Analysis of the Interdependent Co-evolution of Product Structures and Community Structures Using Dependency Modeling Techniques

Authors: Qize Le and Jitesh H. Panchal
DOI: 10.1080/09544828.2012.695014

ABSTRACT

Within the engineering design literature, dependency modeling techniques have been used to model the structure and evolution of products. While a product's structure has significant implications on the product's quality and the complexity of product development process, recent studies within organizational science suggest that an organization’s structure also has a direct impact on the product structures. This interdependence is particularly important in open source products developed by communities because there is no community structure imposed at the beginning of the process. Both the product structure and the community structure undergo dynamic co-evolution. Understanding this co-evolution is crucial as open-source development processes are being extended beyond software products to hardware products. To understand the interdependence between product and community structures, empirically testable hypotheses, based on the theory of socio-technical coordination, are formulated in this paper. A dynamic network based approach is proposed where products are modeled as networks of interfacing modules, and communities are modeled as networks of collaborating participants. The proposed approach for testing the hypotheses involves a) modeling the evolution of networks across consecutive versions, and b) measuring the overlap between anticipated and actual communities. The approach is presented for Drupal, which is an open-source software product. Drupal is chosen due to the availability of product and community data for different versions. Based on the analysis, it is concluded that product structures significantly influence the community structures. However, the impact of community structures on the product structures is weak. While the results are specific to the case study, the approach is general enough to be utilized for other open source software and hardware products. The approach can also be used to model the evolution of loosely coupled informal communities within traditional organizations.

Keywords: Dependency modeling, product structure, organizational structure, complex networks, evolution, open source
1 INTRODUCTION: COORDINATION AND THE RELATIONSHIP BETWEEN PRODUCT STRUCTURES AND ORGANIZATIONAL STRUCTURES

Since the development of the design structure matrix [1], dependency modeling techniques have been widely used to analyze diverse aspects of product development including product complexity, effects of changes, task dependencies, and organizational structures. One of the primary applications of dependency modeling techniques is the analysis of coordination of activities for the development of complex systems. Coordination can be defined as “integrating or linking together different parts of an organization to accomplish a collective sets of tasks” [2] or simply as “management of dependencies” [3, 4]. Coordination between activities can be achieved in various ways such as informal communication, group meetings, and development of plans and rules [3, 4]. Herbsleb and Grinter [5] categorize coordination approaches into three types: architecture-based coordination, plan-based coordination, and process-based coordination. Architecture-based coordination involves modularization of the product. Modularization reduces dependencies across modules, and hence, the need for coordinating activities associated with different modules. Plan-based coordination involves development of plans and timelines to ensure that activities such as integration can be accomplished. Process-based coordination involves developing well-defined processes, such as change management processes, to manage dependencies.

One of the widely adopted strategies to achieve coordination is the alignment of organizational structures with product structures. The rationale behind this alignment is that if there are technical dependencies between different aspects of a product, there should be corresponding communication between the individuals (or teams) working on those aspects. Coordination achieved through the alignment of organizational structures and products is referred to as socio-technical coordination [6] and the extent of alignment is referred to as socio-technical congruence [7]. Literature in organizational science suggests that the design of organizational structures based on product structures is beneficial for product development. High levels of socio-technical congruence have been shown to positively influence productivity and the rate of product development [7, 8]. On the other hand, a lack of congruence has negative effects on productivity [9, 10]. Product structures not only affect inter-team coordination within an organization but also inter-organizational relations within a supply chain, thereby affecting the structures of entire industries [11, 12].
While the socio-technical congruence literature analyzes the impacts of product structures on organizations, there is literature suggesting that organizational structure also affects the product structure. In the organizational science literature, this effect is commonly known as the Conway’s law: “any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure” [13]. As an example, MacCormack and coauthors [14] pointed out that the structures of products developed through loosely coupled communities, such as in open source product development, are different from the structures of the products developed using tightly integrated organizations. Considering the interdependent nature of the product structures and organizational structures, it has also been hypothesized that the social structure of the organization matches the structure of the product [15]. This hypothesis is known as the mirroring hypothesis [14].

Understanding of the interdependence between products and organizational structures in hierarchical organizations helps managers in establishing appropriate team interactions to manage complexity and to maximize productivity. We believe that understanding this interdependence is particularly important in product development by loosely coupled communities (such as open source) because such an understanding helps in determining the impacts of different product structures on community structures and vice versa. With the popularity of open source software, and the increasing adoption of open source processes in hardware, the understanding is valuable not only for software products but also for physical product development. Hence, our objective in this paper is to analyze the interdependent co-evolution of product structures and community structures in loosely-coupled product development communities using dependency modeling techniques.

The overall approach presented in this paper is anchored in the empirical theory [16] of interdependence between products and organizations. While a theory in engineering is a rigorous means of reasoning mathematically from axioms, an empirical theory is tested by generating hypotheses, then observing or constructing situations in which these hypotheses can be tested [16]. Empirical theories can evolve or can be replaced by other theories based on new evidence. The paper is organized as follows. Key literature on using dependency techniques for the analysis of products and community structures, and their interrelationships is discussed in Section 2. Based on the literature review, empirically verifiable hypotheses are listed. Section 3 provides the details of the proposed approach for validating the
hypotheses. Results from the application of the approach to a specific case study are presented in Section 4. Finally, closing comments are presented in Section 5.

2   REVIEW OF RELEVANT LITERATURE
2.1 Use of dependency techniques for the analysis of product and organization structures

Dependency-based techniques have been used to analyze the structures of products and organizations from different viewpoints in engineering design, organizational science, and software engineering literature. Within the engineering design literature, dependency modeling techniques have been used to analyze product modularity and its implications on design team interactions [17]. The studies have resulted in measures for complexity and modularity, and techniques for identifying the impact of product complexity on performance. Within organization science, studies are focused on the implications of different organizational structures on product development performance.

Interdependencies between products and organizational structures have been studied both in software products such as Linux and Apache [14, 18], and physical products such as aircraft engines [17] and air conditioners [11]. These studies are particularly prominent in the software engineering literature because of easier quantification of product dependencies, relative ease of gathering the data, and well-documented communications. These dependencies can be automatically extracted and network representations of products can be automatically generated. The strengths of dependencies can also be quantified by techniques such as measuring the number of times a function is called by another function. In contrast, the interfaces within physical products can be of different types, e.g., spatial, structural, energy-related, material-related, and information-related [17]. Such dependencies are generally extracted by interviewing engineers and design experts. The strengths of relationships are also more difficult to quantify in physical products. Additionally, the software development processes are generally better documented than physical product development processes due to the presence of tools such as concurrent versioning systems and bug management systems.

Within the software engineering domain, existing research is mainly focused on three aspects:

a) comparing the effects of different organizational forms on the products developed (particularly relevant is the literature that compares loosely coupled open source development communities with tightly coupled closed-source software development organizations).
b) testing the mirroring hypothesis, and

c) estimating socio-technical congruence and its impact on software development.

These three aspects are discussed in Sections 2.1.1 through 2.1.3. A summary of the key studies focused on the interrelationship between products and organizational structures is provided in Table 1.

Table 1 – A summary of the key studies on the interrelationships between products and organizational structures in software products

<table>
<thead>
<tr>
<th>Authors</th>
<th>Discussed in</th>
<th>Focus</th>
<th>Open/Closed Source</th>
<th>Product Structure</th>
<th>Organizational/Community Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merlo et al. [19]</td>
<td>Section 2.1.1</td>
<td>Evaluation of co-evolution of products and organizational structure in open and closed source projects</td>
<td>Both open and closed source</td>
<td>Nodes: Functions</td>
<td>Nodes: Individuals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Edges: Function calls</td>
<td>Links: Individuals are linked if they work on the same project</td>
</tr>
<tr>
<td>MacCormack et al. [14, 18]</td>
<td>Section 2.1.1</td>
<td>Comparison of the structures of products developed using open source and closed source approaches</td>
<td>Both open and closed source</td>
<td>Nodes: Functions</td>
<td>Organizational/community structure not modeled explicitly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Edges: Function calls</td>
<td></td>
</tr>
<tr>
<td>Colfer and Baldwin [15]</td>
<td>Section 2.1.2</td>
<td>Review of studies on validating the mirroring hypothesis</td>
<td>Both open and closed source</td>
<td>Different approaches are discussed</td>
<td>Different approaches are discussed</td>
</tr>
<tr>
<td>Cataldo et al. [7, 8]</td>
<td>Section 2.1.3</td>
<td>Evaluation of congruence between technical dependencies and work dependencies</td>
<td>Closed source</td>
<td>Nodes: Source files</td>
<td>Nodes: Individuals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Edges: Sets of files modified together</td>
<td>Links: Communication (IRC chat and modification request systems)</td>
</tr>
<tr>
<td>Sosa [20]</td>
<td>Section 2.1.3</td>
<td>Comparison of potential interactions with actual communications</td>
<td>Closed source</td>
<td>Nodes: Product modules</td>
<td>Nodes: Individuals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Edges: Dependencies identified by system architects</td>
<td>Links: Communication (identified by surveying development actors)</td>
</tr>
</tbody>
</table>

2.1.1 Comparing the effect of different organizational forms (open source communities vs. closed source organizations)

Open source and closed source product development are associated with fundamentally different organizational structures. Closed source product development is associated with formal hierarchical organizations whereas open source processes are associated with informal decentralized communities.

Recently, there have been studies comparing the products developed through open source and closed source approaches. For example, MacCormack et al. [14] analyze and compare the structures of products that fulfill the same function but are developed through different approaches (i.e., open and closed source). In [18], MacCormack et al. compare the structures and evolution of products developed using open source and closed source processes through measures such as clustered cost, and propagation cost. The authors conclude that products developed by loosely coupled (open source)
communities tend to be more modular than the products developed by tightly coupled organizations (closed source). In another study, Merlo et al. [19] compare the products and organizational structures developed using open and closed source approaches. The authors model organizational structures using centrality measures and software structures using coupling and cohesion. Based on their empirical study, the authors conclude that in closed source projects, organizational structures and product structures mutually influence each other but in open source projects, the organizational structure affects the product structure but not vice versa.

2.1.2 Testing the mirroring hypothesis

The mirroring hypothesis is that technical dependencies correspond to organizational communication patterns. The mirroring hypothesis does not imply a direction of causality between technical and organizational structures [15]. Empirical studies on the mirroring hypothesis are focused on analyzing whether there is a correspondence between technical dependencies or not. Different studies show different levels of correspondence between technical and organizational structures. Colfer and Baldwin [15] present a detailed review of literature by comparing 102 empirical studies spanning different levels of organizations including a single firm, across multiple firms, and open communities (e.g., open source). The authors conclude that support for the mirroring hypothesis is strongest in studies on single firms, less strong in studies across multiple firms, and relatively weak in open communities.

Studies on the validation of the mirroring hypothesis are not limited to software products developed by single organizations, but have been extended to physical products developed through collaborations across a supply chain. Camuffo and Cabigosu [11] analyze inter-organizational relationships between the manufacturers and suppliers. The analysis supports the mirroring hypothesis at the component level but only when the product architecture is stable. Camuffo and Cabigosu [11] and Hoetker [21] analyze the extent to which modular products are associated with modular organizations. Using an example of the air conditioning industry, Kratzer et al. [12] analyze multi-institutional communication patterns in the space industry and conclude that the informal communications (which are not captured in formal organizational structures) play a strong role in achieving coordination between different activities.
2.1.3 Estimating socio-technical congruence and its impact on software development

While the mirroring hypothesis and socio-technical coordination are both focused on the interrelationship between technical and organizational structures, there is a subtle difference between the two concepts. In contrast to the mirroring hypothesis, the premise behind socio-technical coordination [6] is that the organizational interactions should match the technical dependencies. Hence, the socio-technical coordination literature is primarily focused on exploring the effect of product structure on the organizational structure. Cataldo et al. [7, 8] introduced the concept of congruence between coordination requirements and the coordination activities carried out within an organization. Coordination requirements refer to the task dependencies due to the dependencies among different aspects of a product. The actual coordination activities are the interactions that individuals are engaged in through different means of coordination. These means of interaction include face-to-face discussions, communication via e-mails or online forums. The authors in [7] present a quantitative measure of congruence based on the comparison of matrices representing coordination requirements and actual coordination. Sosa [20] extends the analysis of congruence by analyzing not only the coordination requirements that are met but also unpredicted and unintended interactions. Studies have also highlighted the impacts of socio-technical congruence on performance. Different authors use different measures of performance. For example, Cataldo et al. [7] use resolution time as a measure of performance and show that congruence results in lower resolution times (i.e., better performance). Kwan et al. [22] use the success of software builds as a measure of performance and show that congruence is proportional to build success in continuous builds by collocated teams. Bolici et al. [23] suggest that within open source projects, even though congruence is low, the productivity is high because of implicit coordination mechanisms such as indirect communications and coordination mediated through means other than communication, such as modularity of the code.

2.2 Limitations of existing approaches

In this section, we discuss the limitations of existing approaches for validating the mirroring hypothesis (see Section 2.2.1) and measuring socio-technical congruence (see Section 2.2.2).
2.2.1 Limitations in modeling the evolutionary dynamics of networks

Existing studies on evaluating the mirroring hypothesis are based on a static view of the product and organization networks. The dynamic nature is partly addressed by calculating different network characteristics (e.g., complexity, modularity, clustering, etc.) for different versions of the networks and comparing them against each other to reveal any trends in their evolution. While this approach helps in answering whether and to what extent products and organizations mirror each other, and to what extent the mirroring is increasing or decreasing, it does not reveal the underlying dynamics that lead to the observed evolutionary characteristics. Without an analysis of the evolutionary dynamics, it is difficult to understand the causal effects leading to the mirroring hypothesis, specifically, the effect of products on organizations and the effect of organizations of products. Hence, there is a need to analyze the interdependence between mechanisms of product network evolution (e.g., addition of new product modules) and community evolution (e.g., new communications between individuals) for a better understanding of the mirroring hypothesis.

To address this limitation, we focus our attention on the network evolution mechanisms, specifically the addition of nodes and links between consecutive time-steps. The assumption of the interdependence between products and communities (i.e., the products influence the communities and the communities influence the products) leads to the following hypotheses:

<table>
<thead>
<tr>
<th>Hypothesis 1.1:</th>
<th>If the number of interfaces of product modules increases, the number of new communication links between people working on those modules also increases.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis 1.2:</td>
<td>If the number of a participant’s communication links increases, the number of new interfaces of corresponding modules also increases.</td>
</tr>
<tr>
<td>Hypothesis 2.1:</td>
<td>If new interfaces are created between two modules, then corresponding communication links are also created between individuals working on those modules.</td>
</tr>
<tr>
<td>Hypothesis 2.2:</td>
<td>If two people communicate with each other, then the corresponding modules also develop interfaces.</td>
</tr>
</tbody>
</table>

These hypotheses are focused on mechanisms of network evolution, specifically the addition of links representing module interfaces and communications between individuals. Hypotheses 1.1 and 1.2 are focused on the impacts of new links of a product module and an individual respectively whereas Hypotheses 2.1 and 2.2 are focused on the impacts of links between pairs of product modules and pairs of individuals respectively (see Table 2). Hypothesis 1.1 is validated if an increase in the number of
interfaces (dependencies) of product modules is accompanied with an increase in the communication between individuals working on that module. Similarly, Hypothesis 2.1 is validated if an increase in the number of interfaces between two product modules results in an increased communication between the sets of individuals working on those modules. If Hypotheses 1.1 and 2.1 are valid, it indicates that the product structure impacts the community structure. Similarly, the validity of Hypotheses 1.2 and 2.2 indicates the impact of community structure on the product structure. To validate these hypotheses, our approach is to analyze the changes in product and community networks between consecutive time-steps for a specific open source community. This approach is inspired by recent advances in network science and is particularly applicable to the highly evolutionary open source communities. The details are provided in Sections 3.1 and 3.2.

Table 2 – Different aspects corresponding to the hypotheses

<table>
<thead>
<tr>
<th>Single Nodes</th>
<th>Product → Community</th>
<th>Community → Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairs of Nodes</td>
<td>Hypothesis 1.1</td>
<td>Hypothesis 1.2</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 2.1</td>
<td>Hypothesis 2.2</td>
</tr>
</tbody>
</table>

Removal of nodes and links also represent mechanisms through which networks evolve. However, we have not addressed removal of nodes in the paper because identifying the relationship between the removal of product nodes and the removal of communication links is challenging. Creation of new links between individuals is easier to identify than removal of links because if two individuals have not communicated during a given time frame, it does not necessarily mean that the link between them has been broken. Similarly, it is unclear how long an individual has to be inactive before he/she can be considered “removed” from the project. Hence, gathering the data about removal of communication links and individuals from the project is difficult. Due to these challenges, we focused only on the addition of nodes and links in this paper.

2.2.2 Limitations in the calculation of socio-technical congruence

Existing approaches for measuring socio-technical congruence are limited when applied to open source communities in that existing approaches compare the information exchange between teams but such teams are not well defined in loosely coupled open source communities. Hence, the congruence measures are calculated by analyzing the direct interactions between individuals. In open source
processes, the number of individuals is generally very large. Many participants work on different aspects of the code, due to which, the values of socio-technical congruence are very small. But this does not necessarily mean that coordination is not taking place. Additionally, as the number of participants increases, the corresponding coordination requirements grow at a faster rate. Hence, even though the actual coordination between participants increases, the socio-technical coordination calculated according to [7, 8] reduces. These challenges are illustrated in Section 4.2.3.

To address these limitations, we present an alternate approach to measuring congruence based on network clustering and cluster comparison. In the proposed approach, instead of comparing the networks at the individual node level, the networks are first clustered into communities and the resulting clusters are compared with clusters anticipated based on the links in the product network. The approach is based on Simon’s notion of near-decomposability and modularity of complex systems [24]. The assumption that the products and communities mirror each other leads to the following hypothesis:

**Hypothesis 3:** The communities get increasingly aligned with the anticipated communication patterns.

The approach to validate this hypothesis is discussed in Section 3.3 and presented for a sample project in Section 4.2.3.

### 3 Analysis Approach

#### 3.1 Modeling products and communities as networks

In this paper, product structures and communities are modeled as networks. A product can be decomposed into sub-systems or components, which implement specific functions. Hence the sub-systems (components) can be viewed as nodes in a network, while the spatial, structural material, energy, and information dependencies between them can be viewed as links [9]. In software products, different levels of abstraction including class level [25, 26], file level [18, 27], and function level [28] have been used to model products as networks. In file-level abstraction, the dependencies between files are determined by function calls across files.

In community networks, the nodes represent individuals and the links indicate communications between individuals. Different types of communications have been used to model links between individuals. For example, Cataldo and Herbsleb [29] use online chat logs and modification requests to
represent communication links. Ducheneaut [30] uses emails and Concurrent Versioning System (CVS) logs to identify communications between individuals. Sack et al. [31] derive the communication links among individuals from emails. Wagstrom et al. [32] use a bloglinks approach, where communications between two individuals exist if there are links from one’s weblog to another. Other approaches to identify communications between participants include internet relay chat (IRC) [29, 33, 34], e-mail communication [35], questionnaires and surveys [20, 36].

In addition to the links between subsystems in product networks and links between individuals in community networks, links can also be established between these networks representing the “participation” of an individual on a product module. The links between individuals and products result in a bipartite network. Such bipartite networks have been used by Cataldo et al. [8], Ducheneaut [30], Madey et al. [37], Ducheneaut [30], and Xue et al. [38, 39] to describe task assignment to participants. Based on the bipartite network linking the product modules and the individuals, we define

i. “module set” of an individual as the set of modules that a given individual works on, and

ii. “participant set” of a module as the set of individuals linked to a module.

In this paper, we consider all three types of links (individual-individual, module-module, and individual-module) described above to generate hybrid network models representing products, communities and participation simultaneously, as shown in Figure 1. The communication links between individuals and the interfaces between modules are weighted to indicate the amount of communication and the strength of interfaces, respectively. The participation links are not weighted.

![Figure 1 – A hybrid network model to represent product structure, community structure and participation](image)

3.2 Analysis of product and community evolution in consecutive versions

We propose two techniques for validating the hypotheses listed in Section 2.2. Both techniques compare consecutive versions of products and communities to determine the evolutionary characteristics.
The first technique is proposed to validate hypotheses 1.1 and 1.2. To validate H1.2, we select each participant in turn and determine:

a) the number of newly created communication links of that participant with other participants, and

b) the number of newly created interfaces between modules within the participant's module set and the modules outside the modules set.

The newly created links are illustrated in Figure 2. In this figure, participant A has two newly created relationships and the corresponding module set has two newly created interfaces. Hypothesis 1.2 is valid if the number of newly created communication links increases with an increase in the number of newly created interfaces of corresponding module sets. Similarly, for Hypothesis 1.1, we determine the relationship between the number of new interfaces of modules and the new communication links of the corresponding participant sets.

![Figure 2 – Extracting the relationship between the number of newly created links of participants and newly created links of modules associated with that participant (for Hypothesis 1.2)](image)

The second technique is employed to validate hypotheses 2.1 and 2.2. In order to validate hypothesis 2.2, we check each pair of participants and determine the number of new communications between the pair. Then we determine the number of newly created interfaces between the modules sets that belong to these two participants, as shown in Figure 3. In the figure, participants A and B are checked as a pair. A new communication link is built between them and two new interfaces are created between the corresponding module sets. Hypothesis 2.2 is valid if an increase in the communication between participants A and B is associated with new interfaces between the corresponding module sets. Note that the communication links between individuals and the interfaces between modules are weighted. Therefore, an increase in the weight of a communication link refers to a new communication between the corresponding individuals. Hence, an increase in weights of the links is also accounted for in validating the hypotheses.
Similar technique is applied to validate hypothesis 2.1. The results from the analysis of consecutive versions of the products are presented in Section 4.2.

![Diagram](image)

Figure 3 – Extracting the relationship between the number of newly created links between a pair of participants and the newly created links between corresponding module sets (for Hypothesis 2.2)

### 3.3 Cluster comparison

To validate Hypothesis 3, we propose an approach based on cluster comparison within which we determine the communities *anticipated* based on the product dependencies and compare them with the *actual* communities. If there are significant dependencies among a set of product modules, it is expected that the participants working on those modules will closely collaborate with each other, and hence cluster together as communities within the network of individuals. Hence, the anticipated communities can be derived based on the knowledge of the product network and the participation links. This is in contrast to the socio-technical congruence approach where anticipated communication links are compared with the actual communication links between participants.

To determine the actual communities, we use clustering algorithms on the network of individuals linked through communication links. The anticipated communities are determined in two ways – a) utilizing clustering algorithms on the product networks and determining the corresponding clusters of participants contributing to different product clusters, and b) utilizing bipartite clustering algorithms on the bipartite network (i.e., individuals and modules linked through participation links) and identifying communities of participants based on the participation links only. The cluster comparison approach is illustrated in Figure 4. In this figure, clusters are identified in the product network, which are then mapped into clusters of individuals based on participation links. The clusters of individuals obtained in this manner are anticipated communities. These anticipated communities are compared with those directly obtained by clustering the community network. The comparisons of clusters are made for each version to determine the evolutionary characteristics. In summary, the cluster comparison approach involves two
main steps – extraction of clusters and comparison of clusters. The specific details of these steps are discussed in Sections 3.3.1 and 3.3.2 respectively.

![Figure 4 – Cluster comparison approach](image)

### 3.3.1 Extraction of clusters

A number of techniques have been proposed in the literature to identify clusters within networks. In this paper, we use the modularity-based method proposed by Newman-Girvan [40, 41] to partition the community network. Using this method, a modularity function, \( Q \), is determined for each partition:

\[
Q = \frac{1}{4m} \sum_{ij} \sum_r \left( A_{ij} - \frac{k_ik_j}{2m} \right) S_{ir}S_{jr}
\]

where \( m \) is the total number of links; \( A_{ij} \) is the number of links between node \( i \) and node \( j \); \( k_i \) and \( k_j \) are the degrees of the node \( i \) and \( j \). \( S_{ir} \) is equal to 1 if node \( i \) belongs to group \( r \). \( S_{jr} \) is equal to 1 if node \( j \) belongs to group \( r \). The best partition of the network is the one that maximizes modularity across all possible partitions. To reduce the computational effort in finding clusters, we utilize the Louvain method [42] to identify a good approximation of the partition in unimodal networks, and the modularity-based method proposed by Barber [43] for bipartite networks. In this paper, we choose the modularity-based community detection methods because of three reasons:

- they generate good quality clusters at low computational cost.
- the user does not need to specify the number and the size of clusters. In contrast, traditional hierarchical clustering algorithms generate many partitions of networks and the users need to choose the best partition, which results in subjectivity in the partitions.
- these algorithms are implemented in a number of commercial and open-source network analysis applications.
3.3.2 Comparison of clusters

The overlap of clusters between two partitions of a network can be measured in different ways. In this paper we use the Rand index [44] for comparing clusters, which is a measure of the similarity between two data partitions. The Rand index can be expressed as:

\[ R = \frac{a + b}{\binom{n}{2}} \]  

(2)

where \( a \) is the number of pairs of participants who are in the same cluster in both partitions being compared, \( b \) is the number of pairs of participants who are in different clusters in both partitions; \( n \) is the total number of participants. The results from the overlap in the example open source community are presented in Section 4.2.3.

4 AN EXAMPLE FROM OPEN SOURCE SOFTWARE

In this section, we present the approach using an open source software project, Drupal [49]. Drupal is a content-management system used for the creation of community-based websites. Drupal has been under development since 2001. We analyze four major versions of Drupal core (version 4 through 7). Drupal is well developed with over 7000 community-contributed add-ons, known as contrib modules. Besides, the project also attracts more than 5000 developers. Drupal is selected because of its maturity, availability of code and the availability of information about communication between participants.

4.1 Overview of Drupal data and basic network statistics

The software structure of Drupal is modeled as a weighted network where nodes represent files and the links represent function-calls between files. A documentation generator tool, Doxygen [50], is employed to extract functions and corresponding function calls from the source code. The strength of the interface between files is defined by the number of function calls between two files. The community structure of Drupal is also modeled as a weighted network. The communications between participants are derived from on-line forums. The communication links between participants are determined by analyzing each post on the forum. A post on the forum contains information about the names of participants and the software version they are discussing about. Relationships are built among participants discussing on the same post.
The participation links are created by analyzing file modifications and issue/patch records over time, recorded on the Drupal website. A file modification indicates which files are modified to resolve a specific issue. An issue/patch record indicates who worked on the specific issue/patch. Combining the information derived from file modifications and issue/patch records, the participation links representing participants working on files are generated. The generation of the networks with the three types of links is illustrated in Figure 5. The basic network statistics for versions 4-7 is shown in Table 3.

![Figure 5 – Generation of the Drupal networks with three types of links](image)

### Table 3 – Basic information about the networks generated

<table>
<thead>
<tr>
<th>Versions</th>
<th>Number of files</th>
<th>Number of interfaces between files</th>
<th>Number of participants</th>
<th>Number of participation links</th>
<th>Number of communication links</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>45</td>
<td>359</td>
<td>1182</td>
<td>12608</td>
<td>15511</td>
</tr>
<tr>
<td>5</td>
<td>47</td>
<td>361</td>
<td>1855</td>
<td>14059</td>
<td>28656</td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>730</td>
<td>3491</td>
<td>30098</td>
<td>89741</td>
</tr>
<tr>
<td>7</td>
<td>180</td>
<td>997</td>
<td>5185</td>
<td>103964</td>
<td>148102</td>
</tr>
</tbody>
</table>

### 4.2 Testing the hypotheses for Drupal product and community

#### 4.2.1 Hypotheses 1.1 and 1.2

Hypothesis 1.1 is that as the number of interfaces of product modules increases, the number of new communication links between people working on those modules also increases. To validate this hypothesis, the number of new interfaces of each file is determined for consecutive versions of the software. Then for each file, the participant set is determined and the number of new communication links of the participant set is evaluated. The number of new interfaces of files is plotted against the average...
number of new communication links of the corresponding participant sets (see Figure 6). Note that the number of communication links (y-axis) is plotted on a logarithmic scale. It is observed from the figure that the relationship between the new interfaces and the new communication links is exponential for each of the consecutive versions of the software. The details of the regression models are presented in Table 4. The p-values associated with the T-tests are < 0.001, indicating that the number of interfaces have a statistically significant relationship with the new communication links. The $R^2$ values for the regression indicate that more than 60% of the variation in the dependent variable is attributed to the independent considered in the statistical model. These results indicate that the increase in the number of links in the product network is associated with an increase in communication between individuals, supporting the validity of Hypothesis 1.1 for Drupal.

![Figure 6 – New interfaces of files vs. average number of new communication links of the corresponding participant sets](image)

Table 4 - Regression model corresponding to Hypothesis 1.1

<table>
<thead>
<tr>
<th></th>
<th>Version 4-5</th>
<th>Version 5-6</th>
<th>Version 6-7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data points</strong></td>
<td>187</td>
<td>448</td>
<td>529</td>
</tr>
<tr>
<td><strong>Regression</strong></td>
<td>$\log_{10}(y) = 1.04 + 0.00303x$</td>
<td>$\log_{10}(y) = 1.46 + 0.000656x$</td>
<td>$\log_{10}(y) = 1.64 + 0.000538x$</td>
</tr>
<tr>
<td><strong>$R^2$</strong></td>
<td>0.609</td>
<td>0.647</td>
<td>0.604</td>
</tr>
<tr>
<td><strong>T-test (slope)</strong></td>
<td>$T = 16.98$</td>
<td>$T = 28.62$</td>
<td>$T = 28.32$</td>
</tr>
<tr>
<td><strong>P-value</strong></td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
</tr>
</tbody>
</table>

Hypothesis 1.2 is that if the number of a participant’s communication links increases, the number of new interfaces of corresponding modules also increases. To test this hypothesis, the number of new communication links of all participants is determined and plotted against the average number of new interfaces of the corresponding module sets (see Figure 7). The corresponding linear regression model is presented in Table 5. While the T-test shows that there is statistically significant relationship between the two variables (with P-values less than 0.05), the $R^2$ value is significantly lower (0.309, 0.399, and 0.127). This shows that the regression model explains only about 30% of the variation in the output. The rest of the variation is due to unknown, lurking variables or inherent uncertainty. Hence, we conclude that the
while Drupal data provides some support for Hypothesis 1.2, there are other important variables not accounted for in this model that contribute to new interfaces between files. For example, new functionality implemented in new versions of the code has a significant effect on the number of new file interfaces. Further investigation on these other variables is necessary to validate Hypothesis 1.2.

![Figure 7](image)

**Figure 7 – New communication links vs. the average number of new interfaces of files in the corresponding module sets**

**Table 5 – Linear regression model corresponding to Hypothesis 1.2**

<table>
<thead>
<tr>
<th></th>
<th>Version 4-5</th>
<th>Version 5-6</th>
<th>Version 6-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Points</td>
<td>13</td>
<td>60</td>
<td>84</td>
</tr>
<tr>
<td>Regression</td>
<td>y = - 65.5 + 0.00355 x</td>
<td>y = - 26.8 + 0.00192 x</td>
<td>y = - 53.9 + 0.000416 x</td>
</tr>
<tr>
<td>Regression (Normalized data)</td>
<td>y_n = - 0.037 + 0.527 x_n</td>
<td>y_n = - 0.0576 + 0.499 x_n</td>
<td>y_n = - 0.0614 + 0.203 x_n</td>
</tr>
<tr>
<td>R²</td>
<td>0.309</td>
<td>0.399</td>
<td>0.127</td>
</tr>
<tr>
<td>T-test (slope)</td>
<td>T = 2.22</td>
<td>T = 6.21</td>
<td>T = 3.45</td>
</tr>
<tr>
<td>P-value</td>
<td>0.049</td>
<td>&lt; 0.001</td>
<td>0.001</td>
</tr>
</tbody>
</table>

### 4.2.2 Hypotheses 2.1 and 2.2

Hypotheses 2.1 and 2.2 are focused on the effects of new relationships between pairs of modules and individuals respectively. Hypothesis 2.1 is that if new interfaces are created between two modules, then the corresponding communication links are also created between individuals working on those modules. To validate this hypothesis, the number of new interfaces is evaluated for each pair of new files. The number of new interfaces is plotted against the average number of new communication links between the corresponding pair of participant sets (see Figure 8). Similar to Hypothesis 1.1, it is observed that there is an exponential relationship between the number of new interfaces created between a pair of files and the number of new communication links. The corresponding regression models are presented in Table 6. The P-value in each of the models is less than 0.001, indicating the statistical significance of the regression models. The R² values are also greater in this case. These values indicate that 68.6% of the
variation in the dependent variable is explained by the independent variable in Version 4-5, and over 80% in Version 5-6 and Version 6-7. This provides strong support towards the validity of Hypothesis 2.1.

![Graphs showing new interfaces between two files vs. average number of new communication links between the corresponding pair of participant sets](image)

**Figure 8** – New interfaces between two files vs. average number of new communication links between the corresponding pair of participant sets

<table>
<thead>
<tr>
<th>Table 6 - Regression model corresponding to Hypothesis 2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 4-5</strong></td>
</tr>
<tr>
<td><strong>Data Points</strong> 227</td>
</tr>
<tr>
<td><strong>Regression</strong> Log(_{10}(y) = -0.134 + 0.00454 x)</td>
</tr>
<tr>
<td><strong>R(^2)</strong> 0.686</td>
</tr>
<tr>
<td><strong>T-test (slope)</strong> T = 22.17</td>
</tr>
<tr>
<td><strong>P-value</strong> &lt; 0.001</td>
</tr>
</tbody>
</table>

Hypothesis 2.2 is that if two people communicate with each other, then the corresponding modules also develop interfaces. To validate this hypothesis, new communication links are evaluated between each pair of individuals. The number of new interfaces between the corresponding module sets is determined and plotted against new communication links in Figure 9. No clear relationship between the communication links and the file interfaces is evident from the figure. We build linear regression models, between the x and y axes. The details of the models are presented in Table 7. As in the case of Hypothesis 1.2, the R\(^2\) values are very low (0.079, 0.158, and 0.061), indicating that only about 6-16% of the variation in the dependent variable can be explained in terms of the independent variables. The rest of the variation is due to unknown variables or inherent variability. Hence, we conclude that the data does not provide strong support for Hypothesis 2.2.
Based on the strong support for the validity of Hypotheses 1.1 and 2.1 but weak support for Hypothesis 1.2 and 2.2 for Drupal, we conclude that there is a clear effect of the product structure on the community structure. However, the effect of the community structure on the product structure is unclear. This is in contrast to the conclusion of Merlo et al. [19] that in open source projects, the organizational structure affects the product structure but not vice versa.

4.2.3 Hypothesis 3

Based on the mirroring hypothesis and Conway’s law, Hypothesis 3 in this paper is that the communities get increasingly aligned with the anticipated communication patterns. This can also be stated as the increase in socio-technical congruence. The existing approach to measure socio-technical congruence [7, 8] is to measure coordination requirements based on the product dependencies and to compare the coordination requirements with the actual communication patterns between individuals. Coordination requirements network consists of individuals and communication links that should ideally be present in order to resolve product dependencies. The links that occur in both coordination requirements network and actual communication network are referred to as matched interactions [20]. The links that are present in the coordination requirements network but absent in the actual communication network are called potential unattended interactions. Finally, the links present in the actual communication network but
absent in the coordination requirements network are called *unpredicted interactions*. The ratio of the matched interactions to the total number of links in the coordination requirements network is used as measure of socio-technical congruence [7, 8].

The values of congruence measure for different versions of Drupal are shown in Table 8. It is observed that congruence values are in the range of [0.0178 0.0099], which indicates that only about 1%-2% of the expected interactions between individuals are actually present. Additionally, the socio-technical congruence monotonically decreases from versions 4 though 7. One potential explanation for the low congruence values is the fact that in open source communities, not all participants contribute or communicate uniformly to the code development activities. Open source communities have been shown to represent core-periphery structures where there is a core set of participants who are highly connected among themselves and a peripheral set of participants who are weakly connected with each other [51]. It is likely that the core participants are responsible for addressing most of the coordination needs. To check whether the core participants have a higher coordination, we determine the congruence for core participants and peripheral participants separately. We utilize the implementation in UCINET [52] to determine the core and peripheral participants. The core-periphery structure of the Drupal community for version 5 is illustrated in Figure 10. The number of participants and the congruence values in the core and the periphery are shown in Table 9. It is observed that the congruence of the core participants is between 0.2208 and 0.0963, which is significantly greater than the congruence of peripheral participants, validating the assumption that core participants play a major role in coordination.

**Table 8 – Congruence measured using approach by Cataldo et al. [7, 8] and matching of interactions using approach by Sosa [20]**

<table>
<thead>
<tr>
<th>Versions</th>
<th>Congruence</th>
<th>Matched Interactions</th>
<th>Unpredicted Interactions</th>
<th>Potential Unattended Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V4</td>
<td>0.0178</td>
<td>12555</td>
<td>2956</td>
<td>693699</td>
</tr>
<tr>
<td>V5</td>
<td>0.0140</td>
<td>21008</td>
<td>7648</td>
<td>1478563</td>
</tr>
<tr>
<td>V6</td>
<td>0.0117</td>
<td>72160</td>
<td>17581</td>
<td>6119698</td>
</tr>
<tr>
<td>V7</td>
<td>0.0099</td>
<td>127983</td>
<td>20119</td>
<td>12782290</td>
</tr>
</tbody>
</table>
Figure 10 – Core and peripheral nodes in the community (Version 5)

Table 9 – Socio-technical congruence and number of participants in the core and periphery

<table>
<thead>
<tr>
<th>Versions</th>
<th>Socio-technical congruence</th>
<th>Number of participants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Core</td>
<td>Periphery</td>
</tr>
<tr>
<td>V4</td>
<td>0.2208</td>
<td>0.0141</td>
</tr>
<tr>
<td>V5</td>
<td>0.1304</td>
<td>0.0092</td>
</tr>
<tr>
<td>V6</td>
<td>0.1213</td>
<td>0.0091</td>
</tr>
<tr>
<td>V7</td>
<td>0.0963</td>
<td>0.0059</td>
</tr>
</tbody>
</table>

Low values of socio-technical congruence have been reported in other studies on open-source software. For example, Kwan et al. [22] observed that over 75% of software builds in an IBM project involving distributed developers had congruence values of less than 25%. Even for software builds with a congruence of 0%, the authors [22] observed a build success of 93%. Despite the low values of congruence, the projects can be successful due to a number of reasons. First, if the code is highly modular, then even though there are function calls across files, it does not mean that all the other files linked by function calls need to be modified. In fact, the number of functions varies significantly across files. Most files have only a few functions whereas some files have a number of functions. In other words, the coordination requirements network has significantly greater number of links than actually needed for coordinating the activities. Second, not all coordination is through explicit communication. Coordination also happens through the shared workspace where individuals can get information about addressing dependencies between technical aspects. Bolici and co-authors [23] argue that within open-source communities, the coordination primarily happens via indirect communication, i.e., the information needed to coordinate activities is embedded within the software itself. The coordination achieved by indirect communication is referred to as *stigmergic coordination*. Hence, the current approach of directly
comparing coordination requirements network with actual communication patterns to measure socio-technical congruence is limiting for open-source communities.

The approach presented in this paper is an alternate view of socio-technical congruence based on the comparison of clusters. Here, the focus is on comparing the anticipated communities (instead of anticipated links) with actual clusters of people working together (instead of actual communication links). If the anticipated communities are similar to the actual communities, we believe that the socio-coordination is high. The proposed approach is different from the socio-technical congruence measure commonly used [7, 8] because the comparison is at the community level, as opposed to the individual level. The anticipated communities are obtained using two different approaches – a) clustering the product network, and b) clustering the bipartite (affiliation) network with participation links. As shown in Figure 11, the overlap between the anticipated communities and the actual communities is significant (with a Rand Index of over 0.61). Moreover, the overlap increases from Version 4 through 7, which validates Hypothesis 3 for the software under consideration. The sizes of the clusters generated by different approaches are shown in Figure 12.

![Figure 11 - Rand-index comparing community clusters generated by different approaches: (a) clustering of product network and community network, (b) clustering of bipartite network and community network](image)

![Figure 12 - Sizes of clusters based on the clustering of different networks](image)
5 CLOSING COMMENTS

Understanding the interdependence between products and organizational structures is essential for managing dependencies and achieving coordination between product development activities. The interdependence has been studied in the context of socio-technical coordination theory and the mirroring hypothesis. Socio-technical coordination is an empirical theory, which can be tested by drawing out implications for observable phenomena, i.e., generating testable hypotheses. In this paper, we lay out a number of empirically testable hypotheses and present different techniques for testing these hypotheses for loosely coupled open source communities.

The proposed approach is based on dynamic network-based modeling of products and communities. The proposed approach is novel in two dimensions. First, instead of relying on the comparison of static views of the product and organization networks, we analyze the underlying mechanisms that result in the evolution of these networks including the addition of nodes and links. Since the focus is on analyzing the socio-technical coordination, the focus is primarily on the number of links added during the consecutive versions of the product. While the static view of networks can be used to determine whether there is mirroring between product and community networks, the dynamics-based view helps in identifying the underlying mechanisms that may result in the mirroring of products and communities. The second novelty of the proposed approach is that instead of measuring socio-technical congruence using by comparing the existing communication links with ideal communication links, we compare the overlap between existing communities and anticipated communities. This helps in addressing the limitations of existing approaches for measuring socio-technical coordination for open source development processes.

The proposed approach is applied to an example from an open source software community, Drupal. In the example, the product networks are modeled as networks of files with links representing function calls between files. The community networks are modeled as individuals with communication links between them. Finally, the links between individuals and files are established by determining who works on which files. By comparing the consecutive versions of the networks, we conclude that for Drupal, product structures significantly influence the communication patterns between individuals but the impact of communication patterns on the product structures is not evident. As discussed in Section 1, the
alignment between products and organizations is beneficial in terms of product quality and productivity. Our results from the cluster comparison approach suggest that in self-organized product development (e.g., in open source), even though the alignment is not enforced by a higher level authority, the community structure naturally gets aligned to the product evolution. This is encouraging. The proposed techniques can help communities and product development organizations in determining a) the extent of this alignment, b) the aspects of the product where alignment is high and c) the aspects where alignment is low. This information can be used to facilitate communication between individuals within organization and community where alignment is low.

While the approach has been used in this paper for a software product, the same approach can be directly utilized for physical products also. For physical products, different types of module dependencies (such as spatial, energy, material, and information) can be used to represent the links in the product network. The software product was chosen because of the availability of data related to the dependencies between files and communications between individuals.

One of the limitations of the data analyzed in this study is that the product is modeled at the file level. File-level granularity was chosen because the information about individuals working on specific functions was not available. Ideally, a more detailed function-level model is preferred. Another limitation of comparing the consecutive versions of networks is that a change in one network (e.g., product network) may be reflected in the other network (e.g., community network) after more than one version. These longer-term influences cannot be captured by simply comparing consecutive versions. Further research is needed to determine these longer term influences between product and community networks.

6 REFERENCES


List of Figures

Figure 1 – A hybrid network model to represent product structure, community structure and participation
Figure 2 – Extracting the relationship between the number of newly created links of participants and newly created links of modules associated with that participant (for Hypothesis 1.2)
Figure 3 – Extracting the relationship between the number of newly created links between a pair of participants and the newly created links between corresponding module sets (for Hypothesis 2.2)
Figure 4 – Cluster comparison approach
Figure 5 – Generation of the Drupal networks with three types of links
Figure 6 – New interfaces of files vs. average number of new communication links of the corresponding participant sets
Figure 7 – New communication links vs. the average number of new interfaces of files in the corresponding module sets
Figure 8 – New interfaces between two files vs. average number of new communication links between the corresponding pair of participant sets
Figure 9 – New communication links between two individuals vs. average number of new interfaces between corresponding pair of module sets
Figure 10 – Core and peripheral nodes in the community (Version 5)
Figure 11 – Rand-index comparing community clusters generated by different approaches: (a) clustering of product network and community network, (b) clustering of bipartite network and community network
Figure 12 – Sizes of clusters based on the clustering of different networks

List of Tables

Table 1 – A summary of the key studies on the interrelationships between products and organizational structures in software products
Table 2 – Different aspects corresponding to the hypotheses
Table 3 – Basic information about the networks generated
Table 4 - Regression model corresponding to Hypothesis 1.1
Table 5 – Linear regression model corresponding to Hypothesis 1.2
Table 6 - Regression model corresponding to Hypothesis 2.1
Table 7 - Regression model corresponding to Hypothesis 2.2
Table 8 – Congruence measured using approach by Cataldo et al. [7, 8] and matching of interactions using approach by Sosa [20]
Table 9 – Socio-technical congruence and number of participants in the core and periphery