) IACSIT Press, Singapore

Using Architectural Patterns to Improve Modularity in Software Architectural Design

SayedMehranSharafi[1+],MortezaGhasemi[2], NaserNematbakhsh [3]

[1,2,3] Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Esfahan, Iran
[1]mehran_sharafi@iaun.ac.ir,[2] morteza.ghasemi@sco.iaun.ac.ir,[3] nemat@eng.ui.ac.ir

Abstract. Architecture plays an essential role to achieve software quality attributes. A wide variety of architectural patterns and tactics are introduced in the literature to design software architecture. Modularity relates to quality attributes such as maintainability, portability, reusability, interoperability and flexibility. Modularity also depends on complexity aspects such as coupling and cohesion. There are several kinds of architectural patterns and tactics for increasing cohesion and decreasing coupling to control complexity that software architects can utilize them. The major question of architects is how to use these patterns and tactics. Hence, in this paper a method is proposed to select appropriate architectural patterns and tactics for improving modularity in software architectural design. Weconduct this method in an applicant project and use component level metrics to measure coupling and cohesion of the architecture design.

Keywords: Architectural patterns, Design patterns, Coupling and cohesion, Architectural tactics.

1.Introduction

Architecture plays an important role for an organization to meet its business objectives. Architecture is the first product that developers and architects use to achieve software quality attributes [1]. Modularity is a significant quality attribute while designing architectures of large systems [8]. Modularity relates to quality attributes such as maintainability, portability, reusability, interoperability and flexibility [11]. Modularity is a key factor in the success of a process because module decomposition is a way of managing the complexity of a system. This means that the degree of modularity is proportional to the degree of how loosely coupled and highly cohesive software elements of a system are. In addition, modularity indicates the degree of granularity or decomposition that the architecture of the system has experienced [6]. Coupling is defined as the connection between two modules. Two modules that have relatively few interdependencies are loosely coupled. Cohesion refers to the degree of dependencies within the boundary of a module. The lower the total coupling among modules, the higher level of design quality is obtained. Furthermore, cohesion should be as high as possible to achieve better design quality [9]. In this paper a method is proposed for architects to design modular architectures. The goal of this method is improvement of modularity by applying architectural patterns and design patterns that increase cohesion and decrease coupling. We applied this method in a case study and evaluated it by using component level metrics for measuring degree of coupling and cohesion.

This paper is organized as follows: in Section 2, an overview of architectural patterns, tactics, design patterns and relationships between them is discussed and a comparison between architectural patterns and design patterns is presented. Section 3 describesuseful architectural patterns and tactics to improve modularity. Section 4 classifies architectural patterns and corresponding tactics to increase cohesion and decrease coupling. In Section 5 design patterns which have positive effects on modularity are discussed. In section 6 the proposed method for software architectural design is explained in detail. In section 7 results of

+Corresponding author. Tel.: + (983312291111-2440);
E-mail address:mehran_sharafi@iaun.ac.ir

applying the method on a case study is presented. Conclusions and a summary of future works are explained in Sections 8.

2.Architectural Patterns, Architectural TacticsandDesign Patterns

A tactic is a design decision to achieve specified quality attributes. In fact, a tactic is a design option for architects [1,2].Patterns designate the schema of basic structures of the software systems of organization as they provide a set of predefined subsystems, their functions, governing rules, and recommendations for building the relationship between them [5]. A set of principles and a coarse grained pattern that provides an abstract framework for a family of systems is called architectural pattern [3]. An architectural pattern is specified through a set of element types, a topological layout of the elements indicating their interrelationships, a set of semantic constraints and a set of interaction mechanisms [reference]. In other words, patterns encapsulate tactics. In fact, a pattern or a collection of patterns is/are designed to realize one or more tactics which are chosen by the architects [1]. For example,reflection pattern that supports maintainabilitywill likely use both maintain a semantic coherence tactic and an encapsulation tactic[2].But,what is a design pattern? And what is the difference between an architectural pattern and a design pattern?A design pattern provides a strategy for refining the subsystems, components of a software system, or the relationships between them; and it describes commonly-recurring structure of communicating components that solves a general design problem within a specific context[5]. Accordingly, an architectural pattern is similar to a design pattern in that they both describe a solution to a problem in a particular context. The only difference is the granularity at which they describe the solution. In a design pattern, the solution is relatively fine grained and is depicted at the level of language classes. In an architectural pattern, the solution is coarser grained and is described at the level of subsystems or modules and their relationships and collaborations [10].

3.Architectural PatternsandTacticsfor Improving Modularity

Coupling and cohesion are defined in terms of responsibility. A responsibility is an action, knowledge to be maintained, or a decision to be carried out by a software system or an element of that system. Strength of the coupling of two responsibilities is defined as the probability that a modification to one responsibility will propagate to the other. Coupling is an asymmetric relation. In other words, the strength of coupling between responsibility A and responsibility B is not necessarily the same as that between responsibility B and responsibility A. If the relationships among elements are minimized, coupling is reduced. To achieve this goal, we can minimize relationships among separated elements and maximize relationships among elements in the same module. Maximizing the relationships among elements in the same module isdefined as cohesion. In fact, cohesion expressesthat if responsibilities A and B are collocated in the same module, then the cost of changing is less. There are many tactics for improving degree of coupling and cohesion.Some coupling tactic, reduce couplingfrom a responsibility to another and some others from a module to another.The purpose of cohesion Tacticsisto move a responsibility from one module to another, and to reduce the likelihood of side effects to other responsibilities in the original module. There are eight software architectural patterns for improving modularity. These patterns use coupling and cohesion tactics. Table 1shows the correspondence between these patterns and the tactics they implement [2]:

Table (1):Architectural Patterns and Corresponding Tactics

Patterns	**IncreaseCohesion**		**Reducecoupling**				
	MaintainSemantic Coherence	**AbstractCommon Services**	**UseEncapsulation**	**UseaWrapper**	**Restrict Comm. Paths**	**UseanIntermediary**	**Raisethe Abstraction Level**
Layers	X	X	X		X	X	X
Pipe-and-Filter	X		X		X	X	
Blackboard	X	X			X	X	X
Broker	X	X	X		X	X	X

Model-View- Controller	X		X			X	
Presentation- Abstraction- Control	X		X			X	X
Microkernel	X	X	X		X	X	
Reflection	X		X				

4.Classification and Composition Patterns

In software architectural design, there are noconstraints for selection patterns. For example, anarchitect may use 3-tier patternin the first architectural decision for security reasons, and in the next step, for decompositionpresentationtier, he/she may useMVC pattern. But there is a problem in combination patterns. This problem is the inconsistency between them. For example, in becauseinconsistency between quality attributes ofReusability and Performance, we cannot combine related patternsof these quality attributestogether. For choosing a pattern, there are a lot of factors such as type of problem, constraints andexperience of architects [3]. In [4], patternsare classifiedwith regardto type of problem. Table2 showsthese patterns.

Table (2):Classification of patternswith regard to type of problem

Problem Category	**From Mud to Structure**	**Distributed Systems**	**Interactive Systems**	**Adaptable Systems**
Pattern	Layers	Broker	MVC	Microkernel
	Pipes	Pipes &Filters	PAC	Reflection
	Blackboard	Microkernel	Layers	

In [3] different patterns are classified according to different views. For example, patterns Client/Server, N-Tier and 3-Tierare located in deploymentview and layer pattern is located in structuralview.

5.Design Patterns for Improving Modularity

In section 2, we discuss the differences between design patterns and architectural patterns.The following table (table 3) lists important design patternswhich have a positive influence on coupling and cohesion. The patterns should therefore keep the coupling between software elements low, and each software element as cohesive as possible [7,8].

Table (3): List of Design patterns for improving modularity

Design Pattern		
Decorator	Cohesive Modules	Module Façade
Code to an Interface	Abstract Factory	Published Interface
Decorator	Factory	Acyclic Relationships

6.Method

In proposed method, we describe how to use related architectural patterns and design patterns to achieve modularity in architectural design. This method proposes a variety of architectural patterns and design patterns regarding granularity and type of the problem in each of the architectural design stages. The main point is how and when to use architectural patterns and design patterns.

This method includes a hierarchy of architecture design decisions that decomposes a system from a coarse grained design into a fine grained one. Therefore, in accordance with what discussed in section 3, at the first stages of design, we should use architectural patterns and in the next other stages we should use appropriate design patterns. The other issue is how to choose and combine architectural patterns. In this method, architect should choose a deployment view pattern for the first decision of the design because these patterns decompose the system in a physical way. In section 3, three patterns which are client/server, n-tier and 3-tier introduced as the deployment patterns. In the next stage, it is recommended to architects that with regard to the type of the problem in hand adopt one of the changeability improvement patterns mentioned in Table 2. In the next stage, regarding the details of the usecases, functional requirements are allocated as the functional modules to the components which have extracted by applying architectural patterns. Afterward, by considering types of interdependencies among functional modules, we use appropriate design patterns which introduced in Table 3 for enhancing modularity. We iterate these operations for all the modules which need to be decomposed.In the following section, we discuss this method.

6.1Method Steps

- Decompose system bychoosing one of the deployment view patterns
- Decompose components obtained from the previous stage, by choosing one of the architectural patterns in table 1and with regard to the type of problem
- Allocate functionality from the use cases to components obtained from the previous stage
- Choose one of the functional modules (subsystems) and decompose it by choosing one of the design patterns in table 3.
- Repeat the previous stage for every module (subsystem) that needs further decomposition.

Figure 1 shows main steps of this method.

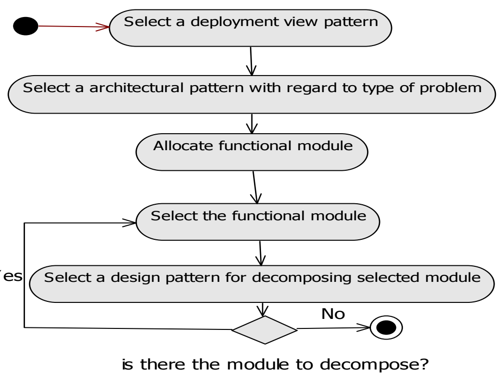
Select a deployment view pattern
Select a architectural pattern with regard to type of problem
Allocate functional module
Select the functional module
Select a design pattern for decomposing selected module
Yes
No
is there the module to decompose?

Fig. 1: main steps of proposed method

7.Case Study: Applicant Management System

Applicant Management System is a software product that can use by different organizations. The goal of this system is to meet people's demands. This system is performed based on the Web. People send their requests by this system. For sending a request, first must register personal information and specification of requests. In the next step, his/her requests will be considered and a letter will be issued for his/her requests.With regard this system is a software product, it needs to be extendable and changeable. Therefore,modularity is one of major quality attribute for this system. Functional requirements or functional modules (subsystems) as follows:

- Letters management system
- Applicant management systems
- User management system
- Persons management system

7.1Applying the proposed method

First step: we applyClient/Server pattern.

Second step: Type of problem is Interactive. We use MVC pattern.These patterns reveal three software elements; the client part which includes the view element and the server part which includes the model element, and the controller element.

Third step: We decompose model component and allocate to it subsystems User management system,letters management system, Applicant management systems, and Persons management system. The following figure (figure 2) shows conceptual architecture up to this step.

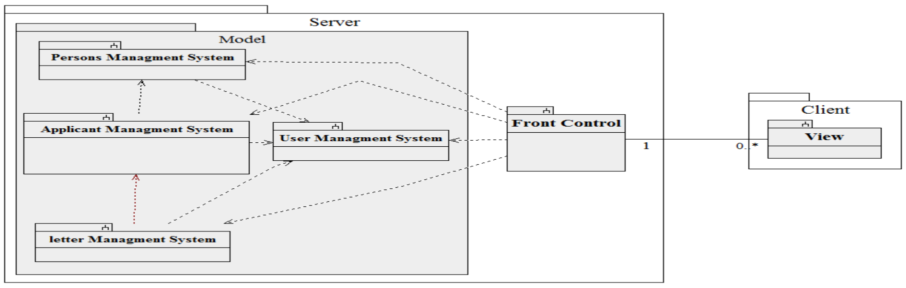

Fig.2: Conceptual Architecture of Applicant Management System

Forth step:In this step, with regard types of dependency (incoming or outcoming) between subsystems, we decompose thembyusing design patterns in table 3. For example, wedecompose User Management System because of its imperative role in the system with several incoming dependencies. We apply the Module Facade Pattern and Cohesive Module Pattern. The output of this step concluded in two child modules: User Manager and Security Manager.Similarly, we can repeat the previous stage for decomposing other subsystem that needs further decomposition.For measuring coupling and cohesion, we use component level metrics that are provided in [6]. These metrics are described in table 4. Table 5 shows these metrics value for applicant project.

Table (4): List of Metrics for Measuring Coupling and Cohesion

Metrics	Description
NCOMPUC	This metric measures how many components share the same responsibilities by looking at the sequence diagram for each use case and count the components involved
NUCPCOM	By looking at the use cases of the system a component is involved in, we can see the responsibilities for that component
CBCOM	This metric counts all the dependencies that one component has to other components

Table (5): Metrics values for Applicant Management System

Component	Use cases	NCOMPUC	NUCPCOM	CBCOM
User management system	Manage user	2	11	4
	Log in	2		
	Log out	2		
	Check identity	1		
Persons management system	Manage person	3	7	3
Applicant management system	Send request	4	6	4
	Check request	4		
Letters management system	Export letter	5	4	3
	Receive letter	5		
	Receive answer	5		
	Follow Up letter	5		
Front controller			9	4

8.Conclusions and a Summary of Future Work

In this paper, we determined the list of Architectural tactics, architectural patterns and design pattern for improving modularity. We categorized architectural patterns based on tactics they implemented, and the type of problem. We classified some other patterns such as client/server in deployment view, so that architects could use them in the first architectural design decision and also because they decompose the system at high levels and usually physically. We presented a novel approach to improve modularity in architectural level. In particular, we proposed how and when architects could use architectural patterns and design patternstogether in this method. We applied the proposed method in applicant management system and then measured the degree of coupling and cohesion with metrics such as NCOMPUC, NUCPCOM and CBCOM in it.
In the proposed method, only consistency modularity quality attributes and corresponding architectural patterns and design patterns were discussed. In future work we can optimize this method by paying attention to other quality attributes, especially opposite quality attributes such as performance and availability.

Reference

[1] L. Bass, P. Clements,R. Kazman,Software Architecture in Practice, vol. 2. Addison Wesley, Boston, 2003

[2] F. Bachmann, L. Bass, R. Nord. Modifiability Tactics. Technical report,2007

[3] www.msdn.microsoft.com/en-us/library/ee658117.aspx

[4] H. Almari. Investigation of the relationship between software patterns and quality attributes. Research Project,2009

[5] F. Buschmann. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.Chichester, NY:Wiley and Sons, 1996

[6] P. Johan, H. Holmberg.On the Modularity of a System. Master's thesis, Center for technology Studies,2010

[7] E.Gamma, R. Helm, Johnson, R., M, J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, vol. 2400. Addison Wesley, Harlow, 1995.

[8] K.Knoernschild. java Application Architecture: Modularity Patterns with Examples Using OSGi, 2012

[9] T, S. Albin, Art of Software Architecture, vol. 1. John Wiley And Sons Ltd, New York , 2003

[10] www.shapingsoftware.com/2008/08/10/architectural-styles-patterns-and-metaphors. 10 August 2008

[11] D. Galin. Software Quality Assurance From Theory to Implementation, vol. 1.Addison Wesley, Harlow, 2003.