FINDING SUITABLE MODULARITY TECHNIQUE FOR OSS PROJECTS

Andi Wahju Rahardjo Emanuel¹, Retantyo Wardoyo², Jazi Eko Istiyanto², Khabib Mustofa²
¹Bachelor Informatics, Faculty of Information Technology, Maranatha Christian University
Jl. Prof. Drg. Suria Sumantri 65 Bandung, INDONESIA
²Department of Computer Science and Electronics, Gadjah Mada University
Jl. Sekip Utara, Sleman, Yogyakarta, INDONESIA
email: ¹awreman@gmail.com, ²{rw,jazi,khabib}@ugm.ac.id

ABSTRACT

Open Source Software (OSS) Projects currently become the trend in Computer Science / Information Technology. The previously assumed to be a chaotic and ‘bazaar’ style of software development has proven itself to be able to produce some of high quality software applications such as Linux Operating System, Openoffice.org productivity suite, Apache web server, and many more. These successes have triggered many studies to determine OSS Project success factors, and one of the identified key success factor is the modularity of the OSS applications. The modularity of software system itself also receives many interests from researchers. Modularity of software system could be achieved by some techniques such as Object-Oriented Design Pattern (OODP), Aspect-Oriented Programming (OAP), Feature-Oriented Programming (FOP), Service-oriented Architecture (SOA) and Rule-Based Programming (RBP). We believe that not all of these techniques are suitable for OSS Projects. This paper presents the analysis of the suitable techniques in achieving modularity for OSS Project by considering the characteristics of OSS Projects and the ideal properties of modular software system. The method used in this research is by defining the characteristics of ideal modular software system and then comparing the features in five techniques in achieving modularity to find the best approach for OSS Projects.

Keywords: Open Source Software, modularity technique, Object-Oriented Design Pattern, Feature-Oriented Programming

1 INTRODUCTION

Open Source Software (OSS) Projects currently become the trend in Computer Science / Information Technology. The previously assumed to be a chaotic and ‘bazaar’ style of software development has proven itself to be able to produce some of high quality software applications comparable to the proprietary / commercial software i.e. Linux Operating System, Openoffice.org productivity suite, Apache web server, and many more. These ‘methodology’ , which is seemingly the violation of the already established Software Engineering discipline, has triggered many studies to determine the project's success factors, and one of the identified success factors is the fact that most of successful OSS Projects are modular [8][10].

The evolution of OSS Project will progress into certain phases in which also correspond to the evolution of the OSS Communities. The project initially started by a project initiator who registered the project into OSS web portal i.e. sourceforge.net.
If the project receives enough attention from other developers, it will evolve into more complex project in which the growth of the source code is in super-linear rate [3]. In this evolution phase, the active role of project administrator is important to reduce the increasing complexity by performing anti-regressive work [5]. The OSS Project is then reaching its stable point which is indicated by the growth of the source code to about 10% in each release. At the end, the project will reach a phase out phase if there is no developer interested to contribute to the project.

The structure of the OSS Communities themselves are also evolve in parallel with the evolution of the software system. From the initial phase with only a single developer, the project will become small to medium project (< 100 developers) in which the actual roles in the project are only two roles [21]. When reaching stable point phase, the structure of the OSS Communities are in onion layer [16] with 7 or 8 distinct positions [16][7]. The most critical stage of the co-evolution of the OSS system and OSS Communities are during the evolution from initial phase into evolution phase in which new developers should be acquired to participate in developing the software system.

2.2 Software Modularity

Modularization involves breaking up of an software system into smaller, more independent elements known as module. Booch has defined modularity as the property of a system whose modules are cohesive and loosely-coupled [13]. Fenton stated that Modularity is the internal quality attribute of the software system [13]. It is also known that modularity is directly related to software architecture, since modularity is separation of a software system in independent and collaborative modules that can be organized in a software architecture [15]. Modular software has several advantages such as maintainability, manageability, and comprehensibility [14].

There are four attributes closely related to modularity in software system which are coupling / dependency, complexity, cohesion, and information hiding. The first attribute is coupling / dependency which consist of direct / syntactic which can be achieved through composition, method signatures, class instantiations, and inheritance[6]; and semantic or indirect coupling[12]. The second attribute is complexity that can be measured by using software metrics such as McCabe’s Cyclomatic Complexity, Halstead’s Software Metrics, Function Points, or Physical Size such as LOC or SLOC [20]. The third attribute is cohesion which measure the integrity of the code inside each of the module. The term used to qualitatively measure cohesion are high cohesion or low cohesion. The last attribute is information hiding [6] which involves hiding the details of implementation from external modules.

The first approach is to define the qualitative measure of modularity. These attributes is then used as the benchmark against the available techniques to achieve modularity. The techniques that are suitable for OSS Projects are the preferred techniques. To have an ideal modular software system, the system should have the following attributes:

- Low coupling / dependency [4]: minimization or standardization of coupling / dependency e.g. through standard format i.e. published APIs [1], elimination of semantic dependencies, etc.
- Low complexity: hierarchy of modules that prefers flatter than taller dependency [13][1].
- High cohesion: high integrity of the internal structure of software modules which is usually stated as either high cohesion or low cohesion.
- Open for extension[4]: capability of the existing module to be extended to create a more complex module.
- Close to modification [4]: avoid changing already debugged code. The creation of new modules should be encourage using available extension and not modifying the already tested module.

There are currently several techniques to achieve modularity in software system. These techniques are [4]:

- Object-Oriented Design Patterns (OODP): using combinations of several software design patterns. Software design pattern is first introduced by Gang of Four [9].
- Aspect-Oriented Programming (AOP): Separation of cross-cutting concerns called aspect from the already functional data and function logic. This techniques is first proposed by Kiczales, et. al [11].
- Feature-Oriented Programming (FOP): programming techniques by creating many basic features. The more complex features or functions are created by combining these basic features in a stacked-layered format [17].
- Service-Oriented Architecture (SOA) [4]: programming techniques by using collection of independent and loosely coupled services
that interact with each other via standardized message exchange format such as XML.

- Rule-Based Programming (RBP) [19]: programming techniques based on strategies. These strategies are created from many rules or constrained. The modification of system's behavior is through the modification of rules and strategies.

The strategies of these techniques to achieve the five main attributes of modularity (low coupling, low complexity, and high cohesion, open for extension, close to modification) are studied and analyzed to find which techniques are suitable for OSS Project.

2.3 Strategies in Each Modularity Techniques

These five preferred attributes of modularity are then analyzed against these five techniques to achieve modularity. Each of these techniques have different strategies in addressing the attributes of modularity.

Modularity Strategy in OODP

The modularity strategy OODP is shown in table 1.

Table 1. Modularity Strategy for Object-Oriented Design Pattern

<table>
<thead>
<tr>
<th>Attributes</th>
<th>OODP Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Coupling</td>
<td>Separation of implementation and data by using Observer Design Pattern to reduce tight coupling among modules.</td>
</tr>
<tr>
<td>Low Complexity</td>
<td>Reducing the complexity by applying Strategy, Composite and Decorator Design Pattern, so that complex functionality is aggregated from collections of simple functionalities of lower level modules / classes.</td>
</tr>
<tr>
<td>High Cohesion</td>
<td>Creation of new classes or modules by using Abstract Factory, Factory Method, or Prototype design patterns. The simplicity of the simple function will result in high cohesion.</td>
</tr>
<tr>
<td>Open for Extension</td>
<td>Reduction of interface in module interconnection by using Façade or Proxy Design Pattern.</td>
</tr>
<tr>
<td>Close to Modification</td>
<td>Using Decorator and Composite Design Patterns in making complex function as collections of simple functions.</td>
</tr>
</tbody>
</table>

Table 1 shows that in OODP, the main feature is the combination of software design patterns to achieve the desired modular software system.

Modularity Strategy in AOP

The modularity strategy in AOP is shown in table 2.

Table 2. Modularity Strategy for Aspect-Oriented Programming

<table>
<thead>
<tr>
<th>Attributes</th>
<th>AOP Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Coupling</td>
<td>Minimize dependencies by eliminating semantic dependencies.</td>
</tr>
<tr>
<td>Low Complexity</td>
<td>Application of three layers implementation: application layer, aspect layer, data layer.</td>
</tr>
<tr>
<td>High Cohesion</td>
<td>Separation of the cross-cutting concerns (aspect) from functional data to create cohesive source code.</td>
</tr>
<tr>
<td>Open for Extension</td>
<td>Enforcing direct dependencies through published APIs.</td>
</tr>
<tr>
<td>Close to Modification</td>
<td>Emphasize on the easy modification of aspect layer without affecting the application and data layer.</td>
</tr>
</tbody>
</table>

Table 2 shows that the separation of cross-cutting concerns called aspects are the main feature in AOP. The ability to easily modify these aspects determine the modularity of the software system.

Modularity Strategy in FOP

The modularity strategy for FOP is shown in table 3.

Table 3. Modularity Strategy for Feature-Oriented Programming

<table>
<thead>
<tr>
<th>Attributes</th>
<th>FOP Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Coupling</td>
<td>Standardized format of coupling /dependency among features to allow the combination of basic features into more complex features.</td>
</tr>
<tr>
<td>Low Complexity</td>
<td>Hierarchy of features in layered-stacked structure, from minor features up into higher level functions.</td>
</tr>
<tr>
<td>High Cohesion</td>
<td>Each minor features is created for a single simple function which to achieve high cohesion.</td>
</tr>
<tr>
<td>Open for Extension</td>
<td>Higher level function is the combination of lower level features using standardized format of interconnection.</td>
</tr>
<tr>
<td>Close to Modification</td>
<td>Modification of basic feature will only affect the more complex features that using that basic feature.</td>
</tr>
</tbody>
</table>

Table 3 shows that the main feature of FOP are the capability to create the basic feature and the capability to create more complex feature by
combining two or more of these basic features. The easy creation of more complex feature from combination of basic features determine the level of modularity in the software system.

**Modularity Strategy in RBP**

The modularity strategy for RBP is shown in table 4.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>RBP Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Coupling</td>
<td>There is no message passing among modules, communication only through shared data</td>
</tr>
<tr>
<td>Low Complexity</td>
<td>The structure of the system is centralized to the shared data, resulting in a simple architecture with no hierarchy.</td>
</tr>
<tr>
<td>High Cohesion</td>
<td>Each modules are independent to each other since the only communication is to the shared data.</td>
</tr>
<tr>
<td>Open for Extension</td>
<td>Addition of rules will change the strategy of the program.</td>
</tr>
<tr>
<td>Close to Modification</td>
<td>Modification of rules are allowed in which it will also affect the strategy.</td>
</tr>
</tbody>
</table>

Table 4 shows that the main feature of RBP is the repeated application of rules to the shared data. The easy modification of rules defines that level of modularity of the software system.

**Modularity Strategy in SOA**

The modularity strategy for SOA is shown in table 5.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Service Oriented Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Coupling</td>
<td>Collection of services interact with each other using standardized format of message exchange such as XML. Each service is self-contained and loosely coupled</td>
</tr>
<tr>
<td>Low Complexity</td>
<td>There exist a management layer that coordinates all the interconnection of services.</td>
</tr>
<tr>
<td>High Cohesion</td>
<td>Each services in SOA is independent to each other.</td>
</tr>
<tr>
<td>Open for Extension</td>
<td>A new service can be added as long as it follows the standardized format of message exchange.</td>
</tr>
<tr>
<td>Close to Modification</td>
<td>Modification of a service will only affect on its own.</td>
</tr>
</tbody>
</table>

Table 5 shows that the main feature of SOA is the uniformity of message exchange using XML. The degree of interaction among these services determine the level of modularity of the software system.

3 RESULT

In order to find the suitable modularity techniques for OSS Projects, the current problems of these projects should be identified. The following are some of the problems of OSS Projects as identified by some researchers:

- Lack of formal methodology, especially in managing large and complex project[7].
- Low participation of developers due to high technical barrier [2].
- Lack of documentation due to the lack of interest in writing documentation [7].

In addressing these problems, three solutions are proposed. These three solutions are then used as the conditions that should be met by modularity techniques in order to be considered suitable to be used in OSS Projects:

- *Standardized format of communication*: there should be a common format of communication for developers of the project that should be easily understood by interested new participants of the project.
- *Lowering the technical barrier*: the high technical skill barrier that prevent new participants to contribute to the OSS Project should be lowered.
- *Reduction of management task*: The management task should be lessen especially for OSS Project administrator.

In achieving standardized mean of communication, every modularity techniques has different approaches. The software design patterns in OODP are the standard terminology that enables developers to communicate in the same language. AOP does not have specific format regarding communication media. The FOP focuses on the standard interfaces among basic features that must be obeyed by developers in order to built more complex functionality. The XML is the standard mean of communication in SOA, while in RBP the standard mean of communication is the shared data.

In attempt to lower the technical barrier in order to attract more developers, each modularity techniques also have different strategies. The similarity of software design pattern existed in OODP should make interested programmer to contribute. In AOP, however, the programmer should have better understanding in the whole
Finding Suitable Modularity Technique for OSS Projects

Andi W.R. Emanue

VIII-37

ISSN 2085-1944

software which creates additional difficulty. The scope in enterprises level in SOA also makes it difficult for new developer to contribute further. In RBP, the understanding of the overall rules and strategies will not help new programmers.

If the OSS Project is evolving into larger project, the reduction of project administrator work is necessary. In OODP, there are delegation of work in each module for groups of developer so it will reduce administrator's work. The separation of aspects from the functional logic in AOP also reduce administrator's work since the aspect layer will make it easier to accommodate changing requirement. In FOP, the delegation similar to OODP exist by giving the control into the basic feature level. In RBP, the standard formation of rules and strategies will reduce administrator's work.

The summaries of above descriptions are shown in table 6. Check mark means the approach / strategy exist, while the dash means that the approach / strategy does not exist.

Table 6. OSS Project Solution in each Modularity Techniques

<table>
<thead>
<tr>
<th>Condition</th>
<th>OODP</th>
<th>AOP</th>
<th>FOP</th>
<th>SOA</th>
<th>RBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard mean of communication</td>
<td>√</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Lower technical barrier</td>
<td>√</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reduction of management task</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

Table 6 shows that OODP and FOP are the more suitable modularity techniques for OSS Project than AOP and RBP. Both techniques address all the ideal conditions for OSS Projects to evolve into more mature project.

4 CONCLUSION AND DISCUSSION

OSS Project is currently the new approach in developing software system. The success of some OSS Projects such as Linux OS, Apache Web Server, etc. has spark many interest to study the success factors of these projects and one of the main key success factor is the fact that these OSS Projects are modular. Modularity of software system itself has been the subject of many studies. Some studies have indicated that the ideal attributes of modular software system are the low coupling, low complexity, high cohesion, open for extension, and close to modification. The approaches of each modularity strategy to achieve each of these ideal attributes are also discussed. Furthermore, there exist five modularity techniques i.e. Object-Oriented Design Pattern (OODP), Aspect-Oriented Programming (AOP), Feature-Oriented Programming (FOP), Service-Oriented Architecture (SOA), and Rule-Based Programming (RBP).

By realizing the current problems in many OSS Projects and the strategies to overcome these problems, the suitable modularity techniques for OSS Project are analyzed. The strategies to overcome the problems include the standard mean of communication, lower technical barrier, and reduction of management task. It can be concluded that OODP and FOP are the more suitable modularity techniques for OSS Projects than AOP and RBP. The future work of this research includes the possibility in combining both techniques into a Software Framework that easily used by OSS Project's initiator to start his / her project with modularity as his/her early concern.

The authors of this paper would like to thank Maranatha Christian University (http://www.maranatha.edu) that provide the funding for this research and Department of Computer Science and Electronics at Gadjah Mada University (http://mkom.ugm.ac.id) which provide guideline and technical assistance for the research.

REFERENCES


