New Modular Software Development Principles, a decentralized approach

GholamAli Nejad HajAli Irani

Faculty of Engineering, University of Bonab
Bonab, 5551761167, East Azerbaijan, Iran

Abstract

Modularity is a critical issue in large-scale software systems. For example more than 1200 Content Management Systems (CMS) have been developed as yet and any CMS consists of numerous modules and lots of modules have been developed and they cannot use modules of each other. It is due to lack of modularity parameters such as extensible, modifiable and flexible etc.
In this paper, new modular software development principles have been provided. To obtain this aim, firstly all problems of existing approaches have been investigated and categorized as the requirement list. Secondly, for solving these requirements, three new modular principles have been provided. Then two case studies have been investigated to show the applicability of provided principles. Finally, to evaluate provided principles, is shown that new principles can cover all disadvantages of existing approaches.
New provided principles can be used in any scopes of information systems such as Service Oriented Platforms and any large-scale modular software.

Keywords: *Modular Software Architecture, Quality Attributes, Object Oriented Design.*

1. Introduction

Modularity as an object oriented principle helps to have extensible, modifiable and flexible software. To have maximum quality of modularity, maximum amount of cohesion and minimum amount of coupling are needed [1]. Based on [1], the biggest problem in modular development is decomposability problem, means that decomposing each system to parts that are independent from each other is so difficult [1]. So the big trade-off has been made between being modular and modular decomposition. Therefore quality of modularity may be affected.
As increasing variety and complexity of software systems, some methodologies have emerged to overcome their complexity, such as Structural or Process-Centered methodologies, Object-Oriented methodologies and Agent-Oriented methodologies etc.
On the other hand, as increasing the scale and complexity of software systems, two thought schools have been used. The Centralized approach and The Decentralized approach. Each methodology can use Centralized thinking or Decentralized thinking or both of them. In the centralized approach, the critical and common parts of data or functionality have been collected in the central part called the Core or Kernel and the major functionalities of system have been performed or managed by the Core.
In most methodologies and software centralized approaches have used. As increasing the scale and complexity of software, scale and complexity of Core is increasing as well. Then management of Core is turned to a big problem. Therefore as increasing scale of Core, the Core can turn a GOD module [2]. Therefore quality of modularity may be affected.
The number of software companies is increasingly being larger. So, nowadays, there are some various solutions and patterns for any given problem. For example for Content Management Systems (CMS) [3] in Web Applications, more than 1200 CMS have been provided [4] and each CMS consist of some module [3]. For example, Drupal has more than 8700 modules [5].
Therefore, for CMS domain, there are some variety of solutions and same implementation of them. For another example there are more than 30 Accounting Systems and each of them consists of some modules [6]. Similarly, there are some same implementations and same solutions for Accounting Systems area. Likewise, this problem exists in some scopes of software systems. Therefore quality of modularity may be affected.
In this paper, to reach a highest modularity, new modular principles have been provided based on decentralized approach and using object-oriented principles and heuristics. To obtain this aim and in order to solving the modularity problem and increase the quality of modularity, following steps have been provided:

1. To investigate and categorize the problems of current approaches.
2. To analyze and investigate different aspects of modularity
3. To provide new modular principles to solve modularity problems.
4. To present a case study to show the abilities of provided principles.

5. To prepare guidelines for using of provided principles.

2. Problems of current approaches

Considering that there are several patterns for any problem in software engineering, increasing the scale of software, leverages the scale and complexity of Core in turn. Therefore Core turned to be a God-Class [2].
There are many heuristics to manage and decrease the complexity of God-Classes [2]. But in general, with increasing the scale of software, the complexity of Core increases as well and the modifications and extensions on Core may affect other modules. All of these disadvantages originate from the centralization of Core. In the remaining, the problems and disadvantages of centralization have been investigated and categorized.

Req1.1: Simple and small-scale modules have to obey the current patterns in Core. So the time required to develop them will increase and the dependency between modules and Core will increase as well. Therefore installing small-scale modules in other systems will be harder and hence portability, integrability and reusability of modules decrease and finally cause to decrease the modularity of modules.
Req1.2: Generally speaking, the patterns placed in Core are not complete certainly and they may not be suitable for developing large-scale modules. So, we may be forced to develop a new pattern in the development of large-scale modules. So, similar Req1.1 portability, integrability and reusability of modules decrease.
Req1.3: Modules are not free in selecting their patterns, but instead they are forced to use the Core's patterns. So, in order to use a written module in another system, we must change all connecting protocols. Therefore the portability and integrability of modules decrease. Due to Req1.1, Req1.2 and Req1.3, in the centralized approaches written modules for a Core are deeply depending on Core. So each modification in Core may affect all modules.
Req2: All approaches, patterns or management mechanisms of Core may vary with the modules. So modules and the Core itself don't need to know their mechanisms (so called Big Picture). Therefore if Core and modules be aware of the structures and patterns of each other, the security of system may be compromised and it may oppose the encapsulation principles of object oriented principles.
Req3: Due to the centralized thinking and collecting common parts and codes in Core and the dependency of modules on Core, performing a Unit Test on modules may be hard as the quality of testability is decreased.
Req4: Due to the centralized thinking and collecting common codes in Core and the dependency of modules on Core, each extension and modification in the Core my affect all modules of system. So, the modifiability and extensibility of modules are decreased.
Based on the centralized approach, module development performs quickly. But centralization has some other problems which finally reduce the modularity of modules and system. All these problems besides the affected quality attributes are shown in table 1.

Table 1. Quality attributes affected by the Centralized approach.

	M	T	I	P	R	S	LEGEND
Req1.1			x	x	x		M: Modifiability;
Req1.2			x	x	x		T: Testability;
Req1.3			x	x			I: Integrity;
Req2						x	P: Portability;
Req3		x					R: Reusability;
Req4	x						S: Security.

3. Modular Principles

Surveying the problems of table 1, some modular principles are needed which will help to develop new architectures to solve the current problems. In this section, five new principles have been provided to obtain maximum quality of modularity.
In [1] five rules and five fundamental principles have been provided for modular software. But the principles have been defined in a high abstraction level and are more general. But more detailed principles are required.
Based on the object oriented principles and heuristics [2] and the five modular principles in [1], three strategies can be identified to help us to approach the main goal that is to achieve the highest modularity. These are as the followings:

1. To minimize the Core: with the decentralized approach and to decrease the scale of Core, dependencies of modules on Core are lowering as well. Then modifications and extensions in Core may have minimum effects on modules. With this strategy the complexity of Core can be distributed among modules.
2. To minimize the modules inter-relationships: to increase the independency of modules, that is; such structures and architectures must be suggested that minimize the relationships and direct access among modules. Therefore the modularity of modules increases.
3. To standardize the external visibility of all modules and the Core per se. All modules and Core and their relationships have to use standard interfaces. These

standards should be defined for any given information system.

3.1 Decentralize Principle

The most important problem in the centralized approach is God-Classes [2] and with increasing the scale of software, the scale of Core increases as well and Core turns to be a God-Module.
God-Modules can be defined as similar God-Classes which are so large scale in data and functionality compared with other modules. There are several object oriented heuristics for the management of GOD-Classes and other object oriented problems [2]. But they are about classes and objects.
Everything can be an object [7], so a module can be considered as an object. Then object oriented heuristics can be applied on the modules to overcome the complexity of God-Modules in the modular software. For example by analyzing the concepts of encapsulation principle [7] and object oriented heuristics H2.1 and H5.3 from [2], we can say that each module must manage all its data by itself and no part of the program has right to access any module's data, except with their permission.
Data of modules can be files, database records or their web services etc. For example, a module can have several files. Each platform for this module should hold the files so that other modules could not access to those files. Based on the encapsulation principle, each module must be able to set Private, Public, Protected and other tags to its own data in any level of abstraction. Therefore, new modular architecture may arise such as Modular File Access System, Modular Service Access System, and Modular Data Access System and so on.
So, to gain maximum quality of modularity the first principle has been provided based on these points:

1. In order to decentralize approach, each module should perform its functionalities itself.
2. To resolve God-Modules of software [2] and distribute Core complexity among modules.
3. To reduce dependencies between Core and modules [1], Core must be minimizing.
4. Due to H2.8, H2.9, H2.10, H3.1 and H3.2, each module should perform its functionalities itself.

Finally, the first principle is as given below:
P1: All functionalities of each module have to be rendered with and within it.

3.2 Pure Modular Principles

To have maximum quality of modularity, modules coupling should be minimized and modules must have minimal dependencies. To capture this issue, the architecture of human body can be used. One of the most important points in the architecture of human body is that there are not any direct relationships among the modules of human body and all relationships are presented by the blood vessels and the modules do not know about the other modules' functionalities. There are some standard compounds like Hormones in blood which modules can either consume or produce (some of) them.
Accordingly, to obtain maximum modularity, the structure can be used that there is no direct relationship among the modules and modules know nothing about each other and just use standard compounds.
In the other hand, by the analysis of object oriented heuristics H2.2, H2.3, H2.4, H4.1, H4.2, H4.3 and H4.4 from [2] and principles of [1], all relationships and interfaces should be minimized for a robust object oriented system. At best, there would be no direct relationships among the modules and all modules just depend on standard interfaces.
Therefore, two critical points arise from this issue. Firstly the architecture can be designed in a way that modules do not have direct relationships with each other. Secondly the architecture can be designed in a way that modules do not know about other modules' functionalities. So, in order to have pure modular systems, the following two principals have been provided:
P2: There should be no direct relationship among the modules.
P3: Modules are not aware of any other modules existence, functionalities and data.
In order to minimize dependencies among modules and considering P2 and P3 on the decomposition approaches of methodologies, one may reach at guidelines or principles that help the betterment of modular decomposition.
All new principles are shown in Table 2.

4. Case Studies

Using provided principles in software development lifecycle, strongly affected on the architecture of software. But the Core cannot be eliminated. In any way, modules should communicate each other, but not directly and the architect of system has to use architectures that support and observe provided principles. For example, some type of event-driven architectures [8] captures some aspects of provided principles. In the reminder some case studies have been provided to show the applications and advantages of provided principles.

Table 2. New provided principles.

Main Goal	Strategies	Principle
To reach maximum modularity	- Decentralize Core complexity among modules. - Decrease the dependency of modules to Core. - Beware of the creation of God-Modules.	P1: All functionalities of each module have to be captured with and within it.
	- Minimize relationships among modules - Minimize dependency among modules - Possibly help to have so modular decomposition.	P2: There should be no direct relationship among the modules. P3: Modules are not aware of any other modules' existence, functionalities and data.

4.1 Authorization

There are some patterns for performing the Authorization [9]. Nowadays in most information systems, all Authorization functionalities are performed by the Core with the centralized approach and all Authorization patterns and methods are gathered in the Core which in turn increases the module dependencies to Core. Based on P1 and the concepts of authorization [9], each module authorization is a part of its tasks. Therefore the complexity of Authorization in Core can be distributed among modules.
However, Cristian and Gabriela showed that by distributing the security functions, a more flexible architecture can be designed that would lower the costs associated with implementation, administration and maintenance [10]. But in most software and Information Systems Centralized thinking have been used.

4.2 Search-Module

Search-Module is responsible to collect search result from whole system for any given title. In most software systems, Search-Module implemented completely in the Core and all functionalities of Search-Module such as reading directly data from module's data, search optimization, authorization functionalities for requested user etc., perform into the Core.
However, based on P1, all functionalities of Search-Module should be distributed. On the other hand, based on P2 and P3, Search-Module should not aware other modules existence.
For those reasons, the architecture of Search-Module should change. Based on P1 and as regards that Search is standard functionalities of each software system, all modules should perform search functionalities within itself. When a user wants to search a title in system, this request can sent to the Core by the Search-Module. Then the Core can send the getSearch request to all existing modules. Each module is able to response to this request. So, the Core deliver search results of each module to Search-Module.
Finally, Search-Module can render all received search result and display the final result to user. The overall suggested architecture is shown in figure 1.

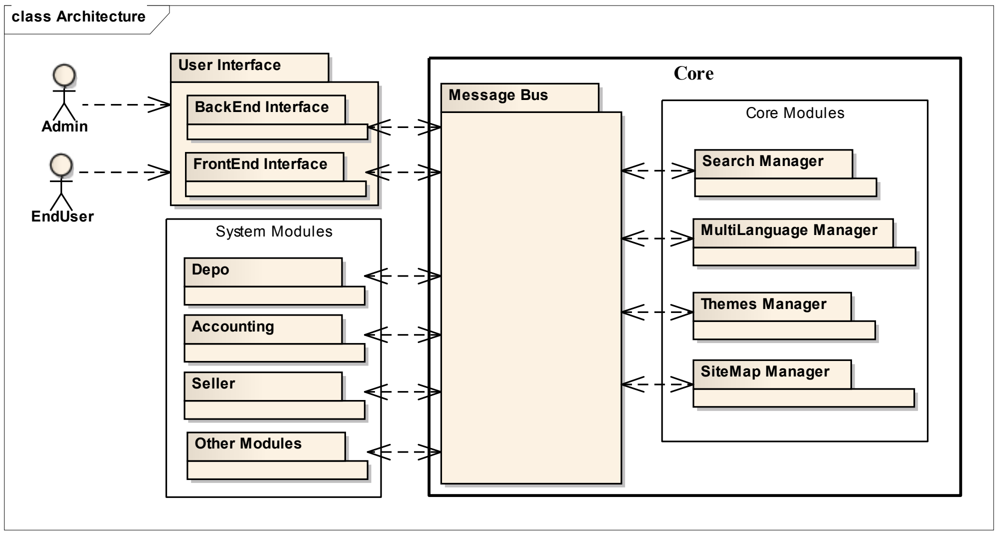

Fig. 1 Suggested architecture for Core's modules such as Search Module.

By the observing provided principles, Search-Module is not depending on system modules. Search module sent the getSearch request via Message-Bus package. Therefore, if any system module add or remove from the system, Search-Module will not change anyway. In the other hand, Search-Module should be an interface standard of a system and any system module must know about that Search-Module standard interface. So, system modules depend on Search-Module just in standard interface.
Using this method, some other operations can be performed with the same method. For example in Site Map Module, similar search module, each module should response the getSiteMap request that is sent from the Core to all modules and Site Map-Module after receive all result from the Core and render them, can display the Site Map. Other examples can similarly be Language Manager-Module, Themes Manager-Module etc.

5. Evaluation

Based on fully observing provided principles and as establishing a new standard interface for Core and modules communications, the modularity of each module and whole system were increased to higher degree. Totally, by the observing provided principles in architecture design steps or existing architecture, all the mentioned quality attributes of modularity will be improved. Table 3 shows that how categorized requirements captured by provided principles.

Table 3. Requirement list is captured by provided principles.

Description	**Captured Requirements**
Modules are independent in selecting their own patterns. They just have to consider Core's standard interface.	Req1.1, Req1.2, Req1.3
Each module can access just its contents and modules encapsulation and security are increased.	Req2
Since modules are not dependent to Core, unit test of each module can perform easily far from the Core.	Req3
Modules are free to choose their patterns and we don't need to collect all patterns in Core.	Req4

6. Conclusion

In this paper new software development principles have been provided to increase the modularity of architectures. To have maximum quality of modularity, maximum amount of cohesion and minimum amount of coupling are needed. Observing provided principles coupling among modules minimized as possible and Core complexity distribute between modules based on robust object oriented thinking and dependency between modules decrease saliently and turn existing systems to more modular systems. So module development will take extra effort than before. Although it could be a disadvantage in comparison with centralized systems, this extra effort is worth benefiting of being decentralized.
By the use of provided principles software architects can improve some quality attributes of their architecture such as modifiability, extensibility, portability etc. New provided principles can be used in other scopes of information systems such as Service Oriented Platforms and any large-scale modular software.

References

[1] B. Meyer, Object Oriented Software Construction, Second Edition, Prentice Hall International Series in Computer Science, 1994.
[2] A. J. Riel, Object-Oriented Design Heuristics, Addison Wesley, 1996.
[3] B. Boiko, Content Management Bible, 2nd Edition, Wiley Publishing, Inc., Indianapolis, Indiana, 2005.
[4] The Content Management Comparision Tool, available at http://www.cmsmatrix.org.
[5] Drupal, Open Source CMS, available at http://Drupal.org/Project/Modules.
[6] Comparisions of Accounting Softwares, available at http://en.wikipedia.org/wiki/Comparison_of_accounting_software.
[7] G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, K. A. Houston, Object-Oriented Analysis and Design with Applications, 3rd Edition, Addison-Wesley Professional , 2007.
[8] G. Mühl, L. Fiege, P. Pietzuch, Distributed Event-Based Systems, Springer-Verlag Berlin Heidelberg, 2006.
[9] Microsoft Corporation, Building Secure Microsoft ASP.NET Applications Authentication, Authorization, and Secure Communication, Microsoft Press, 2003.
[10] C. Opincaru, G. Gheorghe, Service Oriented Security Architecture, 2008.