A Generative Network Model for Product Evolution

Qize Le
School of Mechanical and Materials Engineering, Washington State University, Pullman, WA 99164
Zhenghui Sha
School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907
Jitesh H. Panchal1
Assistant Professor
School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907
e-mail: panchal@purdue.edu

1 Corresponding author.
Contributed by the Design Engineering Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received October 9, 2012; final manuscript received October 23, 2013; published online January 10, 2014. Editor: Bahram Ravani.

Modeling the structure and evolution of products is important from the standpoint of improving quality and maintainability. With the increasing popularity of open-source processes for developing both software and physical systems, there is a need to develop computational models of product evolution in such dynamic product developments scenarios. Existing studies on the evolution of products involve modeling products as networks, taking snapshots of the structure at different time steps, and comparing the structural characteristics. Such approaches are limited because they do not capture the underlying dynamics through which products evolve. In this paper, we take a step toward addressing this gap by presenting a generative network model for product evolution. The generative model is based on different mechanisms through which networks evolve—addition and removal of nodes, addition and removal of links. The model links local network observations to global network structures. It is utilized for modeling and analyzing the evolution of a software product (Drupal) and a physical product (RepRap) developed by open source processes. For the software product, the generated networks are compared with the actual product structures using various network measures including average degree, density, clustering coefficients, average shortest path, partition cost, clustered cost, and degree distributions. For the physical product, the product evolution is analyzed in terms of the proposed mechanisms. The proposed model has three general applications: longitudinal studies of a product’s evolution, cross-sectional studies of evolution of different products, and predictive analyzes. [DOI: 10.1115/1.4025856]

Keywords: product structure, evolution, complex networks, degree-based models, modularity

1 Introduction

The structure of a product is an indicator of its complexity, and hence, impacts its quality and maintainability. Studies have shown that the product structure has an impact on the organizational structure also [1,2]. Due to its importance, the analysis of product structure has received significant attention within the engineering design literature. One of the prominent ways of modeling product structure is through dependency modeling techniques where a product is modeled as a network of components linked through dependency relationships. The network can be represented in a matrix form using the design structure matrix [3,4]. The matrix can be analyzed to answer a number of questions such as:

(a) How modular or complex is the product?
(b) How the modularity affects the system performance?
(c) Is this product more modular than other products?
(d) How does the product structure affect complexity?
(e) How does a change in a particular module propagate to other modules within the product?

Answering these questions can lead to knowledge for better designs and more efficient product development processes. The recent special issue on dependency modeling techniques in the Journal of Engineering Design [5] highlights some of the latest developments in this area. Existing efforts on analyzing product evolution are primarily focused on hierarchical product development where product architecture is determined early in the design process and the product structures do not change significantly during the process. In contrast, emerging mass-collaborative product development processes [6,7] such as open-source product development involve significant changes in the product structure as new requirements are continuously proposed, new modules are created, and new interfaces are designed. Exponential growth in the size of products has been observed in some cases [8]. In such dynamic scenarios, the analysis of the structural changes in the products with time provides important information about the rate of evolution and effectiveness of the product development process.

Our goal is to develop dynamic network-based models for products that undergo such drastic evolution during their development. The work is motivated by (a) the uniqueness of the emerging product development processes, and (b) the unprecedented access to product evolution data. Open-source processes represent a fundamentally different way in which participants organize and coordinate activities to develop products, as compared to traditional top–down processes [9]. While open-source software development has been well known for the past two decades, the approach is increasingly being adopted for hardware development also [10]. Examples of open hardware products include RepRap [11] and Arduino [12]. Due to the increasing interest in using open-source principles for physical products, there is a need to understand how open-source products evolve. Additionally, open-source processes are carried out on the Internet. The product structure and its evolution are well documented and openly available, making them ideal for analysis. There exist well-developed platforms such as SourceForge [13] and GitHub for open-source software development, which enable detailed modeling of product evolution. As open-source hardware matures, it is expected that similar platforms will capture information for physical products also.

Existing approaches for studying product evolution involve taking snapshots of the structure at different time steps, and comparing the structural characteristics. These structural characteristics range from the extent of coupling and cohesion to various metrics for modularity and complexity [2]. Such approaches are suitable for comparing the global characteristics of the product networks but are not effective for capturing the underlying dynamics through which products evolve. For example, comparisons of the
snapshots do not provide information about how the local (module-level) observations result in the evolution of the product networks. Further, comparison of structural measures of snapshots only provides information about the specific versions of the products being analyzed. It does not provide the capabilities to perform predictive or “what-if” analyses to understand the impacts of future design modifications. To facilitate such analyses, models that capture the evolutionary dynamics of networks based on local observations on addition and deletion of nodes and links are required. These models are bottom-up in nature, and are referred to as generative models. Our review of literature reveals that there is a lack of generative models of open-source product development (see Sec. 2.1). Generative models that embody the underlying mechanisms of network growth are important because they can help in understanding the reasons for increasing complexity of products, and identifying specific ways to maintain it, to increase modularity, and to reduce product complexity.

The goal in this paper is to present a generative model for the evolution of products. The model is inspired by existing models in the network science literature, reviewed in Sec. 2.2. The model embodies two categories of mechanisms through which networks evolve: (1) addition or deletion of nodes and (2) addition or deletion of links. These categories are divided into six mechanisms that describe how nodes are added (or removed) and linked with each other. These mechanisms provide information about the following questions: How many new nodes are added at a certain time-step? How many existing nodes are removed? For given existing nodes, what are the probabilities of creation of links with new nodes, and with other existing nodes? What are the probabilities of removal of existing nodes? For new nodes added, what are the probabilities of linking with existing nodes and other new nodes? The evolution of the product networks is modeled using these mechanisms. Depending on the level at which the product is analyzed, the nodes can refer to different aspects of a product (e.g., modules, files, functions, or classes in a software product). The links refer to dependencies between nodes (e.g., class dependencies and function calls).

We apply these mechanisms to model the evolution of an open-source software product, Drupal [14]. The results indicate that the model generates dynamic networks whose evolutionary characteristics are close to that of the original product structure. To illustrate the generality of the approach, we also apply the model to RepRap, an open source hardware product. The results indicate that such a bottom-up approach can be utilized for modeling evolutionary product structures in open-source processes. The paper is organized as follows. In Sec. 2, we discuss the existing literature on product evolution in the open-source domain, and generative models of networks. A discussion of the mechanisms of network evolution and application to Drupal is presented in Sec. 3. The network-level properties of the Drupal product structure are evaluated in Sec. 4. The proposed generative model and its application to Drupal are presented in Sec. 5. Application to RepRap is presented in Sec. 6. Finally, closing comments are present in Sec. 7.

2 Review of Relevant Literature

The relevant literature includes two aspects: analysis of product evolution in open-source domain (discussed in Sec. 2.1), and models of network generation (discussed in Sec. 2.2).

2.1 Existing Studies on the Product Structure in Open-Source Domain. Since open-source hardware is still in its infancy, existing literature in the open-source domain is mainly focused on software development. Crowston and co-authors [15] recently published a survey article highlighting the diverse research efforts on the open-source domain. In the survey, they categorize the literature into inputs (member characteristics, project characteristics, and technology use), processes (software development practices, social practices, and firm involvement practices), emergent states (social states and task related states), and outputs (software implementation, team performance, and evolution). This paper fits into their outputs (evolution) category. Some of the earliest models of software evolution were in the form of differential equations [16] built using general principles such as the Lehman’s laws [17].

The software structure is modeled by the technical dependencies which can be identified using two approaches [18]. The first approach involves extracting relational information between entities such as statements, functions, files, or modules. The relationships between the entities can be data-related dependencies [18], functional dependencies [19,20], or syntactic dependencies [21]. The second approach involves identifying dependencies by examining how modification requests affect the source code [22]. Examples include analysis of code decays based on modification requests [23], and analysis of modifications involving files that tend to change together [24].

Based on the goals, the literature can be categorized into three groups. The first group of studies focuses on the evolution of software architecture in a single product. The studies in this group are primarily focused on the modularity and complexity of software architecture, such as determining how the size increases [25], modularity changes [26], and complexity increases [27] with software evolution. MacCormack et al. [28] and LaMantia et al. [29] analyze the impact of software modularity on evolutionary characteristics. Different metrics are proposed to quantify modularity and structural complexity of software. These metrics include coupling and cohesion [30], propagation cost and clustered cost [2].

Modularity is an important characteristic because it affects evolvability, changeability, maintainability [31], and customizability. Modularity is measured in terms of complex network metrics such as path length and clustering coefficient [32].

The second group of studies is focused on comparing the structures of different open-source products to identify differences and common patterns across different projects. Mockus et al. [33] compare the development process of Apache and Mozilla. MacCormack et al. [2] analyze the structures of Linux and Mozilla and compare them using propagation cost and clustered cost metrics. The authors show that the modularity of Mozilla was initially less than that of Linux, but increased after the redesign efforts. Valverde and Sole [34] analyze 80 OSS projects to determine commonalities. The authors discover that the product structures had hierarchical small-world and scale-free characteristics. Further, the clustering coefficients (C) of the projects are significantly larger than their random counterparts. Valverde and Sole [34] mainly focus on discovering topologies and characteristics of product structures based on complex network analysis.

The third group of studies focuses on comparing the structures of open-source and proprietary software where the main goal is to identify the commonalities and differences between software products developed using fundamentally different techniques and organizational (community) settings. Raymond [35] and O’Reilly [36] claim that opens-source software is more “modular” than proprietary software. On the other hand, Torvalds [37] suggests that modularity is a required property for the success of open source software (OSS) development. MacCormack et al. [2] compare open-source software products with proprietary software.

As a summary, various case studies have been performed to analyze the evolution of software products. Different versions of software are compared using metrics from complex network analysis and metrics for modularity and complexity. Differential equations have been developed to model the growth of commercial software products. However, such general equations only account for evolution in terms of the size. Since dependencies between modules are not taken into account, the models do not capture the evolution of structural complexity (which depends on how the entities are linked). Currently, there is a lack of network-based generative models that capture the evolution of open-source products and the corresponding changes in modularity and complexity. Such network generation models should not only capture the
growth in terms of size but also in terms of structural complexity. To address this gap, we present a network generation model based on local network observations. Such models have been used in the complex network literature to model networks with different topologies. To develop an appropriate model, literature review on existing network generation models is performed in Sec. 2.2.

2.2 Existing Studies on the Network Generation Models.

Network-generation models can be classified into two types: structure-based models and evolution-based models. Structure-based models generate networks based on the underlying structural characteristics while the evolution-based models generate networks based on assumed evolutionary dynamics.

2.2.1 Structure-Based Models. The exponential random graph model, also referred to as $P^*$ model, is a widely used structure-based model [38]. In $P^*$ models, it is assumed that the network is generated by some statistical process and the observed network is one realization from a set of possible networks with similar characteristics (e.g., number of actors). The probability of realizing a specific network is given by [38]

$$
Pr(Y = y) = \frac{1}{Z} \exp\left\{\sum_k \eta_k g_k(y)\right\}
$$

(1)

where $Pr(Y = y)$ represents the probability that a network $y$ emerges, $k$ is a normalizing parameter which ensures that the probability falls in a proper distribution, $A$ is the set of substructure configurations, $g_k(y)$ is the network statistic corresponding to the configurations $A$. Based on the observed network, the parameters $\eta_k$ are calculated using methods such as pseudo-likelihood estimation [39] and Markov chain Monte Carlo (MCMC) maximum likelihood estimation [40]. The $P^*$ model is built by formulating assumptions about substructure configurations and then validating these assumptions using the resulting parameters $\eta_k$. Simple substructure configurations include reciprocity, two-star, three-star, and triangle.

2.2.2 Evolution-Based Models. In evolution-based models, an initial network is chosen to represent the early stage of a real network. New nodes and links are gradually added (and removed) to simulate the growth of the network [41]. The process of linking of new nodes can be driven by different local properties of connecting nodes. The most popular local property used for linking is a node’s degree. The degree of a node is the number of other nodes connected to it. An example of evolution-based models using degree-based linking mechanism is the Barabasi-Albert model of scale-free graphs [42]. In this model, the assumption is that new nodes entering the network attach to existing nodes with a probability proportional to their degree. Hence, the existing nodes with greater connections have greater probability of linking to new nodes. This is also referred to as preferential attachment. A number of researchers have proposed variations to the Barabasi-Albert model to account for different characteristics of real world networks. For example, linear preferential attachment with initial attractiveness [43] is proposed to simulate real networks with power law distribution with arbitrary exponent. Nonlinear preferential attachment models have also been proposed [44] where the probability of attachment depends on $k^z$, where $k$ is the degree of the node and $z$ is an arbitrary parameter.

In this paper, we choose to model the evolution of product structures using degree-based evolution models over structural models due to two reasons. First, the evolution-based models relate the local dynamic behaviors to the global network structures whereas structural models capture the relationships between local structures with global structures. Hence, evolution-based models are a natural choice. Second, some studies suggest that degree-based evolutionary models can be more effective in modeling the global network structures than the structure-based models [45]. Hence, we focus on the degree-based models in the remainder of this paper. The node-level mechanisms based on the degrees of nodes are discussed in Sec. 3.

3 Node-Level Mechanisms for Product Evolution

The premise in the proposed model is that six network evolution mechanisms at the node level determine the evolution of product structures. Additionally, it is assumed that the mechanisms are only dependent on the degrees of nodes. The nodes of a software product network can refer to functions, classes, files, or features. The nodes in a physical product network can refer to parts, parameters within parts, or sub-assemblies. Function-call level of abstraction is considered in this paper for software products and part level abstraction is considered for physical products. We discuss the mechanisms in Sec. 3.1 and illustrate them through application to the Drupal product network in Sec. 3.2.

3.1 Mechanisms for Modeling the Evolution of Open-Source Products.

The six mechanisms through which networks evolve are illustrated in Fig. 1, and discussed next.

(a) Addition of new nodes: The primary growth mechanism of a network is the addition of new nodes. This mechanism corresponds to the addition of new modules, functions, classes, parts, etc., to address new requirements, specifications, and features. The trends in the number of additional nodes can be determined by comparing consecutive versions of the product structure.

(b) Removal of existing nodes: Existing nodes may be removed from a product network because existing features may no longer be needed or are replaced by new features. The number of existing nodes removed from a product can also be determined by comparing consecutive versions.

(c) Linking of new nodes with existing nodes: After new nodes are added, these nodes can be linked to existing nodes by new interfaces (function calls in the case of call graphs). In our degree-based model, we assume that the probability that a new node links to existing node is a function of the
Table 1 Data associated with node-level mechanisms for Drupal network (V2 → V5)

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>V2 → V3</th>
<th>V3 → V4</th>
<th>V4 → V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Nodes added</td>
<td>294</td>
<td>356</td>
<td>780</td>
</tr>
<tr>
<td>(b) Nodes removed</td>
<td>154</td>
<td>139</td>
<td>439</td>
</tr>
<tr>
<td>(c) Links between new and existing nodes</td>
<td>558</td>
<td>987</td>
<td>1451</td>
</tr>
<tr>
<td>(d) Links among new nodes</td>
<td>530</td>
<td>350</td>
<td>1286</td>
</tr>
<tr>
<td>(e) Links among existing nodes</td>
<td>51</td>
<td>140</td>
<td>151</td>
</tr>
<tr>
<td>(f) Links removed</td>
<td>492</td>
<td>744</td>
<td>1790</td>
</tr>
</tbody>
</table>

degree of existing node: \( P(A_{1e}) = F_1(K_e) \), where \( A_{1e} \) represents the attachment between existing nodes and a new node, \( K_e \) represents the degree of the existing node. To determine the relationship, we compare two consecutive versions of the product structure network and calculate the average numbers of interfaces created between existing nodes with a given degree and new nodes.

(d) Linking of new nodes with each other: Since new nodes do not have any initial links, we assume that the new nodes first link with existing nodes and then link with new nodes. After the new nodes link with the existing ones, the degree of a new node is referred to as the “initial degree”. This initial degree is used to determine the probability of creation of links between two new nodes. The probability of a new node being linked with other new nodes is modeled as: \( P(A_{1e,2}) = F_2(K_{n2}) \), where \( A_{1e,2} \) represents the attachment between two new nodes \((n1\) and \(n2)\), \( K_{n2} \) represents the “initial degree” of the new nodes.

(e) Linking of existing nodes with each other: New links can also be added between existing nodes. This corresponds to the addition of a new function call between two existing functions. The probability that an existing node is attached to other existing nodes is a function of its degree: \( P(A_{1e,2}) = F_3(K_e) \), where \( A_{1e,2} \) represents the attachment between existing nodes, \( K_e \) represents the degree of the existing nodes.

(f) Removal of existing links: Existing links can be removed in new product versions because of two reasons. First, existing nodes may be removed. In this case, the existing links associated with these nodes are also removed. Second, the links between two existing nodes are no longer used. Hence, the existing links are removed. The probability of removal of existing links between existing nodes is calculated by comparing consecutive versions of the code. The probability function can be represented as: \( P(R_{1e,2}) = F_4(K_e) \), where \( R_{1e,2} \) represents the removal of links between existing nodes, \( K_e \) represents the degrees of existing nodes.

The existing degree-based models discussed in Sec. 2.2 present specific functional forms for the probability functions. For example, the Barabasi-Albert model [46] assigns a linear probability function for linking between new nodes and existing nodes. However, we do not pre-assign any functional form for the probability functions \( F_1-F_4 \). These functions are determined and estimated based on the observed data.

3.2 Utilizing the Node-Level Mechanisms for Drupal Product Network

Drupal [14] is an open-source content-management system, which is used for the creation of community-based websites. In this paper, we analyze five major versions of Drupal core (2.0 through 5.0). Drupal is well developed with over 7000 community-contributed add-ons, known as contrib modules. Besides, the project also attracts more than 1000 developers. Drupal is selected because of its maturity and the availability of code for different versions. As mentioned earlier, function-level dependencies in the call graph are used to model the product structure. Function calls represent one of the ways of modeling the structure of software. Call graphs have been widely used to model software architecture [18,47,48]. If function B is called within function A, then an interface is created between functions A and B.

In the first step, raw data about the product structure are extracted from major versions of the source code. The raw data consist of all the functions in the source code and the corresponding function calls. The data are used to derive the relationships among the functions. The second step is to model the product structure as a complex network in which functions are nodes and function calls are links. A documentation generator tool, Doxygen [49], is used to create the call graph. Having generated the networks for different versions of the code, consecutive versions of the network are compared to extract quantitative information about the node-level mechanisms.

Mechanisms (a) and (b): The data corresponding to the mechanisms (a) and (b) for different evolutionary steps in Drupal network are displayed in Table 1. On comparing the number of new nodes added, existing nodes removed and the total number of nodes, it is observed that about half of the existing nodes are removed. The number of newly added nodes is close to the total number of nodes in the previous version. This demonstrates significant evolution of the product. For example, new features are added, the outdated features are removed, the features that are useful but not efficient are replaced, and bugs are found and corrected. LaMantia et al. [29] also detect a high value of change ratio for Tomcat-main product.

Mechanism (c): The mechanism (c) is the probability that existing nodes are linked to new nodes. The probabilities with which new nodes link with existing nodes are plotted against the degrees of existing nodes on a log–log scale in Fig. 2. The probability functions are determined by fitting linear functions in the log–log plots. As shown in the figure, the exponents in the probability functions are 1.0705, 0.9270, and 0.9828 for versions V2 → V3, V3 → V4, and V4 → V5, respectively. The closeness of these exponents to 1 is an indication of preferential attachment in the evolution of the Drupal network. Preferential attachment is a mechanism that has been shown to result in scale-free characteristics in a variety of complex networks. Valverde and Sole [34]...
observed the scale-free nature of software architectures in a large number of open-source software products.

Mechanism (d): The probabilities with which new nodes are attached with each other are plotted in Fig. 3. As mentioned above, the degrees plotted in the figure are initial degrees obtained after linking the new nodes to existing nodes. From the probability plots, it is observed that for new nodes with high initial degree, the probability of linking to a new node is high compared to those with low degree. We observe that exponential functions provide good approximation of the relationships between the initial degrees and probabilities. The parameters of the exponential functions are shown in Fig. 3.

Mechanism (e): The probabilities of creation of links between two existing nodes are shown in Fig. 4. The existing degrees of nodes are plotted on the x-axis. Exponential functions are fit on the data. The number of interfaces created between two existing nodes is listed in Table 1. It is observed that from version 3 to version 4, the number of links created among existing nodes increases significantly. From version 4 to version 5, the number of links created among existing nodes increases slightly.

Mechanism (f): The probabilities of removal of existing links between nodes as functions of the degrees of nodes are shown in Fig. 5. It is observed that linear trends from these log–log scale plots (indicating power law distribution) can be used to describe the probability functions.

4 Network-Level Analysis of the Evolution of Drupal

4.1 Network Measures. After modeling the product structure as a complex network, the evolutionary characteristics of the corresponding network are explored. Complex network analysis metrics [50] are employed to quantify the evolutionary characteristics of complex product structures. The metrics used in this paper are average degree, degree distribution, density, clustering coefficient, average shortest path, propagation cost, and clustered cost. The first five metrics are extensively used by the network science community to characterize the topologies of complex networks. Propagation cost and clustered cost are used by the OSS community to characterize the complexity of software products.

Average degree is the average number of nearest neighbors of vertices [43]. It is chosen because it represents the average number of other nodes connected to a node, and indicates the complexity of the product [51]. According to existing studies [52,53], the design complexity of a product, which indicates the redesign work in the product development processes, can be quantified as $D(A) = 1/n \sum_{i=1}^{n} Z_i$, where $D(A)$ is the design complexity, $n$ is the number of components, and $Z_i$ is the number of connections (degree) per component $Z_i$ for all components. Design complexity is related to the average degree in the complex network representation. Manufacturing complexity [53] is also related to the average degree and can be expressed as: $M(A) = n + 1/n \sum_{i=1}^{n} Z_i$.
The degree distribution, \( P(k) \) is defined as the fraction of nodes in the network with degree \( k \) [54]. The degree distribution is important because it indicates the topology of a product structure network.

Density of a network is the average proportion of links incident with nodes in a network [50]. The density of a complex network can be expressed as the ratio of the number of links in a network to the number of maximum possible links. Density is an alternative way to represent the complexity of a system. According to Marczyk and Deshpande [55], higher network density implies that higher complexity can be reached.

Clustering coefficient is the probability that two nearest neighbors of a vertex are also the nearest neighbors of one another [54]. Clustering coefficient indicates possible “cliques” with high connections inside and low connections outside. Prior research highlights that a high clustering coefficient is observed in various open-source software projects when compared to their random network counterparts [34]. A high clustering coefficient means the emergence of cliques in the product structure network. The emergence of cliques reduces rework in the development processes because the interactions among cliques are lower. The reduction of rework enables the system to be decoupled into sub-systems for development.

Average shortest path is the average of shortest path lengths that links two vertices in a network [54]. In the product structure network, average shortest path indicates the efficiency of information exchange between two arbitrary nodes. The average shortest path is related to change complexity [53], which describes the likelihood of a change propagating between two components in a product. The value of change complexity is inversely related to the average shortest path [56]. The change complexity can be expressed as

\[
C(A) = \frac{n(n-1)}{2} \sum_{i \neq j}^{n} \frac{d_{ij}}{C^2}
\]

where \( d_{ij} \) is the shortest path between nodes \( i \) and \( j \).

Propagation cost, proposed by MacCormack et al. [2], is a measure of the degree of coupling in a complex system. The metric quantifies the average percentage of other nodes directly or indirectly affected by a change to a node within a network. The metric is based on the concept of visibility of a node in a network, which is the number of other nodes it is directly or indirectly (i.e., through intermediate nodes) connected to. It is calculated as the average “fan-out visibility” of nodes [2].

Clustered cost [2] is another measure of degree of coupling. In contrast to propagation cost where each dependency is assumed to incur the same cost, the assumption in clustered cost is that the dependencies within a cluster incur a lower cost than the dependencies across clusters. In other words, the clustered cost is first clustered, and then weights are assigned to the dependencies depending on the location of the nodes within different clusters. In this paper, we use the Girvan-Newman [57] clustering algorithm to assign nodes into clusters. MacCormack et al. [2] identify a set of nodes, called the vertical bus, consisting of nodes connected to a large number of other nodes. If a given node \( i \) is connected to a node \( j \) in the vertical bus, the dependency cost is a binary variable \( d_{ij} \). If two nodes \( i \) and \( j \) are within a cluster, the dependency cost is given by \( d_{ij} \times n^c \), where \( n \) is the size of the cluster and \( c \) is a user defined parameter (set to 2 in Ref. [2]). For links between nodes across different clusters, the dependency cost is \( d_{ij} \times N \), where \( N \) is the size of the complete network.

4.2 Analysis of Drupal Product Structure. Using the source code, we identify how the characteristics of the product structure change over time. The degree distribution plots for versions 2–5 are displayed in Fig. 6. The degree distributions are plotted on a log–log scale. It is observed that the general forms of the degree distributions for all the versions are similar and are closer to that of a scale-free graph. The degree distribution plots indicate that the majority of nodes have less than 4 interfaces. Beyond a degree of 4, the degree distribution exhibits a power-law trend, indicating a scale-free network topology. Such a scale-free graph property has been found to be a common pattern across many different software applications. Hyland-Wood et al. [58] show that the degree distribution of Gowai 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 [59] show similar trend in the out-degree distribution of Debian product structure. The nodes with more than 20 interfaces are “hubs” in the product structure network. These hubs are analogous to the nodes within a vertical bus, as defined by MacCormack et al. [2].

Other network characteristics for the four versions of Drupal are listed in Table 2. From the number of nodes and interfaces, it is clear that the Drupal project has been constantly growing at a fast pace. Plotting the number of nodes and interfaces (see Fig. 7) reveals that the number of nodes scales linearly with the number of interfaces, indicating a sparse graph. Valverde and Sole observe a similar trend and conclude that the network grows such that on average, new nodes attach to almost a constant number of existing nodes [34]. However, such a conclusion ignores the fact that a large portion of the nodes and links are also removed from the network. In Sec. 5.2, we investigate this in detail.

The average degree of nodes 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. [2] for the structure of Mozilla. The clustering coefficient remains constant and is close to 0.1 for versions 2–5. In Table 3, 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
The addition and removal of individual nodes and links. The mechanisms for the initial product structure networks are chosen: (a) the product structure network from the first version considered (e.g., version 2 in Drupal), (b) a random network with the same number of nodes and links as the product structure network of version 2, and (c) a scale-free network with the same number of nodes and links as the product structure network of version 2. The reason for selecting these 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 studies of network evolution, random networks are extensively used to represent initial network topologies. Scale-free network is used because existing studies (e.g., Ref. [34]) reveal that many real-world networks (including OSS) have the scale-free property. Three time periods are simulated for Drupal: from version 2 to version 3, from version 3 to version 4, and from version 4 to version 5. In each period, the node-level mechanisms discussed in Sec. 3.1 are simulated based on the probability functions discussed in Sec. 3.2.

5.2 Results from the Execution of the Model. The structures of the networks generated using three types of initial networks are compared with the product structure networks from version 2 to version 5. Figure 9 displays a comparison between the characteristics of Drupal product structure and the generated networks over time. An important observation is that the structures of the networks generated from all the three types of initial networks converge to the structure of Drupal as the network evolution takes place. From the figure, it is observed that the average degree of the generated network using the initial version 2 network matches the Drupal project. Initial scale-free and random networks have different average degrees compared to the initial Drupal version 2 network. However, with the evolutionary process, the values of average degrees of generated networks converge to that of the Drupal network. The average densities of three models are similar to the Drupal product (which is because the density is dependent on the numbers of nodes and links only).

The clustering coefficients of models with the initial scale-free network and the initial version 2 network are close to the Drupal product. The model with initial random network has a small clustering coefficient at the beginning, which represents the characteristics of a random network. However, the clustering coefficient significantly increases and converges to the Drupal product over time. The convergence is not obvious because at version 4, the differences between models and Drupal project are larger compared to versions 3 and 5. In this figure, we also observe that the model with the initial random network has a large average shortest path compared to the Drupal project. Finally, it evolves and becomes closer to Drupal. The propagation costs and clustered costs of the Drupal network are close to those of the generated networks for all the versions. A comparison of degree distribution among models with three types of initial networks and Drupal is provided in Fig. 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.

5 Generative Model for 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 Drupal product structure. Figure 8 outlines the execution of the model. The data for mechanisms (a) and (b) are used from Table 1. For mechanisms (c)–(f), the numbers of interfaces created or removed are based on the functions listed in Figs. 2–5. Although the data are collected only for the major versions of the product (V2 → V5) the network is generated gradually through the addition and removal of individual nodes and links. The mechanisms can be used to represent continuous growth of the network for a given version (e.g., V5.1.1 → V5.1.2 etc.). Information about major versions is required for the model because these represent major milestones in the project, and significant changes to the product structure happen at these milestones.

The struc-
average density, clustering coefficient, average shortest path, propagation cost, clustered cost and degree distribution are close to the Drupal product over time, with the use of the proposed 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 mechanisms, the structures of the networks from the models converge to the Drupal product.

6 Application to RepRap 3D Printer: An Open-Source Hardware Product

In this section, we utilize the proposed approach to analyze the structure and evolution of the RepRap 3D printer [11], which is developed though open-source principles. The RepRap project has been under development since 2005. RepRap has been chosen in this paper for three reasons. First, it is one of the most developed and widely used open hardware projects. Second, the evolution of the entire project is well documented online [11]. Third, the detailed design documents, such as computer aided design (CAD)/computer-aided manufacturing (CAM) files, are openly available for download. Three major versions of RepRap: Mendel, Huxley, and Prusa have been used to analyze the evolution. CAD models for these three versions were downloaded from Refs. [11,60]. The product network structure is extracted from the CAD/CAM files by modeling parts as nodes and their physical connections as links.

Compared to open-source software products analyzed in Sec. 5.2, the hardware product has smaller number of nodes and fewer physical connections. On average, RepRap consists of 133 nodes and 205 links. In contrast, Drupal network has more than 500 nodes and 1500 edges. The RepRap network includes main sub-assemblies such as the x-axis system, y-axis system, and z-axis system. Each system consists of more than 30 parts, such as clamps, vertex, sides, bearings, and plates. Within each sub-assembly, a core component such as the x-bar serves as a key node and has physical connections with many other components. The number of nodes added in each version is listed in Table 5.

6.1 Analysis of Node-Level Mechanisms for RepRap. The six node-level mechanisms are used to analyze the product evolution from Mendel to Huxley, and Huxley to Prusa. Table 5 shows the basic data associated with the node-level mechanisms.

These results indicate that the mechanisms can potentially be used to model the evolution of product structures in Drupal. In the case of Drupal, we found that even when the initial product structure is different, if the same mechanisms are applied, the evolution of product structures converges to the same structure over time. This indicates the robustness of the node-level mechanisms in modeling the product evolution.
Addition of new nodes corresponds to addition of new parts. In Reprap, new parts are added when new features and specifications are needed, and when existing parts are redesigned with new features. For example, from Mendel to Huxley, new parts are added to reinforce the base of the printer and to increase its stability for printing larger parts. A comparison between Mendel and Huxley is shown in Fig. 11, which shows new parts, including base bottom, base top, and base reinforcement clamps to increase the stability of the machine.

Removal of existing nodes corresponds to removal of parts from the assembly. In Reprap, existing parts are removed when existing features are no longer needed, or when existing parts are replaced by redesigned parts. From Huxley to Prusa, 26 existing parts have new links with 78 new parts. 22 out of the 78 new parts are connected to two existing x-bars. By analyzing the Mendel product network, it is found that the nodes corresponding to these two x-bars have the highest degree. Each x-bar has a degree of 13, and 10 out of 26 existing nodes have only one new link each. For these 10 existing nodes, the average degree is 3.1, which is lower than the degrees of the x-bars. Figure 12 shows the probabilities of the new links established between new and existing nodes as a function of degree. It is observed that the new nodes have higher probability to connect to existing nodes with higher degree. We do not perform regression of the probability function due to the small sample size.

Linking of new nodes with existing nodes: The new parts can be connected to existing parts, creating links between new and existing parts. From Mendel to Huxley, 26 existing parts have new links with 78 new parts. 22 out of the 78 new parts are connected to two existing x-bars. By analyzing the Mendel product network, it is found that the nodes corresponding to these two x-bars have the highest degree. Each x-bar has a degree of 13, and 10 out of 26 existing nodes have only one new link each. For these 10 existing nodes, the average degree is 3.1, which is lower than the degrees of the x-bars. Figure 12 shows the probabilities of the new links established between new and existing nodes as a function of degree. It is observed that the new nodes have higher probability to connect to existing nodes with higher degree. We do not perform regression of the probability function due to the small sample size.

Linking of new nodes with each other: New nodes may also be connected with each other to create new functionalities. From Huxley to Prusa, 2 new nodes with initial degree of 4, on average have 3 new connections among other new nodes. However, 27 new nodes with initial degree of 1, on average, have 1.2 new connections among new nodes. Figure 12 shows the probabilities of new links established between new nodes.

Linking of existing nodes with each other: With the product evolution, new links are also added between existing nodes. For this mechanism, we do not plot the probability-degree chart since only a few links were created between existing nodes. We observe that from Mendel to Huxley, two new links are established between existing nodes. These two new links are between y-bar clamps and y-bars in order to reinforce the y-bars during the printing process. From

Table 4 P-values from the Chi-square test on degree distributions

<table>
<thead>
<tr>
<th>Model</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial V2 network</td>
<td>—</td>
<td>0.6394</td>
<td>0.1298</td>
<td>0.1777</td>
</tr>
<tr>
<td>Scale-free network</td>
<td>&lt;0.0001</td>
<td>0.0574</td>
<td>0.3430</td>
<td>0.3872</td>
</tr>
<tr>
<td>Random network</td>
<td>&lt;0.0001</td>
<td>0.0023</td>
<td>0.0048</td>
<td>0.6616</td>
</tr>
</tbody>
</table>

(a) Addition of new nodes corresponds to addition of new parts. In Reprap, new parts are added when new features and specifications are needed, and when existing parts are redesigned with new features. For example, from Mendel to Huxley, new parts are added to reinforce the base of the printer and to increase its stability for printing larger parts. A comparison between Mendel and Huxley is shown in Fig. 11, which shows new parts, including base bottom, base top, and base reinforcement clamps to increase the stability of the machine.

(b) Removal of existing nodes corresponds to removal of parts from the assembly. In Reprap, existing parts are removed when existing features are no longer needed, or when existing parts are replaced by redesigned parts. From Huxley to Prusa, parts are combined to make them multifunctional. The redesign of the x-axis system in Prusa significantly reduced the number of parts. In Huxley, the x-axis system consists of 23 parts, including clamps, spacers, bars, belts, bases, carriages, and motor. However, in Prusa, the x-axis system consists of only 8 nodes. The unique carriage part in Prusa is redesigned by combining the functions of clamping, connection, and load bearing into one.
Huxley to Prusa, no new links are established between existing nodes.

(f) Removal of existing links: Existing links can be removed because of the removal of existing nodes, and the removal of links between two existing nodes. Figure 12 shows the probabilities of existing links removed from the network. The results indicate that the probability of a link that is removed has a linear relationship with the degree of the node to which this link attaches. The higher the degree of a node, the more likely its links will be removed. This is because existing links are removed mainly because of the removal of existing nodes. For example, in Reprap, existing links are mainly reduced due to the removal of existing nodes. From Mendel to Huxley, all of the removed links are due to the removal of existing nodes. From Huxley to Prusa, only one link is removed from two existing nodes, while other links are removed due to the removal of existing nodes. The reason behind this is that in a physical product, it is difficult to simplify the existing design by simply removing links between nodes. Instead, in order to simplify the existing designs, the parts also need to be redesigned.

6.2 Analysis of Network Structure and Its Evolution.

Table 6 displays the characteristics of RepRap product structures over time for three major versions. The network metrics show three general trends from Mendel to Prusa. First, there is a reduction in the average degree of the network, which shows a decrease in the number of links between parts. This is mainly a result of integration and combination of parts. Since several parts are redesigned into one part that has multiple functionalities, links that are associated with the old parts are removed as well. Second, the average cluster coefficient decreases. The average cluster coefficient is proportional to the number of triangles in the network. The decrease in cluster coefficient indicates a decrease in the degree of coupling in design. For example, to hold the z-bar in Mendel, two z-bar clamps are designed and connected by fastening bolts. This results in a physical connection between any two parts of the assembly, causing a triangle connection in the network topology.

Since there are many such types of structures in Mendel, the average cluster coefficient is high at 0.217. In contrast, the redesign of parts for multiple functionalities in Huxley and Prusa reduces the triangle connectivity, thus the average cluster coefficient decreases. Third, there is a decrease in average shortest path. The decrease in this metric also indicates the reduction and simplification of parts and connectivity for the product changes in different versions.

Figure 13 displays the degree distribution for three major versions. The degree distributions indicate the similarity in network structure to the open source software, as shown in Fig. 6. Most of the parts have the same number of connections, and there are only few parts (e.g., y-chassis in Mendel, base sheet bottom in Huxley, and x-carriage in Prusa) with many connections. These parts are either chassis working as the central support framework, or the pivot working as a bridge to connect other parts.

In summary, the evolution of an open source hardware product is analyzed by using proposed network-based framework which contains six node-level mechanisms. This case study does not only show that the proposed approach can provide insights about the product structure and patterns in product development but also shows the generality of the proposed framework. As open source hardware matures and products become more complex, regression techniques can be utilized to obtain the probability models for each mechanism so that the evolution of the open source hardware products can be modeled and predicted.

7 Closing Comments

The product structure plays an important role in the product development. In the open-source domain, the product structure affects not only the efficiency of product development [7] but also the community structure [6,61]. Hence, it is important to get an understanding of product structure and evolution in open-source processes. To facilitate that understanding, we present a generative model of the evolution of open-source software products. The model captures the underlying dynamics of evolution of open-source software products. The uniqueness of the proposed model for open-source software evolution is that the dynamics is modeled in terms of the module-level (i.e., local) observations such as addition and deletion of nodes and links. It is shown that applying the 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. Such an evolutionary model based on the local network observations can help in identifying not only the extent of increase in complexity over time but also the mechanisms through which the complexity increases. There are three general applications of the model presented in this paper: (1) longitudinal studies of a...
node-level mechanisms. Different values of parameters for the node-level mechanisms indicate different evolutionary characteristics.

**Predictive analyses:** The third potential application of the model is that it provides the ability to perform what-if analyses by predicting the product structures that could emerge based on certain design decisions. For example, if we assume that the product will evolve in the same manner as it has in the past, we can extrapolate the parameters associated with the node-level mechanisms. The extrapolated parameters can be used in the model to predict the evolution of the product. Specifically, the probability functions can be summarized as:

\[ P = ax^\gamma + by^\beta, \]

where \( x \) and \( y \) are coefficients and \( K \) is the degree. In the prediction process, the fitted curves can be used to predict the coefficients \( x \) and \( y \) for the mechanisms (c)–(f) based on the existing coefficients. Once the predicted coefficients \( x \) and \( y \) are obtained, the predicted product structure for the next version can be determined. Additionally, knowledge about specific design changes to be carried out in the future versions can also be used in node-level mechanisms to predict the global characteristics of future versions of the software.

There are significant opportunities for further research in this direction. First, function–call graphs are used to model the structure of software. The approach can be applied in future to other levels of granularity (files, classes, modules, etc.). Second, in order to understand the network-level impact of the underlying mechanisms, a comprehensive analysis of the effect of the specific functional forms of the probability functions on the network topology is needed. Third, existing research points to the commonalities between the network topologies of software. However, this model can be utilized in future to explore commonalities in evolutionary patterns in terms of the module-level mechanisms. Finally, the focus in this paper is on studying the evolution of product structures only. Further work is needed to model the co-evolution of products and communities of participants [62]. Such analysis is important to validate the mirroring hypothesis [1] according to which, the product structures and community structures mirror each other.

**Acknowledgment**

We gratefully acknowledge the financial support from the National Science Foundation through the CAREER grant number 1265622.

**References**


---

**Table 6 Evolutionary characteristics of RepRap network**

<table>
<thead>
<tr>
<th>Network measure</th>
<th>Mendel</th>
<th>Huxley</th>
<th>Prusa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average degree</td>
<td>3.064</td>
<td>3.086</td>
<td>2.804</td>
</tr>
<tr>
<td>Average density</td>
<td>0.022</td>
<td>0.019</td>
<td>0.029</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>0.217</td>
<td>0.127</td>
<td>0.034</td>
</tr>
<tr>
<td>Average shortest path</td>
<td>7.587</td>
<td>7.574</td>
<td>6.105</td>
</tr>
<tr>
<td>Propagation cost</td>
<td>0.0112</td>
<td>0.0095</td>
<td>0.0143</td>
</tr>
<tr>
<td>Clustered cost</td>
<td>8010</td>
<td>10256</td>
<td>4124</td>
</tr>
</tbody>
</table>