Abstract

In this paper, an analysis of product structures in open source software (OSS) at both product level and module level is presented. At the product level, the product structures are modeled as complex networks, and the evolutionary characteristics of product structures are analyzed by using network analysis metrics. At the module level, linking mechanisms, which describe how a module is attached with other modules, are proposed. The linking mechanisms are modeled as probability functions dependent on the degrees of linking modules. A case study from an open source software project, Drupal, is presented. The evolutionary trends of Drupal product structures are analyzed and discussed. Finally, a model is presented to illustrate the effects of linking mechanisms at the module level on the product structures at the system level. The results indicate that the model built using the proposed linking mechanisms generates networks whose evolutionary characteristics are close to that of the original network.

1. Introduction

We present a network-based analysis of the structure of open-source software products and their evolution. The structure of open-source products evolves as new requirements are proposed, new modules and functions are created, and new interfaces are designed. The analysis of the product structure and its evolution is critical for gaining an understanding of fundamental similarities in the way open-source processes can be applied to a wide range of products (e.g., open-source hardware).

The approach used in this paper involves: 1) modeling the product structure as a dynamic evolutionary network where product modules are represented as nodes and dependencies are represented as links; 2) development of a quantitative model of network evolution based on module-level mechanisms to model the evolution of product structure; and 3) illustration of the model using an example of an open-source software product.
2. Review of relevant literature

Various researchers have presented both qualitative and quantitative studies on the product structure in open source software (OSS) development. For example, based on their qualitative analysis, Raymond [1] and O’Reilly [2] claim that OSS is more "modular" than proprietary software. Torvalds [3] suggests that modularity is a required property for the success of OSS development. Sanchez and Mahoney [4] analyze one major subsystem within Linux and show that the internal elements have many dependencies with external elements. Russovan et al. [5] examine ARP source files from Linux, and find out many dependencies by analyzing the direct and indirect function calls. Sangal et al. [6] and Huynh and Cai [7] demonstrate the usage of Design Structure Matrix (DSM) in the analysis of complex software structures. MacCormack et al. [8] utilize the DSM to represent the structures of open source code and explore characteristics of software structures based on two types of dependence cost: 1) propagation cost and 2) clustering cost. Structures of Linux and Mozilla are compared in terms of the two metrics. The common characteristics of the two projects are that both are architected into a small number of major subsystems. The differences are that Mozilla has a few large groups that are very tightly connected, whereas Linux comprises a larger number of small groups that are less tightly connected.

LaMantia et al. [9] analyze the structure of Apache Tomcat by using DSM and design rules. A metric called change ratio, which evaluates the substitution of modules between two versions of software, is defined as the sum of new modules added and existing modules removed divided by the total number of modules. Empirical results show that the system can be divided into two substitutable subsystems. With each subsystem, the substitution of individual modules does not affect the software architecture. Besides, by studying the change ratio, software evolution can be explained by specific design rules embodied in the modules.

Valverde and Sole [10] model open source software structures as complex networks consisting of software classes as nodes and class calls as links. They discover hierarchical small-worlds and scale-free characteristics in the class graphs. The authors show that the clustering coefficients (C) of the projects are significantly larger than their random counterparts. Moreover, the C values are independent of the system size. This is a common feature of small-world hierarchical networks. The degree distributions of these projects show scale-free property of these projects.

Milev et al. [11] analyze the change of propagation cost and clustered cost over time in Jasper and Tomcat.

3. Approach for analyzing product structures in open-source processes

As discussed earlier, the product structures are modeled as complex networks with nodes and links, where the nodes represent the modules and the links represent interfaces among modules. The interfaces are defined by Ulrich [17] for physical products as the interactions among physical components. In the
software domain, the modules are represented by functions, which realize specific features. We use function calls to define the interfaces in the product [8, 9]. For example, if function B is called within function A, then an interface is created between function A and function B. Function calls represent one of the ways of modeling the structure of software. The detailed steps of analysis are presented in Sections 3.1-3.4.

3.1. Data collection

In the first step, raw data about the product structure is extracted from the source code. The raw data consists of all the functions in the source code and the corresponding function calls. The data is used to derive information about the relationships among modules in product.

3.2. Modeling the product structure as a network

The second step is to model the product structure as a complex network. Nodes represent modules and the links represent interfaces between modules. In order to analyze the function calls, a documentation generator tool called Doxygen [18] is used. Using the software, the interfaces between pairs of modules (i.e., functions) are determined. The analysis of software structures focuses on the identification of design rules and clusters in the network.

3.3. Modeling the evolution of product structures

After modeling the product structure as a complex network, the evolutionary characteristics of the corresponding network are explored. Complex network analysis metrics [19] are employed for examining and summarizing the complex product structures over time. The metrics used in this paper are average degree, average density, diameter, and clustering coefficient. These metrics are extensively used in analyzing OSS development communities.

Average degree is the average number of nearest neighbors of vertices [20]. It is chosen because it represents the average number of modules connected to a module, and indicates the complexity of the product [21]. The degree distribution, \( P(k) \) is defined as the fraction of nodes in the network with degree \( k \) [22]. The degree distribution is important because it indicates the topology of a product structure network. For example, recent studies [10] show power-law distributions in software architecture.

Average density of a network is the average proportion of links incident with nodes in the network [19]. The average density indicates whether the modules are highly coupled or not within the product. A high average density represents that the product cannot be decoupled into modular sub-systems.

Diameter is the largest distance between any two nodes of a connected network [23]. In the product, when a module changes, the modules dependent on it may need rework [24]. The diameter of a product structure network indicates the farthest possible effect on other modules when one module is modified.

Clustering coefficient is the probability that two nearest neighbors of a vertex are also the nearest neighbors of one another [22]. Clustering coefficient indicates possible "eliques" with high connections inside and low connections outside. Prior research highlights that a high clustering coefficient is observed in various open source software project when compared to their random network counterparts [10].

3.4. Analysis of module-level mechanisms over time

The module-level mechanisms are obtained by comparing two consecutive versions of the product structure network. The mechanisms can be determined by various factors, such as the degree, the clustering [25], the transitivity [25] and the connectivity [25] of the module. In this paper, our assumption is that the linking mechanism is a probability function of the degree because the degree is a basic but key attribute of a module. The assumption is validated by discovering patterns in the plots displayed in Section 4.2.

The module-level mechanisms of network evolution are:

(a) Addition of new modules. New modules are added to address new requirements, new specifications and new features. By comparing consecutive versions of the product, the trends in the addition of new modules can be observed.

(b) Removal of existing modules. Existing modules may be removed because the existing features may no longer be needed or are replaced by new features. By comparing the consecutive versions of the product, the number of existing modules removed from a product can be determined.

(c) Linking of existing modules to new modules. After new modules are added, the existing modules are attached to new modules by new interfaces. Based on these new interfaces, the new modules can use the already-existing modules. In our model, the probability of an existing module attaching to new modules is quantified. The probability is assumed to be a function of the degree of existing modules:

\[
P(A_{e,n}) = F(K_e) \quad (1)
\]
where $A_{en}$ represents the attachment between existing modules and new modules, $K_e$ represents the degree of the existing modules. By comparing two consecutive versions of product structure network, we determine the average numbers of interfaces that are created with new modules as a function of their degree. This information is used to derive the probability in Eq. (1). A simple example is illustrated in Figure 1.

\[
A_{en} = F(K_e)
\]

where $A_{en}$ represents the attachment between existing modules and new modules, $K_e$ represents the degree of the existing modules.

\[
P(R_{e,e}) = F(K_e)
\]

where $R_{e,e}$ represents the removal of interfaces between existing modules, $K_e$ represents the degree of the existing modules.

(d) Linking of new modules to each other. We model the probability of two new modules being linked with each other given the condition that they have already finished linking with existing modules. We define the "initial degree" of a new module, which is equal to the number of existing modules it is attached to. The probability of a new module being attached with other new modules is a function of the "initial degree" of this new module and is modeled as:

\[
P(A_{n,n}) = F(K_{ni})
\]

where $A_{n,n}$ represents the attachment between new modules, $K_{ni}$ represents the "initial degree" of the new modules.

(e) Linking of existing modules with each other. We study the probability that a new interface is created between two existing modules. The probability that an existing module is attached to other existing modules is a function of the latter module’s degree:

\[
P(A_{e,e}) = F(K_e)
\]

where $A_{e,e}$ represents the attachment between existing modules, $K_e$ represents the degree of the existing modules. The relationship (i.e., Eq. 3) is determined using the approach similar to the one shown in Figure 1.

(f) Removal of an existing interface. Existing interfaces are removed in the new versions because of two reasons: i) the existing modules are removed. In this case, the existing interfaces associated with these modules are also removed; ii) the interfaces between two existing modules are no longer used. Hence, the existing interfaces are removed. We study the probability that an existing module removes its existing interface by comparing consecutive versions. The probability is a function of the degree of existing modules.

\[
P(R_{e,e}) = F(K_e)
\]

Figure 2 shows how the mechanisms are applied. An existing product network evolves to a new network by utilizing the module-level mechanisms. In Section 4, the approach is applied to an example product, Drupal. A computational model is built by applying linking mechanisms for Drupal data. Based on the execution of model, the effects of linking mechanisms on the evolution of product structures are analyzed.

4. Network-based analysis of the evolution of Drupal

Drupal [16] is an open-source content-management system which is used for the creation of community-based websites. Drupal has been under development since 2001. We analyze four different versions of Drupal software (2.0-5.0). Drupal is well developed with over 7000 community-contributed add-ons, known as contrib modules. Besides, the project has attracted more than 1000 developers. Drupal was selected
because of its maturity and the availability of code for different versions. It is emphasized that the word module, as used in this paper, refers to a function within the code.

4.1. Analysis of the structure of different versions of Drupal

Using the source code, we identify characteristics of the product structure that change over time and the characteristics that are invariant. In Table 1, the growth in the number of modules and interfaces is listed from Versions 2 through 5.

| Table 1 – Characteristics of different versions of the Drupal product structure |
|-------------------------------|--------|--------|--------|--------|
| Version | 2 | 3 | 4 | 5 |
| No. of Modules | 248 | 412 | 635 | 1018 |
| No. of Interfaces | 661 | 1304 | 2041 | 3139 |
| Average Degree | 4.4981 | 6.122 | 6.123 | 6.081 |
| Average Density | 0.019 | 0.014 | 0.009 | 0.005 |
| Diameter | 7 | 7 | 7 | 8 |
| Clustering Coeff. | 0.11 | 0.099 | 0.107 | 0.100 |

The complex network analysis metrics introduced in Section 3 are used to analyze the characteristics of product structure networks from version 2 through 5 (see Table 1). The average degree of modules shows two stages in the evolution of product structure network. The first stage is from version 2 to version 3. In this stage, the average degree significantly increases. The second stage is from version 3 to version 5, when the average degree does not change significantly. The average density reduces linearly over time. The decreasing trend of the average density is also discovered by MacCormack et al. [8] for the software structure of Mozilla. The diameter of the product structure is between 7 and 8, meaning that the longest possible path between two modules is less than 8. A large diameter causes difficulty in managing the product development because the effect of modifying one module on other modules cannot be detected immediately. In the Drupal project development, the diameter remains stable as compared to the exponential growth of interfaces.

The clustering coefficient remains constant and is close to 0.1 for versions 2 through 5. In Table 2, the clustering coefficients are compared with random graphs consisting of the same number of nodes and edges. The clustering coefficients of product structure networks are about an order of magnitude larger than the corresponding random graphs. Besides, the clustering coefficients of product structure networks are independent of network sizes, while in the corresponding random graphs, the clustering coefficients are linearly decreasing. The comparison is consistent with the conclusion drawn by Valverde and Sole [10]. In their studies, high and size-independent clustering coefficient values are observed in OSS projects.

| Table 2 - Comparison of clustering coefficient between product structure network and corresponding random graphs |
|-------------------|--------|--------|--------|--------|
| Version | V2 | V3 | V4 | V5 |
| Product structure network | 0.110 | 0.099 | 0.107 | 0.093 |
| Corresponding random graph | 0.022 | 0.015 | 0.007 | 0.004 |

The degree distribution plots for versions 2 through 5 are displayed in Figure 3. It is observed that on a log-log scale, the form of degree distribution can be divided into three parts for all versions. In the first part, the number of nodes with degrees smaller than 4 is homogenous and can be represented as a horizontal line. In the second part, the number of nodes with degrees between 4 and 20 display power-law trends, indicating a scale-free property. In the third part, there are fewer of such high degree modules, which make the distribution sparse for high degree.

The degree distribution plots indicate that: 1) the majority of modules have less than 4 interfaces, 2) with the increase of interfaces for a module, the number of the modules with the same number of interfaces decreases according to a power-law trend, exhibiting a scale-free property, and 3) the modules with more than 20 interfaces are "hubs" in the product structure network.

Similar degree distributions are also discovered by other researchers. Hyland-Wood et al. [26] show that the degree distribution of Kowari follows a linear trend when the degree is larger than 4, while displays a homogenous trend when the degree is smaller than 4. LaBelle and Wallingford [27] show similar trend in the out-degree distribution of the Debian product structure.

![Degree Distribution](Figure 3 - Degree distribution of product architecture networks from version 2 to version 5)
4.2. Analysis of the mechanisms at module level

Mechanisms (a) and (b): The data corresponding to the linking mechanisms (a) and (b) proposed in Section 3.4 are displayed in Table 3. The number of new modules added and existing modules removed, which represents module-level mechanisms (a) and (b) are obtained by comparing consecutive versions over time.

On comparing the number of new modules added, existing modules removed and the total number of modules, it is observed that about half of the existing modules are removed. The number of newly added modules is close to the total number of modules in the previous version. This demonstrates high level of activity by participants. For example, new features are
fulfilled, the outdated features are removed, the features that are useful but not efficient are replaced, and bugs are found and corrected. LaMantia et al. [9] propose change ratio, which measures the number of new classes added and existing classes removed as compared with total number of classes. LaMantia et al. detect a high value of change ratio for Tomcat-main product.

Table 3 – Modules and Interfaces created in Drupal (from version 2 through 5)

<table>
<thead>
<tr>
<th>Version</th>
<th>V2→V3</th>
<th>V3→V4</th>
<th>V4→V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of modules added (a)</td>
<td>294</td>
<td>356</td>
<td>780</td>
</tr>
<tr>
<td>Number of modules removed (b)</td>
<td>154</td>
<td>139</td>
<td>439</td>
</tr>
<tr>
<td>Interfaces created between new and existing modules (c)</td>
<td>558</td>
<td>987</td>
<td>1451</td>
</tr>
<tr>
<td>Interfaces between new modules (d)</td>
<td>530</td>
<td>350</td>
<td>1286</td>
</tr>
<tr>
<td>Interfaces created among existing modules (e)</td>
<td>51</td>
<td>140</td>
<td>151</td>
</tr>
</tbody>
</table>

Mechanism (c): The mechanism (c) proposed in Section 3.4, is the probability that existing modules are attached with new modules. It is displayed in Figure 4. The probability functions are determined by fitting linear curves in the log-log plots over time. The linear fitted trend indicates a power-law distribution indicating preferential attachment between existing modules and new modules. In Table 3, the growth of new interfaces between existing modules and new modules is listed. A linear trend is observed.

Mechanism (d): The linking mechanism (d), which is the probability that new modules are attached with each other, is plotted in Figure 5. In this figure, the probability that new interfaces are attached between two new modules, given the condition that these two modules are already linked to existing modules, is provided. From the probability plots, it is observed that for new modules with high "initial degree", the probability of attaching to a new module is high compared to those with low degree. The probability functions are determined by fitting exponential curves. In Table 3, the number of interfaces created between two new modules is listed.

Mechanism (e): In Figure 6, the linking mechanism (e) proposed in Section 3.4, which measures the probability that two existing modules are attached with each other is plotted. The probability that two existing modules are attached with each other in the new version is provided based on their existing degree. In the plots, the probability functions are fitted by exponential curves. A non-linear attachment rule is applied for new interfaces attached between two existing modules. A "winner takes all" phenomenon is observed when there are existing modules with extremely high degree compared to the rest. In Table 3, the number of interfaces created between two existing modules is listed. It is observed that from version 3 to version 4, the number of interfaces created among existing modules increases significantly. From version 4 to version 5, the number of interfaces created among existing modules increases slightly.

Mechanism (f): The linking mechanism (f), which describes the probability that an existing module removes its interfaces, is displayed in Figure 7. The probability that an existing module removes one of its existing interfaces is based on its existing degree. It is observed that linear trends from these log-log scale plots (indicating power law distribution) can be used to describe the probability density functions.

5. Computational model of the evolution of product structure

5.1. Modeling process

A computational model is built to simulate the effect of mechanisms at the module level on the evolution of product structure at the product level in the Drupal project. Three types of initial product structure networks are chosen:
1) the product structure network from version 2,
2) a random network with the same number of modules and interfaces as the product structure network of version 2,
3) a scale-free network with the same number of modules and interfaces as the product structure network of version 2.

The reason for selecting three types of initial product structure networks is to determine whether the types of
initial product structure networks also affect the evolutionary characteristics of the product structure network. Random network is used as a baseline. In the existing study of network evolution, random networks are extensively used to represent initial topology of the network. Scale-free network is used because existing studies reveal that many real-world networks (including OSS) have the scale free property. In this model, three time periods are simulated: from version 2 to version 3, from version 3 to version 4, and from version 4 to version 5. In each period, the linking mechanisms proposed in Section 3.4 are simulated based on the probability functions discussed in Section 4.2.

Figure 8 outlines the execution of the model. For linking mechanisms (a) and (b), the number of modules added and removed is displayed in Table 3. For mechanisms (c), (d), (e), (f), the number of interfaces created (removed) is based on Figures 4 through 7 respectively.

5.2. Results from the execution of the model

The structures of the networks with three types of initial networks are compared with the product structure networks from version 2 to version 5. Figure 9 displays the comparison between the characteristics of Drupal product structure and the models over time. In Figure 9, key evolutionary characteristics including a) average degree, b) average density, c) diameter, and d) clustering coefficient are displayed.

From the comparison between the Drupal Product and models, it is observed that the structure of models with three types of initial networks converge to the structure of Drupal. From Figure 9(a), it is observed that the average degree of model with initial version 2 network matches the Drupal project. Initial scale-free and random networks have different average degrees compared with the initial Drupal version 2 product structure network. However, with the evolutionary process, the values of average degree between models and Drupal product converge.

The average densities of three models are similar to the Drupal product. The diameters of models with the initial random network and the initial version 2 network are the same as the Drupal product over time. The diameter of the model with the initial scale-free network is lower than the Drupal project. However, with the evolutionary processes, the diameter increases and finally converges to the Drupal product.

From Figure 9(d), it is observed that the clustering coefficient of models with the initial scale-free network and the initial version 2 network come close to the Drupal product over time. The model with initial random network has a small clustering coefficient at the beginning, which represents the random network property. However, the clustering coefficient significantly increases and converges to the Drupal product over time. From Figure 9(e), the convergence is not obvious because at version 4, the differences between models and Drupal project are larger compared by version 3 and 5.

The comparison of degree distribution between models with three types of initial networks and the Drupal Project is provided in Figure 10. At the beginning, the differences among different initial networks are significant. The initial random network displays a Poisson distribution, while the initial scale-free network displays a power-law distribution. Both of
them are different from the initial version 2 network. With the evolutionary processes, the degree distributions of three models converge to the degree distribution of Drupal project.

From the execution of the models, we conclude that:

1) When the initial version 2 network is used in the model, the evolutionary characteristics including average degree, average density, diameter, clustering coefficient, and degree distribution are close to the Drupal product over time, with the use of the proposed linking mechanisms.

2) When the initial scale-free network or random network is applied in the model, the evolutionary characteristics are different at the beginning due to the differences in topologies. However, by executing the model with the proposed linking mechanisms, the structures of the networks from the models converge to the Drupal product.

These results indicate that the linking mechanisms can potentially be used to model the evolution of product structures in the Drupal project. In the case of the Drupal product, we found that even when the initial product structure is different, if the same linking mechanisms are applied, the evolution of product structures converges to the same structure over time. This indicates the robustness of the linking mechanism in modeling the product evolution. Additional investigations on other products are necessary for further validation.

6. Closing thoughts

The product structure plays an important role in the open source processes. Not only does it affect the efficiency of open source processes [28], but it also affects the community structure [29, 30]. Hence, it is important from a coordination standpoint to get an understanding of product structure and evolution in open source processes.

The first contribution of this paper is an approach for analyzing the evolution of product structure at the product level as well as the module level and its application to an open source product. At the product level, the objective of this analysis is to extract patterns and trends in the evolution of product structure, which can be described by key evolutionary characteristics. At the module level, the objective of this analysis is to understand the linking mechanism of a module with a certain degree.

The second contribution of this paper is a model to illustrate the effect of linking mechanisms at the module level on the evolution of product structure. The models are initiated by three different initial networks with significantly different characteristics. The linking mechanisms proposed are applied in the model. The model is executed to simulate the development of Drupal project. The evolutionary characteristics obtained from the execution of the models converge to the evolutionary characteristics of product structure of Drupal. Besides, applying linking mechanisms is potentially a robust way to model the product structure over time because the differences in initial product structures do not have a significant effect on the final product structure.

Future work to extend the analysis and modeling approach in this paper can be summarized as follows: 1) Applying the proposed approach to other open source projects so that the similarities among open source processes in the evolutionary characteristics at the product level as well as the linking mechanisms at the module level can be discovered. 2) Exploring how the changes in the linking mechanisms over time (i.e., across different versions) affect the product structures.

7. Acknowledgements

The authors gratefully acknowledge the financial support from the National Science Foundation through the CAREER grant # 0954447.

8. References


