ANALYSIS AND MODELING OF THE PRODUCT STRUCTURE AND COMMUNITY STRUCTURE IN OPEN SOURCE PROCESSES

By

QIZE LE

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

WASHINGTON STATE UNIVERSITY
School of Mechanical and Materials Engineering

August 2012
To the Faculty of Washington State University:

The Members of the Committee appointed to examine the dissertation of QIZE LE find it satisfactory and recommend that it be accepted.

______________________________________________
Jitesh H. Panchal, Ph.D., Chair

______________________________________________
Sankar Jayaram, Ph.D.

______________________________________________
Thomas Rotolo, Ph.D.
ACKNOWLEDGMENT

First of all, I would like to thank my mother and father for all the encouragement and support they have given me to pursue the Ph.D. degree in the United State. Without their continuous financial and emotional support, I could have not achieved any of my goals during the past twenty six years.

I want to thank my advisor - Dr. Jitesh H. Panchal for providing me continuous guidance and advice during the past four years. Dr. Panchal helped me to determine my research topic, gave me valuable advice during my studies, and discussed with me about board research issues and research inspirations. Without his instruction, I could have not achieved the thesis and other papers published in journals and conferences. Therefore, I would like to thank him again for giving me an opportunity to work with him and to conduct new and exciting research topics under his advice.

I would like to thank my committee members Dr. Sankar Jayaram and Dr. Thomas Rotolo for their advice and comments on this thesis. Their comments helped me to improve my work and broaden my knowledge.

I want to thank all of my lab mates - Haoyun Huang, Zhenghui Sha, Yiwen Liu, Bryant Hawthorne, Marc Somers, and Martin Baker. Haoyun Huang and Zhenghui Sha has provided me many valuable comments and thoughts on my research and published paper. Bryant spent a lot of time on discussing with me about research and polishing my English. For all the other lab mates, I would like to thank them as being a big family of Collective Systems Laboratory. Also, I would like to thank all of my friends in China and United States

I would like to thank School of Mechanical and Materials Engineering for providing
me admission and assistantship for my Ph.D. studies.

This work presented in the thesis has been supported by NSF CAREER # 0954447. I am grateful for the financial support from NSF, without which this work would not be finished.
Open source product development refers to a paradigm where large numbers of participants collaborate in the form of communities to develop new products and services. The fundamental differences between traditional product development and open source process are that the former is based on top-down decomposition while the latter is based on evolution and self-organization. The paradigm of open source processes has resulted in highly successful products such as Linux and Apache. Despite the success of various projects using open source processes, it is not well understood how the product structures and community structures evolve over time.

In response, the main objective in this thesis is to analyze and model product structures and community structures of open source processes over time. The research objective consists of three parts: 1) the characterization of product structures in the open
source processes, 2) the evolutionary characteristics of product structures in the open source processes over time, 3) the co-evolution of product structure and community structures over time. To achieve the first research objective, an agent-based model is proposed to simulate the open source processes. The characterization of product structures on the open source process is analyzed based on a mobile phone example. To achieve the second research objective, an open source software project - Drupal is employed as the case studies. The evolution of product structure is determined by node-level mechanisms. To achieve the third research objective, the co-evolution of product structures and community structures is studied at different levels. The results from this research include new knowledge about the evolution of the product structures and co-evolution of product and community structures over time in open source processes.
# Table of Contents

ACKNOWLEDGMENT ................................................................................................................ iii

ABSTRACT .................................................................................................................................... v

LIST OF TABLES ......................................................................................................................... ix

LIST OF FIGURES ....................................................................................................................... xii

1. CHAPTER ONE - INTRODUCTION TO OPEN SOURCE PROCESSES AND RESEARCH QUESTIONS ............................................................................................................. 1
   Section 1.1 - Introduction to Open Source Processes ......................................................................................... 2
   Section 1.2 - Research Motivation .................................................................................................................. 5
   Section 1.3 - Research Questions, Approaches, Tasks, and Contributions ....................................................... 8
   Section 1.4 - Assumptions in the Open Source Processes ................................................................................ 19
   Section 1.5 - Overview of the Dissertation .................................................................................................. 20

2. CHAPTER TWO - BACKGROUND AND LITERATURE REVIEW ................................ 25
   Section 2.1 - Existing Studies on Product Structures in Open Source Processes ................................................... 27
   Section 2.2 - Existing Studies on Community Structures in Open Source Processes ................................................. 35
   Section 2.3 - Existing Studies on the Network Generation Model ....................................................................... 38
   Section 2.4 - Existing Studies on Interdependencies between Product Structure and Community Structure ................................................................. 41
   Section 2.5 - Research Gaps .......................................................................................................................... 46

3. CHAPTER THREE - ANALYSIS AND MODELING EFFECT OF PRODUCT STRUCTURE ON OPEN SOURCE PROCESSES ........................................................................................................ 51
   Section 3.1 - Motivation: Answering RQ1 ........................................................................................................... 51
   Section 3.2 - Background: Agent-based Modeling (ABM) ................................................................................ 53
   Section 3.3 - Details of the Agent-Based Model of open source processes ........................................................... 54
   Section 3.4 - Illustrative Example – Development of Mobile Phones using Mass-Collaborative Product Development Processes ...................................................... 63
   Section 3.5 - Results from the Execution of the Model for different Mobile Phone architectures ........................................... 69
   Section 3.6 - Discussion and Future Works .................................................................................................. 80

4. CHAPTER FOUR - ANALYSIS AND MODELING PRODUCT STRUCTURE ON OPEN SOURCE PROCESSES ................................................................................................................................ 83
   Section 4.1 - Motivation: Answering RQ2 ........................................................................................................... 83
   Section 4.2 - Node-level Mechanisms Associated with Product Evolution .......................................................... 85
   Section 4.3 - Network-Level Analysis of the Evolution of Drupal ..................................................................... 92
   Section 4.4 - Generative Model for the Evolution of Product Structure .............................................................. 99
   Section 4.5 - Conclusion and Future Works ................................................................................................ 108
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Summary of Research Issues in Open Source Processes</td>
<td>4</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Research Focus in This Dissertation</td>
<td>5</td>
</tr>
<tr>
<td>Table 1.3</td>
<td>Central Research Question and Approach</td>
<td>8</td>
</tr>
<tr>
<td>Table 1.4</td>
<td>Research Questions and Approaches</td>
<td>9</td>
</tr>
<tr>
<td>Table 1.5</td>
<td>Approaches Towards Answering the Research Question 3 at Three Levels</td>
<td>17</td>
</tr>
<tr>
<td>Table 1.6</td>
<td>Mapping Among Research Questions, Approaches, Tasks and Contributions</td>
<td>18</td>
</tr>
<tr>
<td>Table 1.7</td>
<td>Structure of Chapter 2 - Background and Literature Review</td>
<td>21</td>
</tr>
<tr>
<td>Table 1.8</td>
<td>Framework of the Research Tasks in the Dissertation</td>
<td>21</td>
</tr>
<tr>
<td>Table 2.1</td>
<td>Literature Review, Research Limitations and Research Approaches</td>
<td>26</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Example of a Design Structure Matrix</td>
<td>29</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Literature on the Analysis of the Structure and Evolution of Open Source Software</td>
<td>32</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Summary of Degree-based Network Models</td>
<td>40</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>A Summary of the Key Studies on the Interrelationships Between Products and Organizational Structures in Software Products</td>
<td>42</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>A Numerical DSM with Rework (W: Weak Dependence with Rework Percentage = 0.1%, M: Medium Dependence with Rework Percentage = 0.3%, S: Strong Dependence with Rework Percentage = 0.5%)</td>
<td>65</td>
</tr>
</tbody>
</table>
TABLE 3.2 - DIFFERENT PRODUCT STRUCTURES GENERATED BY SEQUENTIALLY DECOUPLING DIFFERENT COMPONENTS OF MOBILE PHONES ............................................. 69

TABLE 3.3 - THREE DECOUPLING SEQUENCES CONSIDERED FOR ANALYSIS ......................................................... 74

TABLE 4.1 - DATA ASSOCIATED WITH NODE-LEVEL MECHANISMS FOR DRUPAL NETWORK (VERSION 2 THROUGH 5) ............................................................................................................. 90

TABLE 4.2 - CHARACTERISTICS OF DIFFERENT VERSIONS OF THE DRUPAL PRODUCT STRUCTURE ....................................................................................................................... 97

TABLE 4.3 - COMPARISON OF CLUSTERING COEFFICIENT BETWEEN PRODUCT STRUCTURE NETWORK AND CORRESPONDING RANDOM GRAPHS .......................................................... 98

TABLE 4.4 - DATA RELATED TO NODE-LEVEL MECHANISMS IN WORDPRESS ................................................. 104

TABLE 4.5 - DATA RELATED TO NODE-LEVEL MECHANISMS IN PHPMYADMIN ........................................... 106

TABLE 5.1 - FIVE HYPOTHESES AT THREE LEVELS ................................................................................................. 112

TABLE 5.2 - DIFFERENT ASPECTS CORRESPONDING TO THE HYPOTHESES ......................................................... 113

TABLE 5.3 - BASIC INFORMATION ABOUT THE NETWORKS GENERATED .............................................................. 123

TABLE 5.4 - REGRESSION MODEL CORRESPONDING TO HYPOTHESIS 3.1.1 ................................................. 124

TABLE 5.5 - LINEAR REGRESSION MODEL CORRESPONDING TO HYPOTHESIS 3.1.2 .......... 125

TABLE 5.6 - REGRESSION MODEL CORRESPONDING TO HYPOTHESIS 3.2.1 ................................................. 126

TABLE 5.7 - REGRESSION MODEL CORRESPONDING TO HYPOTHESIS 3.2.2 ................................................. 127

TABLE 5.8 - CONGRUENCE MEASURED USING APPROACH BY CATALDO ET AL. [34, 47] AND MATCHING OF INTERACTIONS USING APPROACH BY SOSA [48] ................................................. 129

TABLE 5.9 - SOCIO-TECHNICAL CONGRUENCE AND NUMBER OF PARTICIPANTS IN THE CORE AND PERIPHERY ..................................................................................................................... 130
TABLE 6.1 - COMPARISON OF KEY FEATURES OF THE NETWORKS OBTAINED BY DIFFERENT LINKING APPROACHES

TABLE 6.2 - TARGET NUMBERS OF NODES AND LINKS IN THE RESIZED NETWORK

TABLE 6.3 - KEY FEATURES OF THE SMALLER SURROGATE NETWORKS (100% - 25% SIZE) USING JOINT DEGREE DISTRIBUTION APPROACH

TABLE 6.4 - BASIC PARAMETERS OF THREE BIPARTITE NETWORKS

TABLE 7.1 - DIFFERENCES BETWEEN OPEN SOURCE SOFTWARE AND OSH

TABLE 7.2 - RESEARCH QUESTIONS IN STUDIES OF OSH PRODUCT STRUCTURES

TABLE 7.3 - RESEARCH QUESTIONS FOR STUDIES OF OSH COMMUNITY STRUCTURES

TABLE 7.4 - THE RESEARCH QUESTIONS FOR STUDIES OF COORDINATION BETWEEN OSH PRODUCT STRUCTURE AND COMMUNITY STRUCTURE

TABLE 7.5 - THE RESEARCH ISSUES FOR DEVELOPMENT OF OSH MANAGEMENT TOOL

TABLE 7.6 - THE RESEARCH QUESTIONS FOR MODELING
LIST OF FIGURES

FIGURE 1.1 - GROWTH OF THE OPEN SOURCE PROCESSES .............................................. 1

FIGURE 1.2 - EXAMPLES OF OPEN SOURCE PROCESSES IN DIFFERENT AREAS ................ 3

FIGURE 1.3 - DIFFERENCES BETWEEN OPEN SOURCE AND TRADITIONAL PRODUCT
DEVELOPMENT PROCESSES ............................................................................................ 4

FIGURE 1.4 - THE AGENT-BASED MODEL TO SIMULATE OPEN SOURCE PROCESSES .... 12

FIGURE 1.5 - RESEARCH APPROACH 2 AND APPROACH 2 ............................................ 15

FIGURE 1.6 - ROAD MAP OF THE DISSERTATION .......................................................... 24

FIGURE 3.1 - HOW THE PROPOSED APPROACH ANSWERS RQ1 BY APPROACH 1 ......... 52

FIGURE 3.2 - AN EXAMPLE OF A PRODUCT MODEL FOR A MOBILE PHONE .................. 56

FIGURE 3.3 - ILLUSTRATION OF REWORK .................................................................. 57

FIGURE 3.4 - ILLUSTRATION OF MODULE MODULARITY ............................................. 58

FIGURE 3.5 - EXAMPLE OF THE EVOLUTION OF DIFFERENT MODULES ..................... 61

FIGURE 3.6 - EXAMPLE OF EVOLUTION OF THE ENTIRE PRODUCT OVER TIME ............ 61

FIGURE 3.7 - DECOMPOSITION OF THE MOBILE PHONE TO IDENTIFY ITS MODULES ....... 64

FIGURE 3.8 - EXAMPLES OF A) STRONG DEPENDENCE: THE UPPER CASE AND THE LOWER
CASE, B) MEDIUM DEPENDENCE: THE UPPER CASE AND THE MAIN BOARD, AND C)
WEAK DEPENDENCE: THE UPPER CASE AND THE ANTENNA ..................................... 66

FIGURE 3.9 - EVOLUTION OF MODULES IN STRUCTURE A ............................................. 71

FIGURE 3.10 - EVOLUTION OF MODULES IN STRUCTURE J .......................................... 71

FIGURE 3.11 - COMPARISON OF EVOLUTION OF PRODUCTS WITH DIFFERENT STRUCTURES 71

FIGURE 3.12 - THE RELATIONSHIP OF DEGREE OF MODULARITY AND EVOLUTION TIME ..... 72
FIGURE 3.13 - THE EFFECT OF MODULE MODULARITY ON THE EVOLUTION TIME OF THREE

MODULES .................................................................................................................. 74

FIGURE 3.14 - COMPARISON OF DIFFERENT DECOUPLING SEQUENCES ...................... 76

FIGURE 3.15 - THE EFFECT OF CHANGING (A) NUMBER OF PARTICIPANTS, (B) VALUE, AND (C)

COST ON THE GROWTH OF PRODUCTS WITH STRUCTURE J ......................................... 79

FIGURE 4.1 - ILLUSTRATION OF ANSWERING RQ2 IN CH4 ........................................ 85

FIGURE 4.2 - NODE-LEVEL MECHANISMS FOR THE EVOLUTION OF PRODUCT NETWORKS ...

86

FIGURE 4.3 - PROBABILITY OF NEW NODES LINKING WITH EXISTING NODES: AN EXAMPLE 87

FIGURE 4.4 - PROBABILITIES OF CREATION OF LINKS BETWEEN NEW AND EXISTING NODES

(MECHANISM (C)) .......................................................................................................... 91

FIGURE 4.5 - PROBABILITY OF NEW NODES LINKING WITH EACH OTHER (LINKING

MECHANISM (D)) ............................................................................................................ 91

FIGURE 4.6 - PROBABILITY OF EXISTING NODES LINKING WITH EACH OTHER (LINKING

MECHANISM (E)) ............................................................................................................ 92

FIGURE 4.7 - PROBABILITY OF REMOVAL OF EXISTING LINKS (MECHANISM (F)).......... 92

FIGURE 4.8 - DEGREE DISTRIBUTION OF PRODUCT ARCHITECTURE NETWORKS FROM

VERSION 2 TO VERSION 5 .......................................................................................... 96

FIGURE 4.9 - RELATIONSHIP BETWEEN THE NUMBER OF FUNCTIONS AND INTERFACES ...

97

FIGURE 4.10 - THE EXECUTION OF THE MODEL BASED ON MECHANISMS ...................... 100

FIGURE 4.11 - COMPARISON OF EVOLUTIONARY CHARACTERISTICS AT PRODUCT LEVEL

BETWEEN DRUPAL PRODUCT AND THE MODELS ....................................................... 102

FIGURE 4.12 - COMPARISON OF DEGREE DISTRIBUTION BETWEEN DRUPAL PRODUCT AND

THE MODELS ............................................................................................................... 103
FIGURE 4.13 - COMPARISON OF EVOLUTIONARY CHARACTERISTICS BETWEEN WORDPRESS PRODUCT AND THE GENERATED NETWORKS ................................................................. 106

FIGURE 4.14 - COMPARISON OF DEGREE DISTRIBUTION BETWEEN WORDPRESS PRODUCT AND THE GENERATED NETWORKS ................................................................. 107

FIGURE 4.15 - COMPARISON OF EVOLUTIONARY CHARACTERISTICS BETWEEN PHPMYADMIN PRODUCT AND THE GENERATED NETWORKS ................................................................. 107

FIGURE 4.16 - COMPARISON OF DEGREE DISTRIBUTION BETWEEN PHPMYADMIN PRODUCT AND GENERATED NETWORKS .................................................................................. 107

FIGURE 5.1 - ILLUSTRATION OF ANSWERING RQ3 BY APPROACH 3 ........................................ 114

FIGURE 5.2 - A HYBRID NETWORK MODEL TO REPRESENT PRODUCT STRUCTURE, COMMUNITY STRUCTURE AND PARTICIPATION ................................................................. 116

FIGURE 5.3 - EXTRACTING THE RELATIONSHIP BETWEEN THE NUMBER OF NEWLY CREATED LINKS OF PARTICIPANTS AND NEWLY CREATED LINKS OF MODULES ASSOCIATED WITH THAT PARTICIPANT (FOR HYPOTHESIS 3.1.2) ................................................................. 117

FIGURE 5.4 - 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 3.2.2) ................................................................. 118

FIGURE 5.5 - CLUSTER COMPARISON APPROACH .................................................................. 120

FIGURE 5.6 - GENERATION OF THE DRUPAL NETWORKS WITH THREE TYPES OF LINKS ..... 123

FIGURE 5.7 - NEW INTERFACES OF FILES VS. AVERAGE NUMBER OF NEW COMMUNICATION LINKS OF THE CORRESPONDING PARTICIPANT SETS ....................................................... 124

FIGURE 5.8 - NEW COMMUNICATION LINKS VS. THE AVERAGE NUMBER OF NEW INTERFACES OF FILES IN THE CORRESPONDING MODULE SETS ....................................................... 125
FIGURE 5.9 - NEW INTERFACES BETWEEN TWO FILES VS. AVERAGE NUMBER OF NEW COMMUNICATION LINKS BETWEEN THE CORRESPONDING PAIR OF PARTICIPANT SETS ............................................................................................................................... 126

FIGURE 5.10 - NEW COMMUNICATION LINKS BETWEEN TWO INDIVIDUALS VS. AVERAGE NUMBER OF NEW INTERFACES BETWEEN CORRESPONDING PAIR OF MODULE SETS .. 127

FIGURE 5.11 - CORE AND PERIPHERAL NODES IN THE COMMUNITY (VERSION 5).......... 130

FIGURE 5.12 - 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................. 132

FIGURE 5.13 - SIZES OF CLUSTERS BASED ON THE CLUSTERING OF DIFFERENT NETWORKS ............................................................................................................................... 132

FIGURE 6.1 - OVERVIEW OF THE APPROACH ............................................................... 140

FIGURE 6.2 - DEGREE DISTRIBUTION OF TYPE A AND TYPE B NODES ................. 141

FIGURE 6.3 - THE DENSITY OF POINTS IN DEGREE DISTRIBUTION OF NODES A ....... 143

FIGURE 6.4 - COMBINATION OF TYPE A AND TYPE B NODES ............................... 145

FIGURE 6.5 - DIVISION OF NETWORK AND COMPONENTS’ INFORMATION ABOUT LINKS’ COMBINATIONS ...................................................................................................................... 147

FIGURE 6.6 - CONVERTING BIPARTITE NETWORK TO THE NETWORK WITH SINGLE TYPE OF NODES ...................................................................................................................... 149

FIGURE 6.7 - NORMALIZED LOG-LOG PLOTS OF PEOPLE AND PROJECT DEGREE DISTRIBUTIONS ...................................................................................................................... 151

FIGURE 6.8 - DENSITY OF POINTS IN PEOPLE AND PROJECT DISTRIBUTIONS .......... 151
FIGURE 6.9 - COMPARISON BETWEEN RECONSTRUCTED DEGREE DISTRIBUTIONS AND
ORIGINAL ONES ....................................................................................................... 152

FIGURE 6.10 - COMPARISON OF DEGREE CORRELATION BETWEEN ORIGINAL AND
SURROGATE BIPARTITE NETWORKS OF IDENTICAL SIZE ....................................... 153

FIGURE 6.11 - PEOPLE-People NETWORK DEGREE DISTRIBUTION ............................... 158

FIGURE 6.12 - COMPARISON BETWEEN RECONSTRUCTED 75% DEGREE DISTRIBUTIONS AND
ORIGINAL ONES ....................................................................................................... 159

FIGURE 6.13 - COMPARISON BETWEEN RECONSTRUCTED 50% DEGREE DISTRIBUTIONS AND
ORIGINAL ONES ....................................................................................................... 160

FIGURE 6.14 - COMPARISON BETWEEN RECONSTRUCTED 25% DEGREE DISTRIBUTIONS AND
ORIGINAL ONES ....................................................................................................... 160

FIGURE 6.15 - COMPARISON OF DEGREE CORRELATION OF BIPARTITE NETWORK WITH 75%,
50% AND 25% SIZES OF THE ORIGINAL BIPARTITE NETWORK USING JOINT DEGREE
DISTRIBUTION APPROACH ...................................................................................... 161

FIGURE 6.16 - COMPARISON BETWEEN SURROGATE AND ORIGINAL NETWORKS (THREE
TOPOLOGIES) ........................................................................................................ 164

FIGURE 7.1 - DEVELOPMENT OF OSH OVER PERIODS ............................................. 174

FIGURE 7.2 - CHALLENGES FOR OSH PRODUCT DEVELOPMENT .............................. 176
Dedication

This dissertation is dedicated to my mother and father who provided
both emotional and financial support
1. CHAPTER ONE - INTRODUCTION TO OPEN SOURCE PROCESSES AND RESEARCH QUESTIONS

The principal goal in this dissertation is to analyze the evolution and co-evolution of product structure and community structure in open source processes, to reveal the unique trends of evolution and co-evolution of product and community at different aspects and levels, and to utilize the revealed trends and mechanisms to simulate and predict the evolution and co-evolution of product and community structure.

Open source processes have emerged as novel product development processes, and have generated significant interest in academia, industry and business. The development of open source processes is listed in Figure 1.1.

![Growth of the Open Source Processes](image)

**Open Source Software**
- Linux, Drupal, Sourceforge
- 1983 ~ Now
- Thousands of successful projects, highly developed communities.

**Open Source Hardware**
- Open CPU, Arduino
- 1998 ~ Now
- Several successful projects, developing communities

**Open Source Design**
- 3D Printer, Car Design
- 2005 ~ Now
- Most projects are still in early development. Lack of communities.

Figure 1.1 - Growth of the open source processes

From an academic perspective, open source processes have attracted increasing attention from many fields, including mechanical engineering, software engineering, social science and organization science. Further evidence of the growing importance in this field is the fact that number of successful applications of open source processes in software [1, 2] and hardware [3, 4] have increased. In spite of these diverse efforts, there is a critical gap in the knowledge base that pertains to designing efficient and successful
application of open source processes. This gap in knowledge limits the success of applications of open source processes. This dissertation contributes quantitative approaches to model and analyze product structures and the community structures based on existing open source projects.

In Section 1.1, the open source processes are introduced. A comparison between open source processes and the traditional processes is made. Examples of open source processes are listed and introduced. In Section 1.2, research questions regarding the evolution and co-evolution of product structure and community structure in open source processes are posed, and approaches towards answering research questions are introduced. In Section 1.3, research tasks towards the proposed approaches are introduced, and the contributions of the research tasks are presented.

Section 1.1 - Introduction to Open Source Processes

1.1.1. Introduction to Open Source Processes

Open source product development refers to community led innovation in which the product released under a license that permits inspection, use, modification, and redistribution. [5]. In open source processes, masses of individuals with different backgrounds, knowledge and experiences participate in product development. Open source processes are not driven by top-down flow of control. Instead, individuals choose their tasks based on their own goals and interests. Successful examples of open source processes include open-source software development (e.g., Linux and Apache), open-source hardware development (e.g., OpenCPU, Arduino), and open-source physical product development (e.g., Open Source Car and RepRap 3D printer). The success of
these projects is defined in terms of their steady growth, significant adoption, and large supporting communities. Open source processes have recently been applied to physical product development also, one example being the Oscar project (i.e., the Open Source Car project) [4]. The objective in the Oscar project is to develop a car using open source principles. Other examples are the Open Prosthetics project [6], RepRap 3D printer [7], and open camera [8]. In physical products, the attempts to open source product development are still in the early stages. Figure 1.2 displays the examples of open source processes in different area.

![Diagram of open source processes](image)

**Figure 1.2 - Examples of open source processes in different areas**

The differences between open source processes and traditional processes are discussed in detail by Panchal and Fathianathan [9, 10]. Traditional design processes are described as top-down processes where the information flows logically from the desired functionality of a product to a design that satisfies the functionality. Top-down processes are embodied in models for systems design such as Systems Engineering Vee model [11] and systematic design approaches such as the Pahl and Beitz [12] design method particularly geared towards designing engineering systems. These processes are based on the assumption that a product can be decomposed into subsystems. The subsystems are
generally developed by different teams and are integrated into a complete system. In contrast, open source processes are characterized by massively decentralized activities and decision-making. Products are developed without explicit hierarchical decomposition and assignment of tasks to different teams. Instead, large independent groups of participants participate in open source processes entirely based on their individual interests. Their decisions are based on their individual benefits and costs. A key difference between traditional and open source processes is that products evolve as a result of independent contributions from different participants. The product is always under continuous development and evolution. Such evolutionary design processes are viewed by Klein et al. [13] as dynamical systems. Figure 1.3 displays the key differences between open source processes and traditional product development processes.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Traditional Process</th>
<th>Open Source Process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design Process</strong></td>
<td>Top – down process</td>
<td>Decentralized decision making</td>
</tr>
<tr>
<td><strong>Product Decomposition</strong></td>
<td>Based on functions</td>
<td>Based on individual’s benefit and interest</td>
</tr>
<tr>
<td><strong>Team Structure</strong></td>
<td>Hierarchy</td>
<td>Flat and community</td>
</tr>
<tr>
<td><strong>Development process</strong></td>
<td>Product life cycle</td>
<td>Evolutionary process</td>
</tr>
</tbody>
</table>

**Figure 1.3 - Differences between Open Source and Traditional Product Development Processes**

From the standpoint of engineering design, research on open source processes is still in its early stages. Panchal and Fathianathan, and Panchal [9, 10] discuss the open research issues in the successful application of open source processes. Table 1.1 provides a summary of research issues in open source processes.

**Table 1.1 - Summary of Research Issues in open source processes**
<table>
<thead>
<tr>
<th>Research Area</th>
<th>Research Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Related Research</td>
<td>Product Structure and Evolution</td>
</tr>
<tr>
<td></td>
<td>Collaborative Design</td>
</tr>
<tr>
<td></td>
<td>Services</td>
</tr>
<tr>
<td>Organization Related Research</td>
<td>Organizational Structure and Evolution</td>
</tr>
<tr>
<td></td>
<td>Participants Incentives</td>
</tr>
<tr>
<td></td>
<td>Collective Learning and Evolution</td>
</tr>
<tr>
<td></td>
<td>Psychological and Sociological Issues</td>
</tr>
<tr>
<td>Coordination Between Product and</td>
<td>Co-Evolution between Product and Organization</td>
</tr>
<tr>
<td>Organization</td>
<td>Effect of Product Structure on Organization</td>
</tr>
<tr>
<td></td>
<td>Effect of Organizational Structure on Product</td>
</tr>
<tr>
<td>Information and Computation</td>
<td>Collective Platform</td>
</tr>
<tr>
<td></td>
<td>Product Data Management</td>
</tr>
<tr>
<td></td>
<td>Product Lifecycle Management</td>
</tr>
</tbody>
</table>

### 1.1.2. Focus of this Dissertation

In this dissertation, the primary research focus is on two areas: 1) product, and 2) co-evolution between product and community. In the product area, the key research interests are: 1) the product structures 2) the product evolution emerged in open source processes. In the co-evolution between product and community area, the key research interest is to analyze and identify the co-evolution between product structure and community structure at different levels. Table 1.2 displays the research focus in this dissertation.

### Table 1.2 - Research focus in this dissertation

<table>
<thead>
<tr>
<th>Research Area</th>
<th>Research Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Related Research</td>
<td>Product Structure</td>
</tr>
<tr>
<td></td>
<td>Product Evolution</td>
</tr>
<tr>
<td>Co-evolution of Product and Community</td>
<td>Co-Evolution between Product Structure and Community Structure at different level</td>
</tr>
</tbody>
</table>

### Section 1.2 - Research Motivation

Because of the early stage of open source processes and insufficient data to analyze,
there is still a lack of knowledge in two research areas: 1) product, and 2) co-evolution of product and community, discussed in Section 1.1.2. The motivation of this dissertation is to obtain knowledge about product structure, evolution, and co-evolution between product and community in open source processes by studying open source case studies, modeling product and community evolution, and simulating the open source development processes.

The product structures in traditional product development processes are studied by researchers based on Design Structure Matrix [14-16]. The product structures are characterized by the modularity metric [17]. In the open source processes, studies of product structure are conducted by analyzing open source software projects. The product structures are modeled as networks with nodes and links, where nodes represent modules, functions or files, and links represents interfaces or calls between modules, functions or files. Quantitative metrics to measure product structures, including propagation and clustering cost [18], connectivity [18], change ratio [19], and complex network metrics [20], are proposed by researchers. However, the characterization of product structures in the open source processes has not yet been discussed. The knowledge gap exists in the characterization of product structures in open source processes. Once the product structures are characterized, the study of how different types of product structures affect the completion time of open source product development can be conducted. The knowledge of the relationships between product structures and completion time can be used for designing product structures in open source processes.

In open source processes, product structure evolves over time. The evolution of product structure is analyzed in existing studies by comparisons of network metrics over
time [19, 21-23]. Existing studies illustrate that different product structures emerge, but have not yet illustrated why these product structures emerge. The underlying mechanisms, which determine the evolutionary patterns of product structures, have not yet been identified. The knowledge gap exists in the identification of underlying mechanisms and the characterization of evolution of product structures. Once the underlying mechanisms are identified, the evolution of product structures can be characterized. The knowledge of the underlying mechanisms and product evolution characterization can serve as guidelines for directing product development in open source processes.

The open source community is modeled as networks with nodes and links, where nodes represent individuals, links represent communication among individuals. The community networks are studied by researchers both qualitatively [24, 25], and quantitatively [26-30]. However, these existing studies of open source community only focus on community structures and evolution. The co-evolution of product and community has not yet been taken into account. The co-evolution of product and organizational structures are proposed by Conway [31]. The studies of co-evolution of product and organizational structures are mainly focused on three aspects: 1) inter-relationships between product and organization [32], 2) the mirroring hypothesis [33], and 3) socio-technical congruence [34]. However, the studies are based on static views of product and community networks. The analysis displays what extent the products and organizations mirror each other. The underlying mechanisms that lead to the observed evolutionary characteristics are not revealed. The knowledge gap exists in characterization of co-evolution of product and community by the underlying mechanisms. Once the underlying mechanisms are revealed, the co-evolution of product
and community can be characterized. The design of product and community can be achieved by following the underlying mechanisms in product evolution and co-evolution of product and community.

**Section 1.3 - Research Questions, Approaches, Tasks, and Contributions**

In order to resolve the gaps identified in the previous section, the central research question is proposed in Table 1.3.

**Table 1.3 - Central Research Question and Approach**

<table>
<thead>
<tr>
<th>Central Research Question:</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can the structure and evolution of product and community in open source processes be characterized?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Central Research Approach:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The structure and evolution of product and community in open source processes are characterized by using techniques, such as agent-based modeling, complex network analysis, and community detection.</td>
</tr>
</tbody>
</table>

The approach towards answering the central research question is derived from published research on product structure and system integration, open-source software (OSS) development communities [10, 35-37]. Once the approach is applied to the open source case studies, the structure and evolution of product and community can be characterized.

The central research question can be divided into three research questions based on our research interest and focus. Research Question 1 is proposed to resolve the knowledge gap in the studies of product structure in open source processes (as discussed
in Section 1.3.1). Research Question 2 is proposed to resolve the knowledge gaps in the studies of product evolution in open source processes (as discussed in Section 1.3.2). Research Question 3 is proposed to resolve the knowledge gaps in the studies of co-evolution of product and community in open source processes (as discussed in Section 1.3.3). Table 1.4 displays the three research questions and corresponding approaches.

Table 1.4 - Research Questions and Approaches

<table>
<thead>
<tr>
<th>Research Question and Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RQ1</strong>: How can the structure of a product be characterized in open source processes?</td>
</tr>
<tr>
<td><strong>Approach 1</strong>: Applying agent-based model, and to characterize the product structure by Degree of Modularity [14] (DOM) and completion time.</td>
</tr>
<tr>
<td><strong>RQ2</strong>: How can the evolution of product structure be characterized in open source processes?</td>
</tr>
<tr>
<td><strong>Approach 2</strong>: Modeling products as networks with nodes and links, and to characterize the evolution of product structure using node level mechanisms.</td>
</tr>
<tr>
<td><strong>RQ3</strong>: How can the co-evolution of product and community be identified in open source processes?</td>
</tr>
<tr>
<td><strong>Approach 3</strong>: Modeling the product and community as hybrid networks with nodes and links, and to identify the co-evolution of product and community by node level, link level and cluster level mechanisms.</td>
</tr>
</tbody>
</table>

1.3.1. Research Question 1, Approach 1, Task 1 and Contributions

The Research Question 1 is: How can the structure of a product be characterized in open source processes? The corresponding approach towards answering research question 1 is to apply agent-based model, and to characterize the product structure by
Degree of Modularity [14] (DOM) and completion time.

The product structure has been well characterized in traditional product development processes. Ulrich [38] presents four different types of product structures, namely, integral, slot, bus, and sectional structures. Pimmler and Eppinger [16] analyze product decomposition and discuss its effects on the traditional product development processes using an automotive climate control system example. Fixson [39], Pimmler and Eppinger [16] and Ulrich [38] analyze product structures from both conceptual and computational standpoints. Yassine and Wissmann [40] point out that designing the product structures could lead to effective strategies for increasing service levels. In these studies, it has been well established that in traditional product development processes, product structures can be characterized based on its effect on the product development time and cost, number of iterations, convergence behavior, and adaptability of the product. Modular product structures significantly improve the efficiency of traditional product development processes by reducing the number of iterations and associated development time and cost because of clear partitioning of sub-functions resulting in fewer dependencies between corresponding tasks.

While various studies related to product structures have been conducted for traditional product development, the product structures have not yet been characterized in open source processes. The analysis is important because it would facilitate: a) the identification of types of products that are suitable for open source processes, and b) modification of product structures for existing systems to support open source processes. In this dissertation, my approach is to apply agent-based model, and to characterize the product structure by Degree of Modularity [14] (DOM) and completion time. In the
agent-based model, we simulate the behavior of participants as they contribute to product
development efforts, and the resulting product evolution. By simulating the open source
processes, the product structure can be characterized based on the open source product
development time.

In Research Task 1, a step towards addressing this research question is presented. In
this task, the agent-based computational modeling [41] is used for simulation purpose.
The agent-based model is constructed to simulate the open source product development
processes. In the agent-based model, we simulate the behavior of participants as they
contribute to product development efforts, and the resulting product evolution. By
simulating the open source processes, the effect of product structure on the evolution of
products in open source processes is studied. Here, evolution refers to the incremental
growth of the product in the development process. Figure 1.4 illustrates simulation of the
open source product development processes by agent-based modeling. In this figure, the
product structure is presented as a network, where the blue circles represent modules and
links represent interfaces between modules. Individuals work on the product until it is
complete. Individuals decide whether to work on a module or not based on their own
costs and benefits. The DOM is calculated based on product structure. The relationship
between DOM and product development time is studied.

![Overview of the Model](image)
The contributions from Research Task 1 are 1) a general approach towards simulating the open source product development processes, which can be used in further studies of open source processes, 2) characterizing the product structures based on DOM, which affects the product development time in open source processes, 3) a computational model, which can be refined for studying different aspects of open source processes, such as product evolution, community structure and evolution. In Chapter 3, the Research Question 1 is answered by the proposed agent-based modeling. The results in Chapter 3 show that the product structures are characterized by DOM and completion time. The product with high DOM has low completion time in the open source product development processes.

1.3.2. Research Question 2, Approach 2, Task 2 and Contributions

The Research Question 2 is: how can the evolution of product structure be characterized in open source processes? The corresponding Approach to answer the research question 2 is: To model products as networks with nodes and links, and to characterize the evolution of product structure using node level mechanisms.

The evolution of products in open source processes has received significant attention from multiple communities. The evolution of product structure has impacts on its quality, maintainability, and even the community structure [18, 33, 42]. Some of the questions of interest while analyzing the evolution of product structure are: a) how modular is the product? b) how complex is the product? c) is this product more modular/complex than other open source products? d) are open source products more modular than those developed by teams within software companies? e) does the modularity/complexity of the
product increase/decrease with time (if so, by how much)? f) what is the impact of modularity on the rate of evolution? g) does the product evolution influence the community evolution (and vice versa)? Answering these questions can lead to knowledge for better designs and more efficient open source processes.

The approach to answer the research question 2 is: to model products as networks with nodes and links, and to characterize the evolution of product structure using node level mechanisms. The node-level mechanisms are analyzed consecutively from open source software projects. The node-level mechanisms captured from open source software projects are applied to model the evolution of the product structures. The evolution of product structure can be characterized since the evolutionary pattern of product structure obtained from simulations match the given case studies.

Existing approaches for studying product evolution involve modeling products as networks, 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 [18]. Such approaches are suitable for comparing the global characteristics of the product structure but are not effective for capturing the underlying dynamics through which products evolve. Our approach is to characterize the evolution of product structure based on node-level mechanisms. Existing studies do not reveal the node-level mechanisms since they only capture the product-level trends and characteristics. To facilitate such analysis at the node level, 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. Generative models that
embody the underlying mechanisms of network growth are important because they can help in understanding the reasons for increasing complexity in software products, and identifying specific ways to maintain the code, to increase modularity, and to reduce product complexity.

The goal in Research Task 2 is to present a generative model for the evolution of open source software. The model is inspired by existing models in the network science literature. 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 modules, files, functions, or classes. The links refer to dependencies between nodes (e.g., class dependencies, function calls). The node-level mechanisms can be captured from case studies of open source projects. By applying the node-level mechanisms in the generative model, the matched characteristics of product evolution are identified. Figure 1.5 displays the answering of research question 2 by approach 2.
The contributions from the Research Task 2 are: 1) an understanding of how node-level mechanisms can determine the product evolution in open source processes, 2) a general approach to capture the node-level mechanisms from open source project consecutively, and 3) a generative model which can simulate the evolution of product structure based on node-level mechanisms. In Chapter 4, the node-level mechanism, which determines the evolution of product structures, is identified. The generative model shows that with different node-level mechanisms, products evolve in different patterns.

1.3.3. Research Question 3, Approach 3, Task 3 and Contributions

Research Question 3 is: How can the co-evolution of product and community be identified in open source processes? The Approach 3 is: Modeling the product and community as hybrid networks with nodes and links, and to identify the co-evolution of product and community by node level, link level and cluster level mechanisms.

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
open source processes because such an understanding helps in determining the impacts of different product structures on community structures and vice versa. Previous studies on the co-evolution between product and community are focused on the theory of socio-technical coordination [43] and the Mirroring Hypothesis [33]. Socio-Technical coordination can be defined as “integrating or linking together different parts of an organization to accomplish a collective sets of tasks” [44] or simply as “management of dependencies” [45, 46]. Coordination between activities can be achieved in various ways such as informal communication, group meetings, and development of plans and rules [45, 46] The mirroring hypothesis is proposed by MacCormack et al [33]. Considering the interdependent nature of the product structures and organizational structures, it is hypothesized that the social structure of the organization matches the structure of the product [33].

Based on the existing studies of co-evolution and coordination between products and communities, our research question is: how do product structures and community structures co-evolve? In existing studies [33, 47, 48], the analysis of co-evolution between product and community are focused on the node level. Researchers study which individuals should communicate with each other based on the commonality in product modules that they work on. This is called expected communication structure. Meanwhile, researchers have proposed metrics to identify how many interactions match with the expected communication structure and how many communication patterns are not matched. In open source processes, information may be shared among individuals even when they do not have direct communication. For example, the posts by individuals in forums can be found by all other individuals even though they do not have direct
communication. Besides, the evolutionary nature of open source processes determines how new participants join over time. The large number of new participants leads to weak measurement power of the proposed matrices. The research approach is focused on the identification of co-evolution between product and community of open source processes at node level, link level and cluster level. At the node level and link level, direct communications are captured in open source processes. At the cluster level, indirect communications, which may happen within the cluster, are also captured. Research question 3 is answered by studying the co-evolution of Open Source Software projects at these three levels.

In Research Task 3, we study the co-evolution processes at three levels namely node level, link level, and cluster level. At the node and link levels, we analyze: a) how the product structures affect the community structure, b) how the community structures affect the product structure. At the cluster level, we study how the communities get increasingly aligned with the anticipated communication patterns. Table 1.5 displays the approaches towards answering Research Question 3 at three levels.

<table>
<thead>
<tr>
<th>Level of Studies</th>
<th>Research Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Level</td>
<td>1) Product Structure $\rightarrow$ Community Structure</td>
</tr>
<tr>
<td></td>
<td>2) Community Structure $\rightarrow$ Product Structure</td>
</tr>
<tr>
<td>Link Level</td>
<td>1) Product Structure $\rightarrow$ Community Structure</td>
</tr>
<tr>
<td></td>
<td>2) Community Structure $\rightarrow$ Product Structure</td>
</tr>
<tr>
<td>Cluster Level</td>
<td>Comparison between actual communities and anticipated</td>
</tr>
<tr>
<td></td>
<td>communities based on product structure.</td>
</tr>
</tbody>
</table>

The contributions of Research Task 3 are: 1) understanding of how co-evolution between product and community emerges at node level, link level and cluster level, and 2) a general approach to analysis the co-evolution at different levels, which can be extended and refined for further studies. In Chapter 5, the co-evolution of product and community
are studied at node-level, link-level, and cluster level. It is observed that the node-level, link level, and cluster level mechanisms determine the co-evolution of product and community in open source processes.

1.3.4. Roadmap among Research Questions, Approaches, Activities, and Contributions

In Table 1.6, the mapping among research questions, approaches, research tasks and contribution is present.

Table 1.6 - Mapping among Research Questions, Approaches, Tasks and Contributions

<table>
<thead>
<tr>
<th>Central Research Question</th>
<th>What kinds of structures of product and community emerge in open source processes over time?</th>
<th>Central Research Approach</th>
<th>The structure and evolution of product and community in open source processes are characterized by using techniques, such as agent-based modeling, complex network analysis, and community detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research Questions</td>
<td>Research Approaches</td>
<td>Research Tasks</td>
<td>Contributions</td>
</tr>
</tbody>
</table>
| **RQ1:** How can the structure of a product be characterized in open source processes? | **Approach 1:** To apply agent-based model, and to characterize the product structure by Degree of Modularity [14] (DOM) and completion time. | **Research Task 1:** An agent-based model to simulate the open source processes. The effect of product structure on open source processes can be captured from the execution of the model | 1) A general approach towards the study of open source processes  
2) Characterization of product structures based on DOM  
3) A computational model that can simulate open source processes |
| **RQ2:** How can the evolution of product structure be characterized in open source processes? | **Approach 2:** To model products as networks with nodes and links, and to characterize the evolution of product structure using node level mechanisms. | **Research Task 2:** A generative model to simulate the product evolution based on node-level mechanisms, which are captured by analysis of case studies. | 1) Characterization of evolution of product based on the node-level mechanisms  
2) A general approach to capture node-level mechanisms from Open Source Projects consecutively.  
3) A generative model to simulate the evolution of product structure based on the node-level mechanisms |
<table>
<thead>
<tr>
<th><strong>RQ3:</strong> How can the co-evolution of product and community be identified in open source processes?</th>
<th><strong>Approach 3:</strong> Modeling the product and community as hybrid networks with nodes and links, and to identify the co-evolution of product and community by node level, link level and cluster level mechanisms</th>
<th><strong>Research Task 3:</strong> Analysis of co-evolution between product and community structure at three levels: node level, link level and cluster level by a case study.</th>
</tr>
</thead>
</table>

**Section 1.4 - Assumptions in the Open Source Processes**

The assumptions we made in this dissertation are idealized characteristics of the open source processes. Open source product development process refers to community led innovation in which the product released under a license that permits inspection, use, modification, and redistribution. Key unique characteristics of open source processes include: 1) in design process, the decentralized decision making in used in open source processes, 2) in product decomposition, the open source processes are based on individual's benefits, costs, and interests, 3) in team structure, the open source processes have flat and community structure, and 4) in the development processes, the open source processes are evolved over time.

The first assumption is that in the design process, decentralized decision making is used in open source processes. In open source processes, the decentralized decision making is widely used in most of the projects. In Drupal project, people can work on the modules which they are interested in. In phpMyAdmin project, the contribution can be divided into seven categories. Participants select the categories which they are most interested in. However, not all of the projects follow the decentralized decision making rule. In Apache Jakarta Project, the project management committee (PMC) holds the right to assign tasks to individuals.
The second assumption is that in the product decomposition, the open source processes are based on individuals' benefits, cost, and interest. In the early stage of open source processes, the product is decomposed based on the developers' benefit and interest. After the decomposition of the product, in the later stage, the product is decomposed based on functions. In our study, we analyze the product structure based on functions because we study the later stage in the open source projects.

The third assumption is that the team structure is flat and community-based in open source processes. The assumption is based on the nature of open source processes. Since individuals work on the tasks based on their interests, the individuals have equal roles. However, with the evolution of the open source team, the hierarchical structure emerges. In the Dural project, the core developers have the right to modify the codes, while other developers submit codes for the review of core developers. In Apache project, the project management committee holds the right to assign tasks and review modifications. In our study, we model the community structure as a complex network, the core developers are represented as hubs in the network.

The forth assumption is that in the development process, the open source processes evolve over time. In the Drupal, phpMyAdmin, and Wordpress projects, the new versions are proposed over time to implement new features and requirements. In Linux and Apache projects, the new versions are still uploaded, but at a slower rate.

Section 1.5 - Overview of the Dissertation

In Chapter 2, background and literature review for this research are presented. The literature review is divided into four components: 1) existing studies of product structure and evolution, 2) existing studies of community structure and evolution, 3) existing
studies of network generating model, and 4) existing studies of co-evolution between product and community. Table 1.7 displays the structure of literature review.

Table 1.7 - Structure of Chapter 2 - Background and Literature Review

<table>
<thead>
<tr>
<th>Existing Studies of Product Structure</th>
<th>Existing Studies of Community Structure</th>
<th>Existing Studies of Network Generating Model</th>
<th>Existing Studies of Co-Evolution between Product and Community</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Structure in Traditional Processes (Section 2.1.1)</td>
<td>Community Structure (Section 2.2.1)</td>
<td>Structure-Based Model (Section 2.3.1)</td>
<td>Interdependence between Product and Community (Section 2.4.1)</td>
</tr>
<tr>
<td>Product Structure and Evolution (Section 2.1.2)</td>
<td>Community Evolution (Section 2.2.2)</td>
<td>Evolution-Based Model (Section 2.3.2)</td>
<td>Mirroring Hypothesis (Section 2.4.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Socio-Technical Congruence (Section 2.4.3)</td>
</tr>
</tbody>
</table>

From Chapter 3 to 5, the corresponding approaches proposed in Research Activity 1 to Research Activity 3 are presented in detail. Table 1.8 displays the framework of the research in this dissertation.

Table 1.8 - Framework of the Research Tasks in the Dissertation

<table>
<thead>
<tr>
<th>Research Question and Approach</th>
<th>Research Tasks</th>
<th>Research Tasks and Results</th>
</tr>
</thead>
</table>
| RQ1 and Approach 1 (Section 1.3.1) | Research Task 1 (Section 1.3.1) | Ch3  
-Introduction (Section 3.1)  
-Background (Section 3.2)  
-Details of Proposed Approach (Section 3.3)  
-Results from Example (Section 3.4)  
-Discussion and Future Work (Section 3.5) |
| RQ2 and Approach 2 (Section 1.3.2) | Research Task 2 (Section 1.3.2) | Ch4  
-Introduction (Section 4.1)  
-Node-Level Mechanisms (Section 4.2)  
-Network Level Analysis (Section 4.3)  
-Generative Model and Results (Section 4.4) |
<table>
<thead>
<tr>
<th>RQ3 and Approach 3 (Section 1.3.3)</th>
<th>Research Task 3 (Section 1.3.3)</th>
<th>Ch5</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Conclusion and Future Work (Section 4.5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Introduction (Section 5.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Proposed Approach (Section 5.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Analysis of the Case Study and Results (Section 5.3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Discussion and Future Work (Section 5.4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Chapter 3, the approach proposed in Research Task 1 towards answering Research Question 1 is presented in detail. In this chapter, agent-based modeling is employed to simulate open source processes. In Section 3.1, an introduction regarding the approach to answering Research Question 1 is presented. In Section 3.2, a background of the agent-based modeling is introduced. In Section 3.3, the details of the model are illustrated. In Section 3.4, a mobile phone example is employed to analyze how different product structures can affect the efficiency of open source processes by using the proposed model. In Section 3.5, the results are discussed and future work for the proposed approach is presented.

In Chapter 4, the proposed approach in Research Task 2, towards answering Research Question 2 is presented in detail. In this chapter, a generative model is proposed to identify whether the node-level mechanisms can determine product evolution in open source processes. In Section 4.1, an introduction is made to explain how the proposed approach can be employed towards answering Research Question 2. In Section 4.2, the node-level mechanisms are captured from case studies of open source processes. In Section 4.3, the product level characteristics are captured from case studies over time. In Section 4.4, the generative model is proposed to simulate the product evolution given the captured node-level mechanisms. In Section 4.5, conclusions and future work are
presented.

In Chapter 5, the proposed approach in Research Task 3, towards answering Research Question 3 is presented in detail. In this chapter, a hybrid network model is proposed to study the co-evolution between product and community. A case study is employed to analyze how co-evolution of products and communities emerges. In Section 5.1, an introduction is presented to illustrate how the proposed approach can be used to answering Research Question 3. In Section 5.2, the proposed approach is discussed in detail. In Section 5.3, an example of Open Source project is used to analyze the interdependence between product structure and community structure at three levels. In Section 5.4, results are documented and future work is presented.

In Chapter 6, we discuss supplemental research for Chapter 5 on the representation of complex bipartite network based on a surrogate model. The motivation of the supplemental research is the difficulties in analyzing complex bipartite networks, which are employed in Chapter 5, due to their complexity and exponential growth. The proposed approach in this chapter provides a surrogate model to represent complex bipartite networks with similar key features. In Section 6.1, the motivation of the supplemental research is presented under the complex network science framework. In Section 6.2, the proposed approach is presented in detail. In Section 6.3, a case study of open source community network, which can be modeled as a complex bipartite network, is employed based on the proposed approach. Comparisons between surrogate model and real network are made.

In Chapter 7, we extend our research to open source hardware (OSH) development processes. We discuss the differences between OSH and open source software
development processes in different aspects. An example, Arduino, is used to illustrate how OSH projects are emerging. The research issues related to OSH are presented. Finally, closing thoughts for this dissertation are presented. Figure 1.6 displays the dissertation overview and roadmap.

Figure 1.6 - Road Map of the Dissertation
2. CHAPTER TWO - BACKGROUND AND LITERATURE REVIEW

This chapter consists of literature review on four different aspects (Section 2.1 to Section 2.4), which serve as basis for the dissertation. In Section 2.1, the existing studies on product structure in open source processes are presented. In Section 2.1.1, we first introduce the existing studies on product structure and Design Structure Matrix (DSM) [17] in traditional product development processes. This section provides a review of the definition of product structure in traditional product development processes and the application of DSM in the studies of product structure for traditional product development processes. In Section 2.1.2, existing studies of product structure and evolution in open source processes, primary related to open source software, are discussed. Existing studies are divided into three groups: 1) study of product structure and evolution in a single open source product, 2) comparison of product structure and evolution between different open source products, and 3) comparison between open source product and proprietary product.

In Section 2.2, we discuss the existing studies of open source communities. In Section 2.2.1, we present the existing studies of the community structure, which are primary based on open source software development projects. In Section 2.2.2, we discuss existing studies of the evolution of community structures, which are primarily based on taking snapshots of product structures over time from open source software projects.

In Section 2.3, the existing studies of network generation models are presented. The network generational models can be divided into two types: 1) structured-based model,
and 2) evolution-based model. In Section 2.3.1, the existing studies of structured-based model are discussed. In Section 2.3.2, we present the existing studies of evolution-based model. We provide a summary of degree-based network models, which serve as the base and inspiration of the models proposed in this dissertation.

In Section 2.4, existing studies on the co-evolution between product structures and community structures are discussed. The studies of co-evolution between product and community can be divided into three parts: 1) studies of inter-relationships between product and community (Conway's Law [31]), 2) Mirroring Hypothesis [33], and 3) socio-technical congruence [47]. In Section 2.4.1, existing studies of inter-relationships between product and community are illustrated. In Section 2.4.2, existing studies of Mirroring Hypothesis are presented. In Section 2.4.3, existing studies of socio-technical congruence are presented.

In Section 2.5, research gaps and limitations regarding existing studies are shown. Table 2.1 displays the structure of literature review, research limitations and research approaches.

<table>
<thead>
<tr>
<th>Literature Review (Section 2.1- Section 2.4)</th>
<th>Existing Studies on Product Structure and Evolution (Section 2.1)</th>
<th>Existing Studies on Community Structure and Evolution (Section 2.2)</th>
<th>Existing Studies on Network Generation Model (Section 2.3)</th>
<th>Existing Studies on Co-Evolution between Product and Community (Section 2.4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Structure and DSM (Section 2.1.1)</td>
<td>Community Structure in open source processes (Section 2.2.1)</td>
<td>Structured-based Model (Section 2.3.1)</td>
<td>Inter-relationships between Product and Community (Section 2.4.1)</td>
<td></td>
</tr>
<tr>
<td>Product Structure an Evolution in open source processes (Section 2.1.2)</td>
<td>Community Evolution in open source processes (Section 2.2.2)</td>
<td>Evolution-based Model (Section 2.3.2)</td>
<td>Mirroring Hypothesis (Section 2.4.2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Socio-Technical Congruence</td>
<td></td>
</tr>
</tbody>
</table>
## Section 2.1 - Existing Studies on Product Structures in Open Source Processes

### 2.1.1. Studies of Product Structure and Design Structure Matrix

Ulrich defines a product structure as a scheme by which the function of a product is allocated to its physical components [38]. A product structure consists of three parts. The *first part* is the arrangement of the functional elements. In the development of a complex engineered system or product, the function can be decomposed into multiple sub-functions so that solutions can be determined for each of the individual sub-functions, and the solutions can be combined into the overall system [16]. There are two advantages of this approach: a) it decomposes the problem into manageable pieces, making it easier to accomplish the overall function, and b) different kinds of sub-functions can be

<table>
<thead>
<tr>
<th>Research Limitations (Section 2.5)</th>
<th>Lack of: 1) Effect of Product Structure on open source processes 2) Reveal How Product Structure Evolve based on Local Observation Results</th>
<th>Lack of: Analysis of Community Structure and Evolution Simultaneously</th>
<th>Lack of: Models Which can Capture Key Features of Complex Networks</th>
<th>Lack of: Evaluation of Co-Evolution at Different Levels for open source processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research Task towards Resolving the Limitations (Ch3-6)</td>
<td>1) Characterization of Product Structure on open source processes (Ch3) 2) Study of Local Level Mechanisms for Product Evolution in open source processes (Ch4)</td>
<td>Studies of Open Source Community Structure and Evolution (Huang, Le, and Panchal [35])</td>
<td>Building Surrogate Model to Represent Complex Network with Key Feature (Ch6)</td>
<td>Studies of Co-Evolution between Product Structure and Community Structure at Three levels: 1) node level, 2) link level, 3) cluster level (Ch5)</td>
</tr>
</tbody>
</table>
achieved concurrently, which reduces the product development time.

The second part of product structure is the mapping from functional elements to physical components. The physical elements of a product are the parts, components, and subassemblies that ultimately implement the product’s functions [49]. The mapping from functional elements to physical components results in the physical design of the product. Based on the mapping, each component is well defined and the function of that component is clearly focused. Considering the different kinds of mappings, two major classes of product structures are proposed by Ulrich [38]: integral structure and modular structure. In an integral structure, the components of the product are highly coupled with each other and cannot be decomposed easily. There is a lack of clear set of mappings from functions to physical realizations. Any change in a component results in significant effects on other components. Hence, a purely integral structure is generally not preferable because it is difficult to produce and maintain. A physical component can represent multiple functions in an integral structure. In contrast, a modular structure has one-to-one mapping between the functions and components.

The third part of product structure is the specification of the interfaces among interacting physical embodiments of the different functions. In a modular structure, interfaces describe in detail how the modules interact, including how they fit together, connect and communicate. In an integral structure, a physical component may carry out more than one function, which is referred to as function sharing [50]. In such cases, the physical embodiments of the two functions are highly coupled.

Ulrich [38] divides modular structures into three sub-types: slot structure, bus structure and sectional structure. Slot structure represents a one-to-one mapping from
functions to physical realizations. In a slot structure, the interfaces between components are different from each other, and hence, the components cannot be interchanged [38]. Bus structure has a common component called a bus to which the other physical components connect via the same types of interfaces [38]. Most of the common products’ structures lie between slot and bus structures. In certain cases, a few components may have many more interfaces than other components, and hence, partially act as a bus in a product.

A powerful mathematical tool, called the Design Structure Matrix (DSM), was developed by Steward [15] to analyze the interfaces and the relationships between modules. Steward originally used DSM to compactly display the structure of design problems in terms of relationships among technical parameters [51]. Table 2.2A shows an example of a Design Structure Matrix. In this table, there are five different parameters labeled as A, B, C, D, and E. The dependencies between these parameters are shown with the ‘x’ elements in the matrix. There are two conventions for representing dependencies in a design structure matrix. The ‘x’ in the row corresponding to row B, column A can represent a dependency of parameter B on parameter A, or a dependency of parameter A on parameter B. In this dissertation, we choose the latter convention for representing dependencies. Similar matrices can be used to represent dependencies between different modules in a product.

Table 2.2 - Example of a Design Structure Matrix
Pimmler and Eppinger [16] developed a score-based DSM to describe the spatial, materials, energy and information dependences among modules. In their definition, the number 2 is used to represent *required* relationships among different modules. These are relationships that cannot be removed from the structure. The number 1 is used to describe *desired* relationships among different modules, which means that if the relationship exits, it will benefit the entire product. However, if the relationship does not exist, the product can still work. The number 0 is used to represent *independent* modules, which do not have any relationship - spatial, material, energy or information. The number -1 is used to express *undesired* relationships among modules that have negative impact on the product. The number -2 is used to show relationships that must be prevented in the product. Table 2.2B shows an example of a score-based DSM. The four numbers in each block refer to the strength of spatial (top left), energy (top right), information (bottom left), and materials (bottom right) dependencies.

Smith and Eppinger [51] present a numerical DSM with percentage rework. It is a quantitative description of interdependencies between tasks in a real design and manufacturing process. The numerical DSM provides a convenient way to perform numerical analysis. In this dissertation, a numerical DSM is used to represent the product structure in the form of modules and interdependencies. When a module changes, other related components need rework with a probability represented by percentage rework.
2.1.2. Studies of Product Structure and Evolution in open source processes

Existing studies of product structure and evolution in open source processes are focused on analysis of Open Source Software project due to: 1) the availability of documentation and codes, and 2) the success and highly development of Open Source Software projects.

Crowston and co-authors [52] recently published a survey article highlighting the diverse research efforts on open source software development. 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 dissertation fits into their outputs (evolution) category. Some of the earliest models of software evolution were in the form of differential equations [53] built using general principles such as the Lehman’s laws [54].

The software structure is modeled by the technical dependencies which can be identified using two approaches [47]. 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 [47], functional dependencies [55, 56], or syntactic dependencies [57]. Examples include dependencies between statements [58], global variables [59], or functional calls [60, 61]. The second approach involves identifying dependencies by examining how modification requests affect a set of source code files [47, 62]. Examples include analysis of code decays based on modification requests [63], and analysis of modifications involving files that tend to
change together [34].

The literature on the analysis of the product structure and evolution in open source software is summarized in Table 2.3. Based on the goals, the literature can be categorized into three groups. The first group of studies focuses on the evolution of software structure in a single product. The studies in this group are primarily focused on the modularity and complexity of software structure, such as determining how the size increases [21, 22], modularity changes [64], and complexity increases [23, 65] with software evolution. MacCormack et al. [42] and LaMantia et al. [19] 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 [66], propagation cost [18], and clustered cost [18]. Modularity is an important characteristic because it affects evolvability, changeability, maintainability [67], and customizability. Modularity is measured in terms of complex network metrics such as path length and clustering coefficient [68].

Table 2.3 - Literature on the Analysis of the Structure and Evolution of Open Source Software

<table>
<thead>
<tr>
<th>Authors</th>
<th>Goal</th>
<th>OSS Products Analyzed</th>
<th>Level of Analysis</th>
<th>Quantitative Network Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mockus et al. [69, 70]</td>
<td>Comparing the development process of Apache, Mozilla, and commercial projects</td>
<td>Apache, Mozilla</td>
<td>Files</td>
<td>cumulative distribution of nodes</td>
</tr>
<tr>
<td>Russovan et al. [61]</td>
<td>Investigating the quality of open source software</td>
<td>Linux ARP Module</td>
<td>Functions and Files</td>
<td>None</td>
</tr>
<tr>
<td>Sangal et al. [71]</td>
<td>Management of the structure of large software</td>
<td>Haystack</td>
<td>Files</td>
<td>None</td>
</tr>
<tr>
<td>Huynh and</td>
<td>Checking the source code modular</td>
<td>KWIC system</td>
<td>Files</td>
<td>None</td>
</tr>
<tr>
<td>Authors</td>
<td>Research Focus</td>
<td>Software</td>
<td>Files Types</td>
<td>Measured Metrics</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Cai [67]</td>
<td>Structure to design modular structure</td>
<td></td>
<td>(Modules)</td>
<td></td>
</tr>
<tr>
<td>MacCormack et al. [18]</td>
<td>Comparison of modularity in open-source and proprietary software</td>
<td>Mozilla, Linux</td>
<td>Files</td>
<td>propagation cost, clustering cost</td>
</tr>
<tr>
<td>MacCormack et al. [18]</td>
<td>Impact of modularity on evolution</td>
<td>Linux</td>
<td>Files</td>
<td>direct connectivity in/out, visibility fan-in/out, survival rate,</td>
</tr>
<tr>
<td>Milev et al. [64]</td>
<td>Evolution of modularity in an open-source software</td>
<td>Apache Tomcat</td>
<td>Classes</td>
<td>propagation cost, clustered cost</td>
</tr>
<tr>
<td>LaMantia et al. [19]</td>
<td>Impact of modularization on design evolution</td>
<td>Tomcat</td>
<td>Classes</td>
<td>change ratio</td>
</tr>
<tr>
<td>Valverde and Sole [20]</td>
<td>Universal patterns in software organization</td>
<td>80 OSS projects including Muds, JDK-B, and JDK-A</td>
<td>Classes</td>
<td>clustering coefficient, degree distribution, average path length, average degree</td>
</tr>
<tr>
<td>Darcy et al. [72-74]</td>
<td>Evolution of size and structural complexity in software</td>
<td>105 OSS projects from Sourceforge</td>
<td>Modules</td>
<td>Coupling and Cohesion, CplXLcoh, LOC</td>
</tr>
<tr>
<td>Terceiro and Chavez [65]</td>
<td>Structural complexity evolution</td>
<td>Ristretto project</td>
<td>Modules</td>
<td>CplXLcoh, LOC</td>
</tr>
<tr>
<td>Sosa et al. [23]</td>
<td>Exploration of the evolution of a software product structure</td>
<td>Ant</td>
<td>Files</td>
<td>Internal and cross-module complexity, propagation cost</td>
</tr>
<tr>
<td>Godfrey and Tu [21, 22]</td>
<td>Exploring the evolution of Linux</td>
<td>Linux kernel</td>
<td>Files</td>
<td>size in bytes, LOC, number of files</td>
</tr>
<tr>
<td>Wen et al. [68]</td>
<td>Tracking the evolution of a software system</td>
<td>Apache</td>
<td>Functions</td>
<td>degree distribution, propagation cost, path length, clustering coefficient</td>
</tr>
<tr>
<td>Myers et al. [75]</td>
<td>Network-based analysis of open software</td>
<td>VTK, DM, Abiword, Linux, MySQL, XMMS</td>
<td>Classes and sub-routines</td>
<td>Degree distribution, degree correlation, clustering, weak connected components, strong connected</td>
</tr>
</tbody>
</table>
The second group of studies is focused on comparing the structures of different open source software to identify differences and common patterns across different projects. Mockus et al. [69, 77] compare the development process of Apache and Mozilla. MacCormack et al. [18] 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 [20] 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 [20] mainly focus on discovering topologies and characteristics of product structures based on complex network analysis. Darcy et al. [72-74] analyze the evolution of size and structural complexity in 105 open source projects from SourceForge. The authors propose a metric called CplXLCoH to quantify software complexity. Myers et al. [75] use network analysis metrics including degree distribution, degree correlation, and clustering to analyze different open source software products.

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 [78] and O'Reilly [79] claim that open-
source software is more "modular" than proprietary software. On the other hand, Torvalds [80] suggests that modularity is a required property for the success of OSS development. MacCormack et al. [18] 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). LaMantia et al. [19] accounts for changes in consecutive versions of open source software using a metric called change ratio, which is defined as the sum of new entities added and existing entities removed divided by the total number of entities. The authors focus on discovering the dynamics of substitution of entities. However, the dynamics of creation and removal of interfaces between entities is not captured. 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.

**Section 2.2 - Existing Studies on Community Structures in Open Source Processes**

Existing research on structure and evolution of communities in Open Source Process mainly focuses in the area of Open Source Software (OSS) development because the
open source processes are still in early stage. Hardware and physical product development processes have emerged recently like OpenCPU [81], Reprep 3D printers [82], but there are few studies focusing on their communities. Hence, the way to study structure and evolution of community in open source processes is to analyze the Open Source Software communities at this stage. The Open Source Software is a public good provided by volunteers – the “source code” used to generate the programs is freely available to read, use and modify [83]. The OSS development project is typically initiated by an individual or a small group with ideas which can realize their intellectual, personal or business interest [84]. Since the OSS code is open to everyone, people can join in and make contributions including coding, testing, reporting bugs, etc. The defects are found and fixed very quickly because there are “many eyeballs” looking for the problems [78]. Code is written with more care and creativity, because developers are working only on things for which they have a real passion [78]. Based on the features of OSS development processes, the OSS development processes are selected to be presented as examples of Open Source process. Hence the structure and evolution of OSS development communities are studied to get deeper understanding for organization of open source processes.

2.2.1. Studies of Structure of Open Source Community

Various researchers have presented empirical and quantitative studies on the structure of OSS communities based on the data from existing OSS projects. Cox [24] presents his initial thoughts of “town councils” structure in OSS community based on Linux 8086 project. In his paper, he illustrates a possible community structures for Linux 8086 project, which is an OSS development project without data analyzing. Weber [25]
discusses different types of organization structures have been showed in different OSS projects. For example, the community structure of Linux project displays as a pyramid, and the community structures of BSD project are represented as concentric circles, etc. The structures that Weber concludes are based on direct observation from the network flowchart without mathematical analyzing. Crowston and Howison [85] discuss the community centralization in OSS development communities by analyzing data from bug tracking system in Sourceforge and demonstrate that the community centralization or decentralization is not a characteristic of OSS projects. Crowston and Howison [86] later analyze hierarchy and centralization of the OSS community by employing social network analysis (SNA) metrics in the case studies of Apache, Savannah and Sourceforge and conclude a tendency that large projects are less centralized and hierarchical. Xu and Madey [27] represent role distribution and degree distribution in Sourceforge community, one of OSS development communities. Xu, Christley, and Madey [30] and Gao and Madey [87] studies topological properties including degree distribution, diameter, clustering coefficient, centrality and component distribution and in the OSS communities by viewing the OSS communities as complex social networks. They also find small world [88] and scale – free [89] properties in OSS communities in the case study of Sourceforge. Xu et al [29] studies the structure of OSS communities by calculating the modularity of the project network, which is defined as the fraction of edges within communities subtracts the expected value of the same quantity if edges fall in a random network and analyzing groups exist in the Sourceforge project network.

2.2.2. Studies of Evolution of Open Source Community

White et al [90] introduced the modeling of social structure over time using snapshot
data, which is a common method to analyze the evolution of OSS development communities. Nakakoji et al [76] discuss the evolution of communities in the aspect of role changes of the members in OSS communities and conclude two factors which determine the evolution of OSS communities, namely the existence of motivated members and the social mechanism of communities. Weiss et al [91] trace the evolution of a community by taking snapshots of its membership at regular intervals and establish a major hypothesis that OSS communities grow by a process of preferential attachment [92]. De Souza et al [93] represent a framework for software modules and software developers and study software project communications at two points of time. By doing this, they can analyze movements of developers in different modules of software systems. Howison et al [94] investigate the structure of OSS development communities over time using snapshot data to understand the dynamics of social structures in OSS development communities. They examine three properties of the social structures which are centralization, network center and stability of participation. Wiggins et al [95] analyze the dynamics of OSS development communities and find variation of communication centralization and the decentralization trends for the OSS development communities.

Section 2.3 - Existing Studies on the Network Generation Model

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.
2.3.1. Structured-Based Models

The exponential random graph model, also referred to as the P* model, is a widely used structure-based model [96-98]. In the 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:

\[
\Pr(Y = y) = \left(\frac{1}{k}\right) \exp \left\{ \sum_{A} \eta_{A} g_{A}(y) \right\} [98]
\]

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, \( g_{A}(y) \) is the network statistic corresponding to the configurations \( A \). Based on the observed network, the parameters \( \eta_{A} \) are calculated using methods such as pseudo-likelihood estimation [99] and Markov Chain Monte Carlo (MCMC) maximum likelihood estimation [100]. The P* model is built by formulating assumptions about substructure configurations and then validating these assumptions using the resulting parameters \( \eta_{A} \). Simple substructure configurations include reciprocity, two-star, three-star, and triangle. Current development in the P* model focuses on discovering new structural specifications, including k-star structure, k-triangle structure, and alternative k-two-paths [101-103]. While the P* models are focused on local substructures within the network, other structural models focus on replicating specific global characteristics of the network. For example, the goal of structure-based Internet topology generators [104-106] is to explicitly model its hierarchical characteristics.
2.3.2. 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 [107]. 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 [108]. 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. A summary of various degree-based network evolution models is provided in Table 2.4

Table 2.4 - Summary of Degree-based Network Models

<table>
<thead>
<tr>
<th>Authors</th>
<th>Degree-based Model</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preferential Attachment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barabási and Albert [108]</td>
<td>Scale Free Network (BA-model): π(i) ∼ k_i</td>
<td>Power-law distribution with γ=3</td>
</tr>
<tr>
<td>Dorogovtsev and Mendes [109]</td>
<td>Initial Attractiveness: π(i) = ∼ (am + k_i) where m is new edges at each step</td>
<td>Power-law distribution with γ=2+a</td>
</tr>
<tr>
<td>Krapivsky and Redner [110]</td>
<td>Nonlinear Attachment: π(i) = ∼ k_i^a</td>
<td>0&lt;α&lt;1: ( P(k) = \frac{\mu}{k^\alpha} \prod_{j=1}^{k} \left( 1 + \frac{\mu}{j^\alpha} \right)^{-1} ) &lt;br&gt; ( \alpha&gt;1, &quot;winner takes all&quot; )</td>
</tr>
<tr>
<td>Accelerating Growth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dorogovtsev and Mendes [109]</td>
<td>C_{dt} new links are added based on the preferential attachment.</td>
<td>Power-law distribution with γ = 1 + \frac{1}{1+\theta}</td>
</tr>
<tr>
<td>Barabasi et al.</td>
<td>new links are equal to K &lt;&lt; K_c, γ=1.5.</td>
<td></td>
</tr>
</tbody>
</table>
and Dorogovtsev and Mendes [92, 109] at+2b at each step K >> Kc, γ=3. Kc: critical degree fast decaying tail of degree distribution

<table>
<thead>
<tr>
<th>Local Events</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dorogovtsev and Mendes [109] internal edges rewiring: when k is small, the probability P(k) saturates at a fix value</td>
<td>Dorogovtsev and Mendes [109] internal edges removal Power-law distribution with γ = 2 + ( \frac{1}{1+2c} ) c is the no. of edge removal.</td>
<td></td>
</tr>
</tbody>
</table>

| Other Evolutionary Conditions | | |
| --- | --- | |
| Amaral et al. [111] Aging or cost the degree distribution will have a cutoff for large k. | Bianconi and Barabási [112] Fitness model: \( \pi(i) = \frac{\eta_i k_i}{\sum_j \eta_j k_j} \) Depend on distribution of η | Dorogovtsev et al. [109] Edge inheritance Depend on density h(c) |

Section 2.4 - Existing Studies on Interdependencies between Product Structure and Community Structure

Interdependencies between products and organizational structures have been studied both in software products such as Linux and Apache [2, 18], and physical products such as aircraft engines [17] and air conditioners [115]. 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.

A summary of the key studies focused on the interrelationship between products and organizational structures is provided in Table 2.5

<table>
<thead>
<tr>
<th>Authors</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.</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 Edges: Function calls</td>
<td>Nodes: Individuals Links: Individuals are linked if they work on the same project</td>
</tr>
<tr>
<td>MacCormack</td>
<td>Comparison of</td>
<td>Both open</td>
<td>Nodes:</td>
<td>Organizational/community</td>
</tr>
</tbody>
</table>
### 2.4.1. Studies of Inter-relationships between Product and Organization

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. [33] 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

<table>
<thead>
<tr>
<th>Authors</th>
<th>Focus</th>
<th>Methods</th>
<th>Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>et al. [18, 33]</td>
<td>the structures of products developed using open source and closed source approaches</td>
<td>and closed source</td>
<td>Functions Edges: Function calls</td>
</tr>
<tr>
<td>Colfer and Baldwin [116]</td>
<td>Review of studies on validating the mirroring hypothesis</td>
<td>Both open and closed source</td>
<td>Different approaches are discussed</td>
</tr>
<tr>
<td>Cataldo et al. [34, 47]</td>
<td>Evaluation of congruence between technical dependencies and work dependencies</td>
<td>Closed source</td>
<td>Nodes: Source files Edges: Sets of files modified together</td>
</tr>
<tr>
<td>Sosa [48]</td>
<td>Comparison of potential interactions with actual communications</td>
<td>Closed source</td>
<td>Nodes: Product modules Edges: Dependencies identified by system architects</td>
</tr>
</tbody>
</table>
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. [32] 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.

2.4.2. 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 [116]. 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 [116] 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 Cabigiosu [115]
analyze inter-organizational relationships between the manufacturers and suppliers. The analysis supports the mirroring hypothesis at the component level but only when the product structure is stable. Camuffo and Cabigiosu [115] and Hoetker [117] analyze the extent to which modular products are associated with modular organizations. Using an example of the air conditioning industry, Kratzer et al. [118] 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.

2.4.3. Socio-Technical Congruence

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 [43] 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. [34, 47] 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 [47] present a quantitative measure of congruence based on the comparison of matrices representing coordination requirements and actual
coordination. Sosa [48] 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. [47] use resolution time as a measure of performance and show that congruence results in lower resolution times (i.e., better performance). Kwan et al. [119] 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. [120] 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.

Section 2.5 - Research Gaps

The research limitations exist in the studies of product structure, community structure, network generation model, and co-evolution between product and community structures.

In the studies of product structures and DSM (Section 2.1.1), the definition of product structure is given and two major types of product structures are also proposed by Urich. The mathematical tool – DSM is proposed by Pimmler et al. Several applications of analysis are based on DSM. However, the studies of product structure are based on traditional product development processes instead of open source processes. The effect of product structure on the open source product development processes is still unclear.

In the studies of product structure and evolution in open source processes (Section
2.1.2), various case studies have been performed to analyze the evolution of open source 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). The dynamics of creation and removal of interfaces between entities is not captured. 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.

In the studies of community structures and evolution (Section 2.2), Madey et al. discuss the structures of open source software development communities – Sourceforge. Nakakoji et al. discuss the evolutionary properties of open source software development communities. However, there is a lack of analysis of the open source community structures and evolution simultaneously. Besides, researchers in this area always utilize the social network metrics to analyze the structure of communities. However, other methods, like the clustering technology can also be employed to analyze community structures and get intuitive results.

In the studies of network generation models (Section 2.3), three models with different key topologies: random networks, small world networks and scale-free networks are proposed by researchers. However, the key topologies only describe a topology feature (degree distribution) of the networks without well matching network parameters in specific applications.
In the studies of co-evolution between product structure and community structure (Section 2.4), 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. Besides, 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 reduces.

The research tasks proposed in Chapter 3 to 6 are employed to address these research limitations and answer the research questions proposed in Section 1. In Chapter 3, the product structure is characterized by DOM and completion time. The effect of product structure on the completion time is studied by the agent-based modeling. The proposed approach in Chapter 3 answers the RQ1. Besides, the proposed approach in Chapter 3 also address research limitation in studying the effect of product structure on product development processes in open source processes. In Chapter 4, the node-level mechanisms of product evolution are analyzed from open source software projects. A generative model is proposed to simulate the evolution of product structure based on the node-level mechanisms. The proposed approach in Chapter 4 answers the RQ2. Besides, the proposed approach in Chapter 4 address the limitation in the existing studies of product structure and evolution in open source processes. In Chapter 5, the co-evolution between product structure and community structure is analyzed in three levels: node level, link level and cluster level. The proposed approach answers the RQ3. Besides, the proposed approach in Chapter 5 address the limitation in the existing studies of co-evolution between product and community in the open source processes. Followed by Chapter 5, in Chapter 6, we propose a surrogate model to represent the complex bipartite network since the bipartite network we study in Chapter 5 can be complex and need high computational effort. The supplemental research in Chapter 6 proposes an approach to build a surrogate model to represent the complex bipartite network with matched key characteristics. The proposed approach in Chapter 6 addresses the limitations for the
existing studies of network generation model.
3. CHAPTER THREE - ANALYSIS AND MODELING EFFECT OF PRODUCT STRUCTURE ON OPEN SOURCE PROCESSES

Section 3.1 - Motivation: Answering RQ1

In this chapter, the objective is to answer the RQ1 through Approach 1. The Research Question 1 is: How can the structure of a product be characterized in open source processes?

The motivation of the RQ1 is that although product structure is well studied and characterized on the traditional product development processes, the product structure of open source processes has not been well characterized yet. To address this research gap, the studies to analyze how the product structure affects the open source processes are proposed in this chapter. The proposed approach is based on agent-based computational modeling [41]. In the agent-based model, we simulate the behavior of participants as they contribute to product development efforts, and the resulting product completion time. By simulating the open source processes, the product completion time for different product structures with different Degrees of Modularity (DOM) in open source processes is studied in this chapter. The comparison among product completion time for different product structures in the model can be used to reveal the effect of product structures with different DOM on completion time.

In this chapter, an agent-based model is proposed to represent the general open source processes because: a) the product structure is represented with nodes and links, where the nodes represent modules (or sub-systems), the links represent the interfaces.
The product structures can be modified in the agent-based model to represent any products, which are developed using open source processes. b) The core of the agent based model is that individual's actions can affect the product development processes, which is the base of open source processes. Once the agent based model is built, the open source process can be simulated by execution of the agent based model. By having different product structures in the agent based model, the different product completion time can be revealed. Hence, the computational results from agent based model can be used to analyze how different product structures can affect the product development processes in open source processes. Hence, the RQ1 can be answered by execution of the proposed model. From existing studies of product structures, the Degree of Modularity [14] (DOM) is employed as a quantitative metric to distinguish different product structures. By calculating the DOMs associated with different product structures, the product structure can be characterized by DOM and completion time. Figure 3.1 displays the how the proposed approach answers the RQ1.

Figure 3.1 - How the Proposed Approach Answers RQ1 by Approach 1

The outline of the chapter is as follows. In Section 3.2, we provide a background of
agent-based modeling. In Section 3.3, the details of the agent-based model are presented. In Section 3.4, an example of mobile phone design is presented and different possible architectures of mobile phones are discussed. In Section 3.5, the results from the execution of the model are discussed. Finally, the closing thoughts are presented in Section 3.6.

**Section 3.2 - Background: Agent-based Modeling (ABM)**

Agent-based modeling (ABM) is a technique used to simulate systems consisting of autonomous interacting entities called agents [41, 121]. It is a computational approach for simulating the actions and interactions of autonomous individuals, with the goal of assessing their effects on the entire system. As discussed by Bonabeau [41], an agent is a discrete entity or individual with behavior rules. In ABM, the system is modeled as an aggregation of independent units interacting with each other. The interactions of units result in diverse, sometimes unexpected and interesting emergent phenomena.

Bonabeau [41] presents several benefits of the ABM technique. First, ABM facilitates modeling emergent phenomena. Because of the interactions among agents and unexpected behaviors of agents, on executing an agent-based model, some emergent phenomena may be observed that are generally difficult to predict. Second, ABM provides a natural description of many real-world systems such as social systems. In social sciences, ABM has been used in a variety of applications such as flow management [41], traffic simulation [122], organizational modeling and diffusion [41]. ABM is becoming a popular technique in computational economics and is applied to supply chains [123] and changes of tickets’ price in NASDAQ. Agent-based modeling has recently been used in engineering design also. Sim et al. [124] model integrated
product development teams using agent-based modeling. They divide product development tasks into sub-tasks and each agent is assigned to one sub-task. Using the model, they show the relationships among number, competency and motivation of agents and the cost, time, quality of products. Crowder et al. [125] use agent-based models to investigate how agents in integrated product development teams interact and accomplish tasks in an ongoing project. Nissen and Levitt [126] use agent-based models to simulate the dynamic knowledge management in the information system design phase.

Various software tools such as Netlogo [127], Swarm [128] and AnyLogic [129] are available for developing ABMs. Due to its user friendliness, we use Netlogo for developing the model presented in this dissertation. The details of the agent-based model of open source product development are presented next.

Section 3.3 - Details of the Agent-Based Model of open source processes

In this section, we present the agent-based model used to simulate the evolution of products in open source processes. The general agent-based model for Open Source Process was first developed by Panchal [10]. In this research task, the emphasis is on the study of the product structures in open source processes using the agent-based model, rather than the model itself. The model consists of two main entities: a) products, which are composed of modules, and b) participants, who contribute to the development of the product. The product model is discussed in Section 3.3.1, and the model of the participants is discussed in Section 3.3.2. The outputs and the assumptions of the model are discussed in Section 3.3.3.
3.3.1. Product Model

In the proposed model, a product is represented as a directed weighted graph consisting of a set of modules as nodes with interdependencies between them. A module essentially represents a physical embodiment of a function. Hence, for an integral structure, two modules may correspond to a single physical component. Each module is associated with two attributes: a) percentage completion, and b) growth rate. The percentage completion captures the extent of completion of a module at a given point in time. A module is complete when the percentage completion reaches 100%. The growth rate represents how fast the module grows. It is used to account for the fact that different modules may vary in the amount of effort and time required for development. A high growth rate represents a module that can be developed faster than a module with a low growth rate. The growth rate is expressed as a percentage value. For example, a module with a percentage completion of 10% and a growth rate of 5% will reach a percentage completion of 10.5% in the following time step and a percentage completion of 11.025% after two time steps.

The links between the modules represent the interdependencies between modules. They are also called interfaces because interfaces are linkages shared among components, modules, or sub-systems of a given product [130]. Each link is associated with an attribute, percentage rework [131], which represents the amount of rework that must be carried out on a target module if there are changes in the originating module. An example of a product model of mobile phones is shown in Figure 3.2. The model is not intended to represent a particular mobile phone available in the market. It represents some of the components commonly available in cell phones and is used as an illustrative example.
The product model is asymmetric because of the differences among different components. As an example from a mobile phone, the modification of the antenna results in a minor change in the upper case because it only requires a minor geometric change in the upper case. Hence, the dependence of the antenna on the upper case is considered weak. But the strength of dependence is different when the opposite direction is considered. Since the antenna is attached to the upper case, a change in the upper case may result in a significant modification to the antenna. Hence, the dependence from the uppercase to the antenna is stronger.

![Diagram of a mobile phone product model](image)

**Figure 3.2 - An example of a product model for a mobile phone**

The percentage rework is modeled after the work of Cho and Eppinger [131, 132]. A simple example is showed in Figure 3.3. In this figure, a link originates from Module $A$ and terminates in Module $B$, which indicates the dependency of Module $B$ on Module $A$. When Module $A$ changes, the amount of rework on Module $B$ is equal to the extent of completion of Module $B$ multiplied the percentage rework from $A$ to $B$. If the percentage completion of Module $B$ is 30% and the percentage rework of the link is 1.5%, then any
change in Module A will result in a rework of 0.45 (=30x1.5/100). Hence, the resulting percentage completion of Module B will be 29.55% (=30-0.45).

![Module A: % Completion: 50% % Rework: 1.5 Module B: % Completion: 30%]

**Figure 3.3 - Illustration of rework**

The modules and dependencies describe a product’s structure. The modularity of a product’s structure is characterized by a metric called the Degree of Modularity (DOM) [14]. The metric facilitates the comparison of different structures of a product. Sosa et al. [14] utilize the DOM metric to compare different modules based on their connections with other modules. The authors define the modularity of a single module using a metric, *Module Modularity*, which is defined as “the level of independence of a module from the other modules within a product” [14]. A module is recognized as a highly independent one if it has a high Module Modularity. Mathematically, Module Modularity [51] is represented as:

\[
M_i = \frac{(m - 1)x_{\text{max}} - x_{i+}}{(m - 1)x_{\text{max}}}; \quad x_{i+} = \sum_{j=1, j \neq i}^{m} x_{ji}
\] (3.1)

In the relationship above, \(x_{ji}\) is the dependency represented by the \(j^{th}\) row and the \(i^{th}\) column in the DSM, \(x_{\text{max}}\) is the maximum value that \(x_{ji}\) can take, and \(m\) is the number of modules. Figure 3.4 shows an example of Module Modularity of Module A for a product with four modules. A change in Module B results in a percentage rework of 0.5 in Module A. A change in Module C results in a percentage rework of 0.3 in Module A. Module D does not affect Module A. The DSM of the product is also shown in Figure 3.4. If the maximum value of \(x_{ij}\) (i.e., \(x_{\text{max}}\)) is 1, and \(x_{i+}\) is the sum of \(x_{ij}\) from \(x_{11}\) to \(x_{41}\) (which is 0.8 in this case), then the Module Modularity calculated using Equation 3.1 is 0.73.
The DOM of a product is the average level of modularity of modules. So in this dissertation, the DOM is based on the average of the Module Modularity. The mathematic expression of DOM:

$$M(ID)_{average} = \frac{\sum_{i=1}^{m} M(ID)_i}{m}$$

(3.2)

Other measures based on DSM have also been utilized in the literature to determine the DOM. One such measure is the Singular-value Modularity Index (SMI) [133, 134]. By performing singular values decomposition (SVD) on the DSM, singular values are captured for the analysis of DOM. The SMI index measures the average, weighted decay rate of sorted singular values in the system [134]. Another modularity measure is proposed by Guo and Gershenson [135] In this chapter we use the Degree of Modularity (DOM) measure because of its simplicity.

### 3.3.2. Participants

The participants in the agent-based model are modeled as decision-making agents. The number of agents in the model is represented as $N$. Each agent makes decisions about whether to contribute to the product development effort or not. The decisions are made based on participants’ individual cost ($C$) and value ($V$). The cost is due to the time and effort invested in contributing to the product development effort. The value corresponds to the benefit that the participant gains from the product.
The decisions made by different participants can be modeled using the game of involuntary altruism presented by Baldwin and Clark [136]. This game is an adaptation of the game of “public provision of private goods” [137] and has been applied to mass collaborative processes such as open-source [138]. Each participant has value \( V \) and cost \( C \) associated with an activity. They independently decide whether to contribute to the effort or not. If the players decide not to contribute, the task is not completed and the value to all of them is 0. If one of the participants decides to contribute and the others do not, then the non-contributing participant receives the entire value \( V \) and does not incur any cost. Since the contributing participant also incurs the cost \( C \), the resulting value for the contributing participant is \( V - C \). The mixed equilibrium strategy of the game is that each participant contributes with a probability [136]:

\[
a_s = 1 - \left( \frac{C}{V} \right)^{\frac{1}{N-1}}
\]  

(3.3)

In Equation 3.3, if the cost \( C \) of a participant is large, the probability of participation is small. If a participant’s value \( V \) is large, the probability of participation of the agent is high. The probability of participation is also dependent on the number of participants. In the equation, cost and value parameters are quantified as dimensionless numbers. The cost is a simplified parameter that accounts for the monetary costs and time expended. Similarly, value is a simplified parameter that accounts for both financial and non-financial benefits associated with the activity. Research is being carried out in other domains to identify the value associated with mass collaborative activities [139]. One potential approach to map actual costs and benefits to dimensionless numbers is using multi-attribute utility functions [140].

The model is executed in cycles. During each cycle, each participant decides
whether to participate in the effort or not. The decision is made based on the probability calculated using Equation 3.3. If the participant decides in favor of contribution, he/she works on the module with the lowest percentage completion. The choice of the module with the lowest percentage completion is chosen for simplicity. In a real open-source scenario, the choice of the module to contribute to depends on factors such as participants' expertise, prior work on different modules, and individual needs. If a participant is working on a module at a given time step, other participants cannot change that module. That means the module is “locked” by the participant for that particular time step. In that case, the next participant works on the next least-complete module. If all the modules are locked, the next participant does not contribute during that time step. During each contribution to a module, the effect on the other modules is calculated based on the module dependencies. Rework percentage is used to calculate the resulting percentage completion of the affected modules.

### 3.3.3. Outputs of the Model

The outputs of the model are 1) the evolution of each module over time and 2) the evolution of the entire product over time. Note that in this model, time is measured in terms of the number of cycles. Hence, the unit of time is a cycle of model execution. The evolution of each module is observed during the execution of the model. An example of evolution of modules is shown in Figure 3.5. It is assumed that all modules are equally important and require equal amounts of effort. Hence, the percentage completion of the entire product is evaluated as the average percentage completion of the modules in the product. An example of the evolution of the entire product is shown in Figure 3.6. This is again a simplification in the model because in a real scenario, different modules require
different amounts of effort for completion. The product evolution plot provides two main indicators: a) the evolution time, which represents how fast the product development is completed, and b) the slope of curve, which represents the rate of product evolution at a particular point in time.

![Evolution of modules](image1)

**Figure 3.5 - Example of the evolution of different modules**

![Evolution of the entire product](image2)

**Figure 3.6 - Example of evolution of the entire product over time**

### 3.3.4. Assumptions in the Model

The basic assumptions used in the model are as follows:

*Growth rate:* The growth rate is expressed as a percentage of current progress. In
terms of the growth of a module, this means that the growth of a module is slower during the initial stages and faster during the latter stages. (see Figure 3.6) For instance, the initial conceptual design phase requires more time than the latter detailed design and manufacturing phases. The percentage growth rate for each module is considered equal and remains constant during the development process. This is a simplifying assumption which implies that all modules require equal amount of effort to develop. However in a real project, different modules have different levels of design complexity and generally require different levels of effort. Hence, the growth rates of different modules are different.

**Locking:** It is assumed that during a cycle, when a participant decides to work on a module, that module cannot be changed by other participants. The advantage of this is that when a module is locked, multiple participants cannot make conflicting changes within a module. This is a typical approach in software development and is implemented using the check-in and check-out process. If multiple participants are allowed to contribute to different aspects of a module, it is recommended that the module be partitioned into separate modules and interdependencies be defined between them.

**Cost and Value:** It is assumed that cost \((C)\) and value \((V)\) are constant for each participant during the simulation. In a real scenario, however, the cost \((C)\) and value \((V)\) for a participant can change over time. For example, the cost of participation can reduce as an individual gains more experience with a project. Similarly, the value to individuals can increase as the product development proceeds and the product matures.

**Participants' expertise:** Currently, all the participants are assumed to be identical. There is no difference in terms of their expertise. Hence, any participant can contribute to
any module. However, in a real mass-collaborative scenario, different participants may be more suitable (and may prefer) to work on different modules. Further, the expertise of participants may also affect the rate at which the modules grow. An expert may take less time to develop a module than a novice participant. All these effects have been ignored to simplify the model in this dissertation.

The agent-based model of open source product development processes, presented in this section, is executed for an illustrative example – the development of mobile phones. The example is discussed next.

Section 3.4 - Illustrative Example – Development of Mobile Phones using Mass-Collaborative Product Development Processes

3.4.1. Mobile Phone Structure

Panchal [10] presents an example problem for studying the evolution of products in open source processes. The product model used in that publication consists of a set of core modules and a set of external modules. Such a product model is representative of a variety of software and physical products. It provides an illustrative example of product evolution in general open source product development processes. Panchal [10] observed that the evolution of core modules was completed ahead of other modules. The development of the external modules finished at the end of the process. However, for some complex product structures, it may be challenging to identify core modules and the external modules. The order of completion of different modules may also vary significantly with the product structure. Hence, in this dissertation, instead of using a general product model, we consider an example of a specific product type – mobile
phones. The product structures of mobile phones is derived from Holtta et al. [133, 134], who analyzed the structures of a cellular phone, a desk phone, a desktop, and a laptop. By utilizing the DSM, the authors decomposed these four products and analyzed their modularity.

Figure 3.7 - Decomposition of the mobile phone to identify its modules

The mobile phone is used as an example in this dissertation because: a) the basic DSM of mobile phone is already available from the literature and different possible ways of changing the product structure can be identified, b) the product structure of a mobile phone is complex enough so that valuable information about the effect of complex product structures can be captured in the simulation. Figure 3.7 shows the modules in a mobile phone. Holtta et al. [133, 134] utilized the DSM to represent the structure of mobile phones based on its physical connections. A mobile phone is decomposed into several physical modules that are used to address the major functions. For example, the battery provides energy to the mobile phone. The antenna connects with a base station and transmits information. According to the recent developments in mobile technology, a mobile phone is no longer merely viewed as a communication device. Instead, it is treated as a terminal that can be used for communication, entertainment and information exchange. In order to represent mobile phones with increasingly popular functionalities, a modified DSM of mobile phones is developed and used in this dissertation. It is shown in
Table 3.1.

The growth rate of each module is assumed to be 1% in each cycle. In order to use the product model presented in Section 3.3.1., an important aspect is to determine the percentage rework for the linkages between modules. It makes the DSM a numerical matrix with parameters that can be used for analysis. It is understandable that different physical connections will have different effects on the components. For example, the upper and lower cases of a mobile phone have a strong physical connection because they must have the same shapes and sizes to match with each other. Developing the numerical estimate for the percentage rework associated with each dependency is challenging. Hence, we classify the dependencies into three classes with different dependency strengths: a) strong dependence, b) medium dependence, and c) weak dependence. A strong dependence implies that the relationship between components is highly coupled, such as the upper and lower cases of the mobile phone. This classification is based on the ratings suggested by Pimmler and Eppinger [16]. The strengths of couplings in Table 3.1 are estimated by the authors based on their judgment. It is recognized that this is a subjective assessment. The rework percentage for strong dependence is assumed to be 0.5%. Figure 3.8(a) shows one example of strong dependence.

Table 3.1 - A numerical DSM with rework (W: Weak dependence with rework percentage = 0.1%, M: Medium dependence with rework percentage = 0.3%, S: Strong dependence with rework percentage = 0.5%)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Power Con.</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Power Cbl.</td>
<td>Upper Case</td>
<td>Lower Case</td>
<td>Speaker</td>
<td>Display Unit</td>
<td>Keypad</td>
<td>Microphone</td>
<td>Antenna</td>
<td>Main Board</td>
<td>Battery</td>
<td>Wi-Fi</td>
<td>GPS</td>
<td>Camera</td>
<td>Blue Tooth</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>---------</td>
<td>--------------</td>
<td>--------</td>
<td>------------</td>
<td>---------</td>
<td>------------</td>
<td>---------</td>
<td>-------</td>
<td>-----</td>
<td>--------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>M</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td></td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td></td>
<td>W</td>
<td>M</td>
<td>M</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td></td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>Blue Tooth</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The rework percentage for medium dependence is assumed to be 0.3%. An example of medium dependence is the connection between the upper case and main board in the mobile phone. The upper case constrains the size and the shape of the main board but they do not need to have exactly the same size and shape to get matched. Figure 3.8(b) displays the example of the medium dependence in a mobile phone.

![Figure 3.8](image)

**Figure 3.8 - Examples of a) strong dependence: the upper case and the lower case, b) medium dependence: the upper case and the main board, and c) weak dependence: the upper case and the antenna**

The rework percentage associated with the weak dependence is assumed to be 0.1%. An example of a weak dependence is the connection between the upper case and the antenna in the mobile phone. The upper case is primarily constrained by the location the antenna. In the case of the weak dependence, one component may be a part of the other.
component. For large components, the changes in connected small components have a minor effect on the large components. Figure 3.8(c) shows one example of weak dependence in a mobile phone.

We acknowledge that the choice of the percentage rework values for the three classes of dependencies is subjective. However, even with these subjective estimates, the general trends in the effect of product structure on mass-collaborative product development can be elicited. More accurate estimates of percentage rework for the dependencies can be used to refine the results.

By applying three different kinds of dependence, a numerical DSM is developed, which represents not only the relationships between modules of a mobile phone, but also indicates different degrees of the relationships between them. The numerical DSM presented in Table 3.1 is the basis for the simulation results presented in the Section 3.5.

3.4.2. Modeling Different Mobile Phone Structures

As shown in Section 3.4.1, a mobile phone has a combination of slot and bus structures. From its numerical DSM with percentage rework, the upper case and the main board have a significantly greater number of interfaces than others. Hence, these components act as buses. In order to analyze the structures and modularity of products and their effects on the evolution time in mass-collaborative product development, the mobile phone structure is gradually converted from slot structure to bus structure by sequentially decoupling interfaces between different components. For example, the interface between the power cable and the main board is decoupled using a standardized interface so that only the main board affects the power cable. The decoupling of components is carried out sequentially to study the effect of individual dependencies on
the product evolution. For example, after decoupling the dependence between blue-tooth and the main board and lower case, the dependence between the camera and the main board and lower case is decoupled. Then the dependence between the GPS and the main board and lower case is removed.

Decoupling of different components physically refers to the design changes that eliminate the propagation of changes from one component to another. For example, the coupling between the speaker and the main board is considered high if the speaker is embedded on the main board. Any change in the size or geometry of the speaker affects the location and configuration of the other components on the main board, thereby requiring redesign. One way of decoupling the speaker from the main board is to have a separate physical speaker attached to the main board through a standard interface. In that case, as long as the interface remains standard, the changes in the design of the speaker do not propagate to main board. Similarly, other components can also be decoupled.

Table 3.2 displays the series of steps follows for gradually converting the slot structure into bus structure. The product structures corresponding to the different steps in decoupling are labeled from A through M. The corresponding links eliminated in each step are listed in the table. For example, in product structure B, the links between a) camera and mother board and b) camera and lower case are decoupled in addition to the links already decoupled in the product structure A. As the modules are progressively decoupled, the modularity of the product structure increases from A through M. The results from the execution of the agent-based model for each of these product structures are discussed in Section 3.5
Table 3.2 - Different product structures generated by sequentially decoupling different components of mobile phones

<table>
<thead>
<tr>
<th>Product Structure</th>
<th>Additional links decoupled (Links Between)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>blue tooth &amp; main board&lt;br&gt;blue tooth &amp; lower case</td>
</tr>
<tr>
<td>B</td>
<td>camera &amp; main board&lt;br&gt;camera &amp; lower case</td>
</tr>
<tr>
<td>C</td>
<td>GPS &amp; main board&lt;br&gt;GPS &amp; lower case</td>
</tr>
<tr>
<td>D</td>
<td>Wi-Fi &amp; main board&lt;br&gt;Wi-Fi &amp; lower case</td>
</tr>
<tr>
<td>E</td>
<td>battery &amp; main board&lt;br&gt;battery &amp; lower case</td>
</tr>
<tr>
<td>F</td>
<td>antenna &amp; main board&lt;br&gt;antenna &amp; upper case</td>
</tr>
<tr>
<td>G</td>
<td>microphone &amp; main board&lt;br&gt;microphone &amp; upper case</td>
</tr>
<tr>
<td>H</td>
<td>keypad &amp; main board&lt;br&gt;keypad &amp; upper case</td>
</tr>
<tr>
<td>I</td>
<td>display unit &amp; main board&lt;br&gt;display unit &amp; upper case</td>
</tr>
<tr>
<td>J</td>
<td>speaker &amp; main board&lt;br&gt;speaker &amp; upper case</td>
</tr>
<tr>
<td>K</td>
<td>lower case &amp; main board&lt;br&gt;lower case &amp; upper case</td>
</tr>
<tr>
<td>L</td>
<td>upper case &amp; main board</td>
</tr>
<tr>
<td>M</td>
<td>power cable &amp; main board</td>
</tr>
</tbody>
</table>

Section 3.5 - Results from the Execution of the Model for different Mobile Phone architectures

The model is executed using different product structures obtained by sequentially decoupling the components of the mobile phone. The evolution times for the product and each module are determined for each scenario. The effect of the product structure on the evolution time is determined. The product architectures are characterized using the degree of modularity (DOM) metric. In Sections 3.5.1 through 3.5.3, the results are presented for a baseline set of parameters. A sensitivity analysis of the parameters around the baseline values is presented in Section 3.5.4.
In general open source processes, such as open source development, a large number of participants are involved. The number of participants is generally very large compared to hierarchical product development. It can range anywhere from a few hundred to thousands of participants. The baseline number of participants (N) was set at 2000. The baseline of cost (C) is distributed across participants as a uniform random variable between [1 10]. The baseline value (V) is distributed uniformly between [1 15]. These distributions are used to account for the variation in the participants in terms of their expertise, interest levels, goals, etc. A lower cost of participation, for example, corresponds to greater expertise. Similarly a higher value corresponds to greater benefit from the development of the product. The model is stochastic in nature due to the probabilistic decision-making process adopted by different participants. To determine the effect of the stochastic nature of the model on the results, the model is executed multiple times using the same initial conditions. It is observed that the results from different runs are almost identical. This is primarily due to the large number of participants and the large number of cycles for which the model is executed.

Figure 3.9 through Figure 3.10 show the evolution of product and corresponding modules for structures ranging from slot-like structure to bus-like structure. Four structures (A, D, G, and J) are chosen to illustrate this process. Figure 3.11 provides a comparison among the evolution corresponding to structures A, D, G, and J.
Figure 3.9 - Evolution of modules in Structure A

Figure 3.10 - Evolution of modules in structure J

Figure 3.11 - Comparison of evolution of products with different structures
3.5.1. Effect of Product Architecture on the Evolution of the Overall Product

For the structures ranging from slot-like structure to the bus-like structure, the degree of modularity and the modularity measure proposed by Guo and Gershenson [135] are calculated. Note that the DOM increases but the modularity measure by Guo and Gershenson reduces as the modules are increasingly decoupled (i.e., from the slot-like architecture to the bus-like architecture). The evolution time of the product is also determined from the figures showed above. Based on these, the changes in the evolution time with increasing degree of modularity (DOM) and decreasing Modularity Measure are determined. Thus, the relationship between the DOM and the evolution time of the product can also be viewed as the relationship between the structures and the evolution time of the product in open source product development processes.

![Figure 3.12 - The relationship of degree of modularity and evolution time](image)

The relationship for the mobile phone example is plotted in Figure 3.12. It is observed that both the DOM and the modularity measure show similar behavior (the only difference is that DOM reduces when the modularity measure increases). Hence, the figure is only discussed in terms of DOM. When the DOM of the product is less than a
certain value (0.89 in this case), the development of the overall product does not reach 100% because the amount of rework due to dependencies is higher than the growth of the modules. This threshold depends on the product architecture, the growth rate of modules, and the strength of dependencies. The curve also shows that within the range [0.89 and 0.91] of the degree of modularity, the decoupling of the links causes a rapid decrease in the evolution time of the product. Beyond the degree of modularity of 0.91, the evolution time decreases at a slower rate. A degree of modularity of 1 represents totally modular product structure (i.e., all the modules are independent of each other).

3.5.2. Effect of Module Modularity on the Evolution of Individual Modules

The module modularity of different components is different. The major components that connect with many other components have higher module modularity. The components that connect with one or more other components have lower modularity. In a mobile phone, the upper case, the lower case and the main board are major components because they have higher module modularity. Other components are external components. Hence the upper case, the lower case, and the main board are selected for analysis. The relationship between the module modularity of the components and the corresponding evolution time is plotted in Figure 3.13.
Figure 3.13 - The effect of module modularity on the evolution time of three modules

For the major components, when the degree of modularity increases, the time required for the evolution of these components decreases. When the module modularity is less than 0.8 (i.e., greater coupling) the decomposition causes rapid decrease in evolution time. Hence, for the slot structure, the best strategy is to decouple the components whose module modularity is low. Decoupling these components results in a faster evolution of the entire product. The power converter, which has the highest module modularity, is also studied. It is observed that this decoupling does not have a significant effect on the evolution of the entire product.

3.5.3. Effect of Alternate Decoupling Sequences

The discussion of evolution of products as well as modules in Sections 3.5.1 and 3.5.2 is based on a specific decoupling sequence presented in Table 3.2. In this section, three alternative decoupling sequences ($S_1$, $S_2$, and $S_3$) are used to evaluate the effect of decoupling sequence on the time required for product evolution. Table 3.3 displays the three decoupling sequences. The DOM is calculated for the resulting product structures at each step.

<table>
<thead>
<tr>
<th>Steps of Decoupling</th>
<th>Sequence $S_1$</th>
<th>Sequence $S_2$</th>
<th>Sequence $S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decoupled links between</td>
<td>Decoupled links between</td>
<td>Decoupled links between</td>
</tr>
<tr>
<td>1</td>
<td>power cable &amp; main board (DOM: 0.8886)</td>
<td>antenna &amp; main board antenna &amp; upper case (DOM: 0.8895)</td>
<td>microphone &amp; main board microphone &amp; upper case (DOM: 0.8886)</td>
</tr>
<tr>
<td>2</td>
<td>upper case &amp; main board (DOM: 0.8914)</td>
<td>battery &amp; main board battery &amp; lower case (DOM: 0.8914)</td>
<td>keypad &amp; main board keypad &amp; upper case (DOM: 0.8924)</td>
</tr>
<tr>
<td>3</td>
<td>lower case &amp; main board lower case &amp; upper case</td>
<td>Wi-Fi &amp; main board Wi-Fi &amp; lower case (DOM: 0.8933)</td>
<td>display unit &amp; main board display unit &amp; upper case (DOM: 0.8933)</td>
</tr>
<tr>
<td></td>
<td>(DOM: 0.899)</td>
<td>GPS &amp; main board (DOM: 0.8943)</td>
<td>speaker &amp; main board (DOM: 0.8952)</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
<td>---------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>speaker &amp; main board</td>
<td>display unit &amp; main board camera &amp; lower case (DOM: 0.8981)</td>
<td>display unit &amp; upper case (DOM: 0.9048)</td>
</tr>
<tr>
<td></td>
<td>speaker &amp; upper case</td>
<td>display unit &amp; upper case (DOM: 0.9086)</td>
<td>display unit &amp; upper case (DOM: 0.901)</td>
</tr>
<tr>
<td>5</td>
<td>GPS &amp; main board</td>
<td>camera &amp; main board lower case &amp; upper case (DOM: 0.8971)</td>
<td>GPS &amp; main board lower case &amp; upper case (DOM: 0.8971)</td>
</tr>
<tr>
<td></td>
<td>GPS &amp; lower case</td>
<td>camera &amp; main board lower case &amp; upper case (DOM: 0.8971)</td>
<td>GPS &amp; main board lower case &amp; upper case (DOM: 0.8971)</td>
</tr>
<tr>
<td>6</td>
<td>keypad &amp; main board</td>
<td>blue tooth &amp; main board blue tooth &amp; lower case (DOM: 0.899)</td>
<td>keypad &amp; main board blue tooth &amp; lower case (DOM: 0.899)</td>
</tr>
<tr>
<td></td>
<td>keypad &amp; upper case</td>
<td>blue tooth &amp; main board blue tooth &amp; lower case (DOM: 0.899)</td>
<td>blue tooth &amp; main board blue tooth &amp; lower case (DOM: 0.899)</td>
</tr>
<tr>
<td>7</td>
<td>microphone &amp; main board</td>
<td>power cable &amp; main board (DOM: 0.901)</td>
<td>power cable &amp; main board (DOM: 0.9)</td>
</tr>
<tr>
<td></td>
<td>microphone &amp; upper case</td>
<td>power cable &amp; main board (DOM: 0.901)</td>
<td>power cable &amp; main board (DOM: 0.9)</td>
</tr>
<tr>
<td>8</td>
<td>antenna &amp; main board</td>
<td>upper case &amp; main board upper case &amp; main board (DOM: 0.901)</td>
<td>upper case &amp; main board upper case &amp; main board (DOM: 0.901)</td>
</tr>
<tr>
<td></td>
<td>antenna &amp; upper case</td>
<td>upper case &amp; main board upper case &amp; main board (DOM: 0.901)</td>
<td>upper case &amp; main board upper case &amp; main board (DOM: 0.901)</td>
</tr>
<tr>
<td>9</td>
<td>battery &amp; main board</td>
<td>lower case &amp; main board lower case &amp; main board (DOM: 0.9105)</td>
<td>lower case &amp; main board lower case &amp; main board (DOM: 0.9105)</td>
</tr>
<tr>
<td></td>
<td>battery &amp; lower case</td>
<td>lower case &amp; main board lower case &amp; main board (DOM: 0.9105)</td>
<td>lower case &amp; main board lower case &amp; main board (DOM: 0.9105)</td>
</tr>
<tr>
<td>10</td>
<td>Wi-Fi &amp; main board</td>
<td>speaker &amp; main board speaker &amp; upper case (DOM: 0.9105)</td>
<td>speaker &amp; main board speaker &amp; upper case (DOM: 0.9105)</td>
</tr>
<tr>
<td></td>
<td>Wi-Fi &amp; lower case</td>
<td>speaker &amp; main board speaker &amp; upper case (DOM: 0.9105)</td>
<td>speaker &amp; main board speaker &amp; upper case (DOM: 0.9105)</td>
</tr>
<tr>
<td>11</td>
<td>GPS &amp; main board</td>
<td>display unit &amp; main board display unit &amp; upper case (DOM: 0.9181)</td>
<td>display unit &amp; main board display unit &amp; upper case (DOM: 0.9181)</td>
</tr>
<tr>
<td></td>
<td>GPS &amp; lower case</td>
<td>display unit &amp; main board display unit &amp; upper case (DOM: 0.9181)</td>
<td>display unit &amp; main board display unit &amp; upper case (DOM: 0.9181)</td>
</tr>
<tr>
<td>12</td>
<td>camera &amp; main board</td>
<td>keypad &amp; main board keypad &amp; upper case (DOM: 0.921)</td>
<td>keypad &amp; main board keypad &amp; upper case (DOM: 0.921)</td>
</tr>
<tr>
<td></td>
<td>camera &amp; lower case</td>
<td>keypad &amp; main board keypad &amp; upper case (DOM: 0.921)</td>
<td>keypad &amp; main board keypad &amp; upper case (DOM: 0.921)</td>
</tr>
<tr>
<td>13</td>
<td>blue tooth &amp; main board</td>
<td>microphone &amp; main board microphone &amp; upper case (DOM: 0.9219)</td>
<td>microphone &amp; main board microphone &amp; upper case (DOM: 0.9219)</td>
</tr>
<tr>
<td></td>
<td>blue tooth &amp; lower case</td>
<td>microphone &amp; main board microphone &amp; upper case (DOM: 0.9219)</td>
<td>microphone &amp; main board microphone &amp; upper case (DOM: 0.9219)</td>
</tr>
<tr>
<td></td>
<td>(DOM: 0.9219)</td>
<td>(DOM: 0.9219)</td>
<td>(DOM: 0.9219)</td>
</tr>
</tbody>
</table>

The degree of modularity vs. evolution time for the three alternative decoupling sequences is plotted in Figure 3.14. It is observed from the figure that different decoupling sequences can result in different rates of reduction in the product evolution time. Even for the same degree-of-modularity values, the evolution time can be different. This is because DOM is a measure of the product structure that depends only on the dependencies between modules. It does not account for the dynamics of open source processes, particularly the manner in which individuals’ participation decisions are made.
Hence, the fastest reduction in the product evolution time can be achieved by appropriately choosing the decoupling sequence.

For the cell phone example, it is observed that when the degree of modularity varies between 0.895 and 0.91, there is significant difference in the product evolution time. In some cases, it is desirable to decouple minimum number of links to maintain the product’s original structure. In such cases, the product evolution time can be reduced by appropriately choosing the links to decouple. In Figure 3.14, the decoupling sequence $S_1$ is the most efficient because it results in the fastest product completion. It can also be observed that when the degree of modularity is greater than 0.905, all decoupling sequences result in identical completion times. This indicates that if entire product is decoupled into bus-like architecture, which has a higher degree of modularity, the decoupling sequences have minor effect on the product completion time.

![Comparison of different decoupling sequences](image)

**Figure 3.14 - Comparison of different decoupling sequences**

Furthermore, it is observed from the four decoupling sequences that the key steps are to decouple the links among the upper case, the lower case, and the main board, which are the major components in the product. This is primarily because the major components are components that are connected to large numbers of other components.
The likelihood of changes propagating from major components to a number of other components is high. In all of the four decoupling sequences, before we decouple the links among these major components, the percentage completion for the product does not converge to 100% because of the significant rework. When the links among these major components are decoupled, convergence to 100% completion takes place. Hence, the insight from these results is that a suitable decoupling sequence should focus on decoupling the links among major components first, and then on decoupling other components.

3.5.4. Critical Analysis of the Effect of Model Parameters

The results presented in the previous sections are based on the baseline set of parameters: \(N = 2000\), \(C = [1 \ 10]\), and \(V = [1 \ 15]\). To study the effect of these parameters, we perform sensitivity analysis by fixing the structure other varying only one parameter at a time. The growth of the product for different values of \(N\), \(V\), and \(C\) are presented in Figure 3.15 (a), (b), and (c) respectively. Since the cost and value parameters are uniformly distributed across participants, the lower bounds of cost and value distributions are kept constant at 1. The upper bounds \(C_u\) and \(V_u\) are varied. The architecture is fixed at \(J\) and the other parameters are fixed at their baseline values. It is observed from Figure 3.15 that the growth of the products is fastest for the baseline parameter values. The growth is slowed down by a) decreasing the number of participants below 100, b) decreasing the value below 15, and c) increasing the cost of participation above 20. Any increase in the number of participants and value, and any decrease in the cost of participation beyond these values do not affect the growth characteristics of the product.
This can be attributed to the participation strategy of individuals presented in Equation (3.3). The three parameters in the model directly affect the number of active participants in each cycle. It can be observed from Equation (3.3) that the probability of contribution by each individual reduces as the number of individuals increases, the cost increases, and/or the value decreases. As the probability of contribution reduces, the average number of active participants in each cycle reduces. Due to the simplifying assumptions in the model, the growth characteristics of the product are mainly dependent on the average number of participants contributing to the product development effort. When the average number of participants is low, only the least developed modules (which change in each cycle) are worked on. On the other hand, after the number of participants has reached a critical value, when all the modules are locked by active participants, addition of more active participants does not affect the growth of the modules. Hence, the growth of the product remains the same even after the number of total participants is increased in the model.
Figure 3.15 - The effect of changing (a) number of participants, (b) value, and (c) cost on the growth of products with Structure J

We acknowledge that this model presents a simplified view of open source processes, which are highly complex in nature due to the interactions between human beings with subjective values. However, the strength of the model is its extensibility. The agent-based nature of the model makes it particularly suitable for modeling the dynamics of open source processes because it enables modeling of individual decisions and product structures independently and allows the study of their interdependent dynamics. The model can be extended in the future to account for the nuances of open source processes.

Agent-based modeling is one of many modeling approaches that can be used to
model open source processes. Due to the relatively simple nature of the model, we believe that the same results can also be replicated with other modeling approaches such as general discrete event simulation or even simple difference equations. Hence, the agent-based part is not particularly necessary for the assumptions used in the model. However, the agent-based model can be easily extended by improving the product and participant models, has an advantage over other techniques. Other techniques such as difference equations or general discrete event simulations would be very difficult to use as the model is extended and refined.

Section 3.6 - Discussion and Future Works

Research in the field of open source processes is in its early stages. Successful application of open source requires a deeper understanding of the dynamics of participation and resulting product evolution. One of the key factors affecting the product evolution in open source processes is the product structure. In this chapter, we utilize an agent-based model to study the effect of different product structures on product evolution. The model was first presented by Panchal [10] to explore product evolution patterns, the effects of the number of participants, prior work, and participant incentives. In this chapter, we extend the study further. The primary research contribution of this chapter is the analysis of the relationship between product structure and product evolution in open source product development. The study is presented in the context of a real product – mobile phones.

The simulation shows the product structure can be characterized by DOM and completion time. The product with high DOM has low completion time. Based on the execution of the agent-based model, it is observed that different kinds of product
structures lead to significantly different product evolution characteristics. Integral structures with low DOM result in higher product development times. Modular structures with high DOM are ideal for open source product development. When the degree of modularity is lower than a certain threshold, the rework is higher than the product growth. In those cases, the product development effort does not lead to a complete product. As the modularity of the product increases, there is a steep decrease in the product completion time. Finally, beyond certain degree of modularity, the reduction in completion time is small. The bus architecture is suitable for many products since it lies within the region of maximum decrease of product completion time.

Appropriate product structures for open source product development can be achieved by adapting existing product structures through strategic decoupling of the key interactions between components. It is shown in this chapter that the sequence of decoupling has an effect on the product completion time. The model helps in comparing alternative decoupling sequences and identifying the best sequence. As a summary, we illustrate the importance and impact of product structure in open source product development. The agent-based modeling and simulation, combined with the Design Structure Matrix provides an effective methodology for analyzing the dynamics of product evolution in open source product development.

The model presented in this chapter is an initial step towards computational modeling of open source processes. There are significant opportunities for improving the models by a) refining the product and participant models, and b) relaxing the assumptions listed in Section 3.3.4. The product model can be refined by more accurate quantifications of the dependencies and associated percentage rework. The dependencies
can also be associated with the probability of rework. The participant model can be improved by using more sophisticated approaches such as Utility functions to model costs and values. The distribution of cost and value across participants can be more accurately assessed. Such information can be gained either from past projects or by polling the existing participants. The key assumptions about growth rates, locking, and participants’ expertise can be relaxed in the future models. Finally, the model in this chapter does not account for the information flow between participants. The information flow is dependent on the structures of communities involved in such mass collaborative processes. All these are avenues of modeling the nuances of real open source projects that are currently absent in the simplified model presented in this dissertation. At the same time, we would also like to highlight that just like in any other model addition of particular details makes the model more specific and less general. Such a detailed model is useful for gaining specific insights about certain projects but is limited for gaining general insights about general open source processes. Hence, depending on the modeling goals an appropriate level of abstraction should be sought.
4. CHAPTER FOUR - ANALYSIS AND MODELING PRODUCT STRUCTURE ON OPEN SOURCE PROCESSES

Section 4.1 - Motivation: Answering RQ2

The RQ2 is: how can the evolution of product structure be characterized in open source processes? The corresponding approach to answer the research question 2 is: to model products as networks with nodes and links, and to characterize the evolution of product structure using node level mechanisms.

In existing studies, 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. 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, the analysis does not capture the evolution of structural complexity (which depends on how the entities are linked). The dynamics of creation and removal of interfaces between entities is also not captured. The motivation of this chapter is to address this research limitation by analyzing the product evolution based on node-level mechanisms. The node-level mechanisms are included in two categories: 1) addition or deletion of nodes, and 2) addition or deletion of links. These two categories are further 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? Such node-level mechanisms provide underlying dynamics through which product evolve.

In this chapter, we employ three case studies and analyze the corresponding node-level mechanisms respectively. After analysis of the node-level mechanisms from product evolution, we built the generative model to simulate the product evolution processes based on proposed node-level mechanisms. In this model, the product structure is evolved over periods based on the proposed node-level mechanisms. The simulated product structures are compared with the original product structure over time. The matching between the simulation results and the original product structures indicates that the product evolution can be modeled only based on the node-level mechanisms. Hence, the proposed node level mechanisms can be served as underlying dynamics governing the product evolution. The RQ2 can be answered by the analysis of node-level mechanisms for individual open source project. Figure 4.1 illustrates answering RQ2 by approach 2 in this chapter.
This chapter is organized as follows. A discussion of the mechanisms of network evolution and application to Drupal is presented in Section 4.2. The network-level properties of the Drupal product structure are evaluated in Section 4.3. The proposed model based on the mechanisms, and its application to Drupal, WordPress, and phpMyAdmin are presented in Section 4.4. Finally, closing comments are present in Section 4.5.

Section 4.2 - Node-level Mechanisms Associated with 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 product network can refer to functions, classes, files, or features. The level of abstraction considered in this dissertation is the function-call level. Hence, functions are the nodes in the network. We discuss the mechanisms in Section 4.2.1 and apply them to the evolution of Drupal product network in Section 4.2.2.

4.2.1. Node-level Mechanisms for Modeling the Evolution of Open Source Software

The six node-level mechanisms through which networks evolve are illustrated in Figure 4.2, and discussed next.
Figure 4.2 - Node-level mechanisms for the evolution of product networks

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, etc. in software to address new requirements, specifications and features. The trends in the number of additional nodes can be determined by comparing consecutive versions of the software.

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 of the software.

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 degree of existing node:

\[
P(A_{e,n}) = F_1(K_e) \tag{4.1}\]

where \(A_{e,n}\) represents the attachment between existing nodes and a new node, \(K_e\) represents the degree of the existing node. To determine the relationship in Eq. (4.1), 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. A simple example is illustrated in Figure 4.3. When a new node (N) is added, the nodes with degree 2 (i.e., A and B) have 1/3 probability of linking with the new node. Nodes with degree 1 (i.e., C and D) have 1/6 probability of linking to the new module.

![Diagram of nodes A, B, C, and D with N as a new module.](image)

**Figure 4.3 - Probability of new nodes linking with existing nodes: an example**

**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:

\[ P(A_{n1,n2}) = F_2(K_{n,i}) \]  

(4.2)

where \( A_{n1,n2} \) represents the attachment between two new nodes (n1 and n2), \( K_{n,i} \) 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:

\[ P(A_{e1,e2}) = F_3(K_e) \]  

(4.3)

where \( A_{e1,e2} \) represents the attachment between existing nodes, \( K_e \) represents the degree of the existing nodes. The relationship in Eq. (4.3) is determined using the approach similar to the one shown in Figure 4.2.

**f) Removal of existing links.** Existing links can be removed in new software versions because of two reasons: i) the existing nodes are removed. In this case, the existing links associated with these nodes are also removed; ii) 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 given by the following function, and is calculated by comparing consecutive versions of the code:

\[ P(R_{e1,e2}) = F_4(K_e) \]  

(4.4)

where \( R_{e1,e2} \) represents the removal of links between existing nodes, \( K_e \) represents the degrees of existing nodes.

The existing degree-based models discussed in Section 2.3 present specific functional forms for the probability functions. For example, the Barabasi-Albert model [92] 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 \( F_1 \) through \( F_4 \). These functions are determined and calibrated based on the observed data.

**4.2.2. Utilizing the Node-level Mechanisms for Drupal Product Network**

Drupal [141] is an open-source content-management system which is used for the creation of community-based websites. Drupal has been under development since 2001.
In this dissertation, 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 structures [33, 47, 60, 62]. If function B is called within function A, then an interface is created between function A and function B.

In the first step, raw data about the product structure is extracted from the source code. The raw data consists of all the functions in the source code and the corresponding function calls. The data is used to derive 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 called Doxygen [142] is used to create the call graph. Having generated the networks for different versions of the code, consecutive versions of the code 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 4.1. The number of new nodes added and existing nodes removed, which represent node-level mechanisms (a) and (b) are obtained by comparing consecutive versions over time. 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 fulfilled, the outdated
features are removed, the features that are useful but not efficient are replaced, and bugs
are found and corrected. LaMantia et al. [19] propose change ratio, which measures the
number of new classes added and existing classes removed as compared with total
number of classes. LaMantia et al. [19] also detect a high value of change ratio for
Tomcat-main product.

Table 4.1 - Data associated with node-level mechanisms for Drupal network (version 2 through 5)

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

*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 Figure 4.4. The
probability functions are determined by fitting linear curves 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 [20] observed
the scale-free nature of software architectures as a commonality in a large number of
open source software products.
Figure 4.4 - Probabilities of creation of links between new and existing nodes (mechanism (c))

Mechanism (d): The probabilities with which new nodes are attached with each other are plotted in Figure 4.5. 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 observed that exponential functions provide good approximation of the relationships between the initial degrees and probabilities. The parameters of the exponential functions are shown in Figure 4.5.

Figure 4.5 - Probability of new nodes linking with each other (linking mechanism (d))

Mechanism (e): The probabilities of creation of links between two existing nodes are shown in Figure 4.6. 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 4.1. 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.

Figure 4.6 - Probability of existing nodes linking with each other (linking mechanism (e))

Mechanism (f): The probabilities of removal of existing links between nodes as functions of the degrees of nodes are shown in Figure 4.7. It is observed that linear trends from these log-log scale plots (indicating power law distribution) can be used to describe the probability functions.

Figure 4.7 - Probability of removal of existing links (Mechanism (f))

Section 4.3 - Network-Level Analysis of the Evolution of Drupal

4.3.1. Network Measures

After modeling the product structure as a complex network, the evolutionary characteristics of the corresponding network are explored. Complex network analysis
metrics [143] are employed to quantify the evolutionary characteristics of complex product structures. The metrics used in this dissertation are average degree, degree distribution, density, diameter, clustering coefficient, average shortest path, propagation cost, and clustered cost. The first six 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 [109]. It is chosen because it represents the average number of other nodes connected to a node, and indicates the complexity of the product [144]. According to existing studies [14, 145], the design complexity of a product, which indicates the redesign work in the product development processes, can be quantified as $D(A) = \frac{1}{n} \sum_{i=1}^{n} Z_i$, where $D(A)$ is the design complexity, $n$ is the number of components, and $Z_i$ is 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 [145] is also related to the average degree and can be expressed as: $M(A) = n + \frac{1}{n} \sum_{i=1}^{n} Z_i$. The degree distribution, $P(k)$ is defined as the fraction of nodes in the network with degree $k$ [146]. The degree distribution is important because it indicates the topology of a product structure network. For example, recent studies [20] show power-law distributions in the structure of a variety of software products.

*Density* of a network is the average proportion of links incident with nodes in a network [143]. 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 [147], higher network density implies that higher complexity can be reached.

*Diameter* is the largest distance between any two nodes of a connected network [148]. In the product, when an entity changes, the entities dependent on it may need rework [132]. The diameter of a product structure network indicates the farthest possible effect on other entities when one module is modified.

*Clustering coefficient* is the probability that two nearest neighbors of a vertex are also the nearest neighbors of one another [146]. 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 [20]. 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 significant less compared with the low clustering coefficient cases. 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 [149]. 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 [145], 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 [150]. The change complexity can be expressed as:

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

where \(d_{ij}\) is the shortest path between nodes \(i\) and \(j\).
Propagation cost, proposed by MacCormack et al. [18], 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 a set of intermediate nodes) connected to. It is calculated as the average “fan-out visibility” of nodes [18].

Clustered cost [18] 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 order to calculate clustered cost, the network is first clustered, and then weights are assigned to the dependencies depending on the location of the nodes within different clusters. In this chapter, we use the Girvan-Newman [151] clustering algorithm to assign nodes into clusters. MacCormack et al. [18] 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 $d_{ij} \times n^\lambda$ where $n$ is the size of the cluster and $\lambda$ is a user defined parameter (set to 2 in [18]). For links between nodes across different clusters, the dependency cost is $d_{ij} \times N^\lambda$ where $N$ is the size of the complete network.

4.3.2. Analysis of the Structures of Different Versions of Drupal

Using the source code, we identify characteristics of the product structure that
change over time and the characteristics that are invariant. The degree distribution plots for versions 2 through 5 are displayed in Figure 4.8. 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. [152] show that the degree distribution of Kowari follows a linear trend when the degree is larger than 4, while displays a homogenous trend when the degree is smaller than 4. LaBelle and Wallingford [153] 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.[18].

![Degree Distribution](image)

**Figure 4.8 - Degree distribution of product architecture networks from version 2 to version 5**

Other network characteristics for the four versions of Drupal are listed in Table 4.2. 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 Figure
4.9) reveals an interesting trend – 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 an average, new nodes attach to almost a constant number of existing nodes [20]. However, such a conclusion ignores the fact that a large portion of the nodes and links are also removed from the network. In Section 4.4.2, we investigate this in detail.

Table 4.2 - Characteristics of different versions of the Drupal product structure

<table>
<thead>
<tr>
<th>Version</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Nodes</td>
<td>248</td>
<td>412</td>
<td>635</td>
<td>1018</td>
</tr>
<tr>
<td>No. of Interfaces</td>
<td>661</td>
<td>1304</td>
<td>2041</td>
<td>3139</td>
</tr>
<tr>
<td>Average Degree</td>
<td>4.4981</td>
<td>6.122</td>
<td>6.123</td>
<td>6.081</td>
</tr>
<tr>
<td>Average Density</td>
<td>0.019</td>
<td>0.014</td>
<td>0.009</td>
<td>0.005</td>
</tr>
<tr>
<td>Diameter</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Clustering Coeff.</td>
<td>0.11</td>
<td>0.099</td>
<td>0.107</td>
<td>0.100</td>
</tr>
<tr>
<td>Avg. Shortest Path</td>
<td>2.895</td>
<td>2.901</td>
<td>2.965</td>
<td>3.084</td>
</tr>
<tr>
<td>Propagation Cost</td>
<td>0.0106</td>
<td>0.0086</td>
<td>0.0055</td>
<td>0.0035</td>
</tr>
<tr>
<td>Clustered Cost</td>
<td>92239</td>
<td>271051</td>
<td>676840</td>
<td>1815612</td>
</tr>
</tbody>
</table>

Figure 4.9 - Relationship between the number of functions and interfaces

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. [18] for
the structure of Mozilla. The diameter of the product structure is between 7 and 8, meaning that the longest possible path between two nodes is less than 8. A large diameter causes difficulty in managing the product development because the effect of modifying one node on other nodes cannot be detected immediately. In the Drupal project development, the diameter remains stable as compared to the exponential growth of interfaces.

The clustering coefficient remains constant and is close to 0.1 for versions 2 through 5. In Table 4.3, 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 corresponding random graphs. Besides, the clustering coefficients of product structure networks are independent of network sizes, while in the corresponding random graphs, the clustering coefficients are linearly decreasing. The comparison is consistent with the conclusion drawn by Valverde and Sole [20]. In their studies, high and size-independent clustering coefficient values are observed in OSS projects.

<table>
<thead>
<tr>
<th>Version</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product structure network</td>
<td>0.110</td>
<td>0.099</td>
<td>0.107</td>
<td>0.093</td>
</tr>
<tr>
<td>Corresponding random graph</td>
<td>0.022</td>
<td>0.015</td>
<td>0.007</td>
<td>0.004</td>
</tr>
</tbody>
</table>

As shown in Table 4.2, the propagation cost decreases from 0.0106 to 0.0035 for versions 2 through 5. As stated earlier, the propagation cost measures the average visibility of the nodes in the network. It measures the average number of nodes affected by a change in a node. A decrease in visibility indicates a decrease in the extent of
coupling within the network from Version 2 through Version 5. Propagation cost is based on the assumption that the costs associated with all links are equal. On the other hand, clustered cost is based on the assumption that links within a cluster have lower costs than the links across clusters. In other words, the assumption is that it is easier to address change propagation within a single module than changes across different modules. It is observed that the clustered cost for Drupal increases monotonically from 92239 to 1815612 for versions 2 through 5. This is revealing. While the decrease in propagation cost indicates a reduction in the extent of overall coupling in the software over time, the increase in clustered cost implies that more links are created across clusters than within clusters. Similar trends for propagation and clustered costs have also been observed by MacCormack et al. [18] for the Mozilla project after significant redesign in 1998.

**Section 4.4 - Generative Model for the Evolution of Product Structure**

**4.4.1. 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 4.10 outlines the execution of the model. The data for mechanisms (a) and (b) is used from Table 4.1. For mechanisms (c), (d), (e), (f), the numbers of interfaces created or removed are based on the functions listed in Figure 4.4 to Figure 4.7.
Three alternatives 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 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 reveal that many real-world networks (including OSS) have the scale free property (e.g., [20]). 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 Section 4.2.1 are
simulated based on the probability functions discussed in Section 4.2.2.

4.4.2. 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 4.11 displays the 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 Figure 4.11(a), 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 with the initial Drupal version 2 product structure 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 diameters of models with the initial random network and the initial version 2 network are the same as the Drupal product. The diameter of the model with the initial scale-free network is lower than the Drupal project. However, with the evolutionary processes, the diameter increases and finally converges to the Drupal product.
Figure 4.11 - Comparison of evolutionary characteristics at product level between Drupal product and the models

From Figure 4.11(d), it is observed that the clustering coefficient of models with the initial scale-free network and the initial version 2 network come close to the Drupal product over time. The model with initial random network has a small clustering coefficient at the beginning, which represents the random network property. However, the clustering coefficient significantly increases and converges to the Drupal product over time. From Figure 4.11(e), 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 Project is provided in Figure 4.12. At the beginning, the differences
among different initial networks are significant. The initial random network displays a Poisson distribution, while the initial scale-free network displays a power-law distribution. Both of them are different from the initial version 2 network. With the evolutionary processes, the degree distributions of three models converge to the degree distribution of Drupal project.

**Figure 4.12 - Comparison of degree distribution between Drupal product and the models**

From the execution of the models, we conclude that:

1) When the initial version 2 network is used in the model, the evolutionary characteristics including average degree, average density, diameter, clustering coefficient, 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.

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. In order to build confidence in these conclusions and to display the generality of the model, we use the generative model to two other open-source projects: Wordpress [154], and phpMyadmin [155]. The results from these projects are discussed in Section 4.4.3.

4.4.3. Applying the Model to Other Open Source Software (OSS) Projects

WordPress [154] is web software that can be used to create websites or blogs. The WordPress project started in 2003 to enhance the typography of everyday writing. We model the structures of four different versions of the software: V1.5, V2.0, V2.5, and V3.0. phpMyAdmin [155] is an open source tool intended to handle the administration of MySQL over the World Wide Web. The project started in 1998. We analyze four versions of the software: V3.1, V3.2, V3.3, and V3.4. The data for the node-level mechanisms for WordPress and phpMyAdmin projects are summarized in Table 4.4 and Table 4.5 respectively.

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Versions</th>
<th>V1.5 → V2.0</th>
<th>V2.0→V2.5</th>
<th>V2.5→V3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>Number of nodes added</td>
<td>271</td>
<td>1024</td>
<td>1511</td>
</tr>
<tr>
<td>(b)</td>
<td>Number of nodes removed</td>
<td>42</td>
<td>94</td>
<td>216</td>
</tr>
<tr>
<td>(c)</td>
<td>Interfaces created between new and existing nodes</td>
<td>401</td>
<td>1684</td>
<td>2812</td>
</tr>
<tr>
<td></td>
<td>Probabilities of linking</td>
<td>$P(A_{c,n}) = 0.005e^{0.0341K_e}$</td>
<td>$P(A_{c,n}) = 0.001e^{0.102K_e}$</td>
<td>$P(A_{c,n}) = 0.0002K_e + 0.0008$</td>
</tr>
<tr>
<td>(d)</td>
<td>Interfaces created between new nodes</td>
<td>300</td>
<td>1408</td>
<td>2119</td>
</tr>
<tr>
<td></td>
<td>Probabilities of linking</td>
<td>Probabilities of linking</td>
<td>Probabilities of linking</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------------------</td>
<td>-------------------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P(A_{n,n}) = 0.0006K_{n,i} + 0.0003$</td>
<td>$P(A_{n,n}) = 0.0003K_{n,i} + 0.0005$</td>
<td>$P(A_{n,n}) = 0.0003K_{n,i} + 0.0004$</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>Interfaces created among existing nodes</td>
<td>62</td>
<td>348</td>
<td>656</td>
</tr>
<tr>
<td>(f)</td>
<td>Interfaces removed among existing nodes</td>
<td>118</td>
<td>513</td>
<td>1495</td>
</tr>
<tr>
<td></td>
<td>Probabilities of link removal</td>
<td>$P(R_{e,e}) = 0.0042K_{e}^{0.389}$</td>
<td>$P(R_{e,e}) = 0.0008K_{e}^{0.78}$</td>
<td>$P(R_{e,e}) = 0.0002K_{e}^{0.7365}$</td>
</tr>
</tbody>
</table>

The generative model is executed based on the corresponding node-level mechanisms for the two projects. Similar to the analysis of Drupal, in Section 4.4.2, three initial networks are applied: a) the product structure network from initial version, b) a random initial network with the same number of nodes and interfaces as the product structure network, c) a scale-free network with the same number of nodes and interfaces.

Figure 4.13 displays a comparison of the evolutionary characteristics of the WordPress product as compared to that of the generated network models. Figure 4.14 displays the degree distributions of the Wordpress and the generated networks. Similarly, Figure 4.15 and Figure 4.16 show the evolutionary characteristics and degree distributions for the phpMyAdmin project. It is observed from these comparisons that the proposed network generation model faithfully replicates the characteristics of the original networks in each of the products if the initial original network is used as an input. Additionally, the network characteristics converge to those of the original network even if we use initial networks with other topologies. Hence, the proposed generational model is appropriate for modeling the evolution of software products.
Table 4.5 - Data related to node-level mechanisms in phpMyAdmin

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Versions</th>
<th>V3.1 → V3.2</th>
<th>V3.2→V3.3</th>
<th>V3.3→V3.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Number of nodes added</td>
<td></td>
<td>3</td>
<td>917</td>
<td>394</td>
</tr>
<tr>
<td>(b) Number of nodes removed</td>
<td></td>
<td>65</td>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>(c) Interfaces created between new and existing nodes</td>
<td></td>
<td>6</td>
<td>176</td>
<td>495</td>
</tr>
<tr>
<td>Probabilities of linking</td>
<td></td>
<td>P(A_{c,n}) = 0.2</td>
<td>P(A_{c,n}) = 0.0109K_e^{0.0515}K_e</td>
<td>P(A_{c,n}) = 0.0033K_e^{0.0204}K_e</td>
</tr>
<tr>
<td>(d) Interfaces created between new nodes</td>
<td></td>
<td>0</td>
<td>1238</td>
<td>338</td>
</tr>
<tr>
<td>Probabilities of linking</td>
<td></td>
<td>P(A_{n,n}) = 0.00009K_{n,i} + 0.0013</td>
<td>P(A_{n,n}) = 0.003K_e^{0.0329}K_e</td>
<td></td>
</tr>
<tr>
<td>(e) Interfaces created among existing nodes</td>
<td></td>
<td>6</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>Probabilities of linking</td>
<td></td>
<td>P(A_{c,c}) = -0.0019K_e + 0.1286</td>
<td>P(A_{c,c}) = -0.0016K_e + 0.1638</td>
<td>P(A_{c,c}) = 0.0002K_e + 0.0088</td>
</tr>
<tr>
<td>(f) Interfaces removed among existing nodes</td>
<td></td>
<td>162</td>
<td>23</td>
<td>324</td>
</tr>
<tr>
<td>Probabilities of link removal</td>
<td></td>
<td>P(R_{c,c}) = 0.0033K_e + 0.0003</td>
<td>P(R_{c,c}) = 0.0002K_e + 0.0292</td>
<td>P(R_{c,c}) = 0.0017K_e^{0.4332}</td>
</tr>
</tbody>
</table>

Figure 4.13 - Comparison of evolutionary characteristics between WordPress product and the

![Graphs showing evolutionary characteristics](image)
generated networks

Figure 4.14 - Comparison of degree distribution between WordPress product and the generated networks

Figure 4.15 - Comparison of evolutionary characteristics between phpMyAdmin product and the generated networks

Figure 4.16 - Comparison of degree distribution between phpMyAdmin product and generated networks
Section 4.5 - Conclusion and Future Works

The product structure plays an important role in the open source processes. Not only does it affect the efficiency of open source processes [156], but it also affect the community structure [37, 157]. 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 chapter: 1) longitudinal studies of a product’s evolution, 2) cross-sectional studies of evolution of different products, and 3) predictive analyses.

Longitudinal studies - The six mechanisms discussed in Section 4.2.1 provide different insights into how the products grow in size and complexity. Mechanisms (a) and (b) describe the number of nodes added and removed. Hence, they provide an indication of the growth rate and the extent of change in the products. Mechanisms (c) through (f) describe the network evolution in terms of linking of new and existing nodes. Hence, they indicate a change in structural complexity of products. Different probability functions
result in different topologies of the overall network. For example, simple probability function where the probability of linking is proportional to the nodes’ degrees results in a power-law degree distribution with $\gamma=3$. Non-linear attachment functions result in more complex degree distributions. Through these relationships between the node-level behaviors and the network’s global characteristics, the mechanisms can provide information about how specific design changes during a particular version affect the overall product structure.

*Cross-sectional studies* – The goal of this chapter is not to compare the evolution of different software applications. But this could be a potential application of the model proposed in his paper. The differences in the evolution of different products can be explained in terms of the differences between the corresponding 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: $P=\alpha K^\beta$ or $ae^{\beta K}$, where $\alpha$ and $\beta$ are coefficients and $K$ is the degree. In the prediction process, the fitted curves can be used to predict the coefficients $\alpha$ and $\beta$ for the mechanisms (e) - (f) based on the existing coefficients. Once the predicted coefficients $\alpha$ and $\beta$ 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.

Here, 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.).

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.

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.

In this chapter, the focus is on studying the evolution of product structures only. Further work is needed to model the co-evolution of products and communities of participants [48]. Such analysis is important to validate the mirroring hypothesis [33] according to which, the product structures and community structures mirror each other.
5. CHAPTER FIVE - ANALYSIS OF CO-EVOLUTION BETWEEN PRODUCT STRUCTURE AND COMMUNITY STRUCTURE

Section 5.1 - Motivation: Answering RQ3

Research Question 3 is: How can the co-evolution of product and community be identified in open source processes? The Approach 3 is: Modeling the product and community as hybrid networks with nodes and links, and to identify the co-evolution of product and community by node level, link level and cluster level mechanisms.

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 (RQ3) 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. Existing studies on evaluating the co-evolution between product and community are based on a static view of the product and community 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 co-evolved, it does not reveal the underlying dynamics that lead to the observed co-evolutionary characteristics. The motivation of this chapter is to reveal the co-evolutionary characteristics between product and community structures at three different levels: single-node level, pair-node level and cluster level. By analyzing the co-evolution between product and community at three levels, a comprehensive understanding of co-evolution between product and community structures can be formed. Hence, the RQ3 can be answered in a comprehensive way.

In this chapter, we propose five hypotheses, which belong to three levels. Table 5.1 displays the five hypotheses.

**Table 5.1 - Five Hypotheses at Three Levels**

<table>
<thead>
<tr>
<th>Hypotheses at Node Level</th>
<th><strong>Hypothesis 3.1.1</strong>: 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>Hypotheses at Link Level</td>
<td><strong>Hypothesis 3.1.2</strong>: 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 at Cluster Level</td>
<td><strong>Hypothesis 3.2.1</strong>: 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></td>
<td><strong>Hypothesis 3.2.2</strong>: If two people communicate with each other, then the corresponding modules also develop interfaces.</td>
</tr>
<tr>
<td></td>
<td><strong>Hypothesis 3.3</strong>: The communities get increasingly aligned with the anticipated communication patterns.</td>
</tr>
</tbody>
</table>

The hypotheses at node level and link level are focused on mechanisms of network evolution, specifically the addition of links representing module interfaces and communications between individuals. Hypotheses 3.1.1 and 3.1.2 are focused on the
impacts of new links of a product module and an individual respectively whereas Hypotheses 3.2.1 and 3.2.2 are focused on the impacts of links between pairs of product modules and pairs of individuals respectively (see Table 5.2). Hypothesis 3.1.1 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 3.2.1 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 3.1.1 and 3.2.1 are valid, it indicates that the product structure impacts the community structure. Similarly, the validity of Hypotheses 3.1.2 and 3.2.2 indicates the impact of community structure on the product structure. To validate these hypotheses, our approach proposed in this chapter is to build a hybrid network model to represent product structure, community structure and participation over time. Then we analyze the changes in product and community networks between consecutive time-steps for a specific open source community base on the hybrid network model. This approach is inspired by resent advances in network science and is particularly applicable to the highly evolutionary open source communities. The details are provided in Sections 5.2 and 5.3.

<table>
<thead>
<tr>
<th></th>
<th>Product → Community</th>
<th>Community → Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Hypothesis 3.1.1</td>
<td>Hypothesis 3.1.2</td>
</tr>
<tr>
<td>Links</td>
<td>Hypothesis 3.2.1</td>
<td>Hypothesis 3.2.2</td>
</tr>
</tbody>
</table>

The hypothesis at cluster level (H3.3) is focused on the coordination between product and community at the clusters over time. Existing studies of coordination between product and community are focused on measuring the direct interactions
between individuals. In open source processes, the number of individuals is generally very large. Hence, the coordination measures proposed by existing studies are very small. But this does not necessarily mean that coordination is not taking place. In H3.3, instead of measuring direct interactions between individuals, we measure the alignment of clusters in open source processes. To validate H3.3, the networks are first clustered into communities (clusters) and the resulting clusters are compared with clusters anticipated based on the links in the product network. Figure 5.1 illustrates the proposed approaches in this chapter to answer RQ3 by approach 3.

![Figure 5.1 - Illustration of Answering RQ3 by Approach 3](image)

**Section 5.2 - Analysis Approach**

**5.2.1. Modeling products and communities as networks**

In this chapter, 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 [157]. In software products, different levels of abstraction including class level [19, 64], file level [18, 42], and function level [61] 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 [158] use online chat logs and modification requests to represent communication links. Ducheneaut [159] uses emails and Concurrent Versioning System (CVS) logs to identify communications between individuals. Sack et al. [160] derive the communication links among individuals from emails. Wagstrom et al. [161] 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) [158, 162], e-mail communication [163], questionnaires and surveys [48, 164].

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. [34], Ducheneaut [159], Madey et al. [26], Ducheneaut [159], and Xue et al. [27, 28] to describe task assignment to participants. Based on the bipartite network linking the product modules and the individuals, we define

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

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

In this chapter, 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
5.2. 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.

![Hybrid network model](image)

**Figure 5.2 - A hybrid network model to represent product structure, community structure and participation**

5.2.2. Analysis of product and community evolution in consecutive versions

We propose two techniques for validating the hypotheses listed in Section 5.1. Both
techniques compare consecutive versions of products and communities to determine the
evolutionary characteristics. The first technique is proposed to validate hypotheses 3.1.1
and 3.1.2. To validate H3.1.2, 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
The newly created links are illustrated in Figure 5.3. In this figure, participant A has two newly created relationships and the corresponding module set has two newly created interfaces. Hypothesis 3.1.2 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 3.1.1, we determine the relationship between the number of new interfaces of modules and the new communication links of the corresponding participant sets.

Figure 5.3 - Extracting the relationship between the number of newly created links of participants and newly created links of modules associated with that participant (for Hypothesis 3.1.2)

The second technique is employed to validate hypotheses 3.2.1 and 3.2.2. In order to validate hypothesis 3.2.2, 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 5.4. 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 3.2.2 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 3.2.1. The results from the analysis of consecutive versions of the products are presented in Section 5.3.2.

**Figure 5.4 - 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 3.2.2)**

### 5.2.3. Cluster comparison

To validate Hypothesis 3.3, 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 5.5. 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 5.2.3.1 and 5.2.3.2 respectively.
5.2.3.1. Extraction of clusters

A number of techniques have been proposed in the literature to identify clusters within networks. In this chapter, we use the modularity-based method proposed by Newman-Girvan [151, 165] to partition the community network. Using this method, a modularity function, $Q$, is determined for each partition:

$$Q = \frac{1}{4m} \sum_{ij} \sum_{r} \left( A_{ij} - \frac{k_i k_j}{2m} \right) S_{ir} S_{jr}$$  \hspace{1cm} (5.1)

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 [166] to identify a good approximation of the partition in unimodal networks, and the modularity-based method proposed by Barber [167] for bipartite networks. In this dissertation, 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.
5.2.3.2. Comparison of clusters

The overlap of clusters between two partitions of a network can be measured in different ways. In this dissertation we use the Rand index [168] for comparing clusters, which is a measure of the similarity between two data partitions. The Rand index can be expressed as:

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

(5.2)

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 5.3.2.3

Section 5.3 - An Example from Open Source Software

In this section, we present the approach using an open source software project, Drupal [141]. 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.

5.3.1. 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 [142], 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 5.6. The basic network statistics for versions 4-7 is shown in Table 5.3.
Figure 5.6 - Generation of the Drupal networks with three types of links

Table 5.3 - 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>

5.3.2. Testing the hypotheses for Drupal product and community

5.3.2.1. Hypotheses 3.1.1 and 3.1.2

Hypothesis 3.1.1 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 5.7). 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 5.4. 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 3.1.1 for Drupal.

![Figure 5.7 - New interfaces of files vs. average number of new communication links of the corresponding participant sets](image)

### Table 5.4 - Regression model corresponding to Hypothesis 3.1.1

<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>Data points</td>
<td>187</td>
<td>448</td>
<td>529</td>
</tr>
<tr>
<td>Regression</td>
<td>$\log_{10}(y) = 1.04 + 0.00303 \times x$</td>
<td>$\log_{10}(y) = 1.46 + 0.0000656 \times x$</td>
<td>$\log_{10}(y) = 1.64 + 0.0000538 \times x$</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.609</td>
<td>0.647</td>
<td>0.604</td>
</tr>
<tr>
<td>T-test (slope)</td>
<td>$T = 16.98$, P-value &lt; 0.001</td>
<td>$T = 28.62$, P-value &lt; 0.001</td>
<td>$T = 28.32$, P-value &lt; 0.001</td>
</tr>
</tbody>
</table>

Hypothesis 3.1.2 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 5.8). The corresponding linear regression model is presented in Table 5.5. 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 3.1.2, 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 3.1.2.

Figure 5.8 - New communication links vs. the average number of new interfaces of files in the corresponding module sets

Table 5.5 - Linear regression model corresponding to Hypothesis 3.1.2

<table>
<thead>
<tr>
<th>Data Points</th>
<th>Version 4-5</th>
<th>Version 5-6</th>
<th>Version 6-7</th>
</tr>
</thead>
<tbody>
<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 = - 0.037 + 0.527 x</td>
<td>y = - 0.0576 + 0.499 x</td>
<td>y = - 0.0614 + 0.203 x</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 P-value = 0.049</td>
<td>T = 6.21 P-value &lt; 0.001</td>
<td>T = 3.45 P-value = 0.001</td>
</tr>
</tbody>
</table>

5.3.2.2. Hypotheses 3.2.1 and 3.2.2

Hypotheses 3.2.1 and 3.2.2 are focused on the effects of new relationships between pairs of modules and individuals respectively. Hypothesis 3.2.1 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 5.9). Similar to Hypothesis 3.1.1, 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 5.6. The P-value in each of the models is less than 0.001, indicating the statistical significance of the regression models. The $R^2$ 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 3.2.1.

![Figure 5.9 - New interfaces between two files vs. average number of new communication links between the corresponding pair of participant sets](image)

**Table 5.6 - Regression model corresponding to Hypothesis 3.2.1**

<table>
<thead>
<tr>
<th>Data Points</th>
<th>Version 4-5</th>
<th>Version 5-6</th>
<th>Version 6-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>$\log_{10}(y) = -0.134 + 0.00454x$</td>
<td>$\log_{10}(y) = -0.0449 + 0.00236x$</td>
<td>$\log_{10}(y) = 0.0998 + 0.00202x$</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.686</td>
<td>0.825</td>
<td>0.865</td>
</tr>
<tr>
<td>T-test (slope)</td>
<td>T = 22.17, P-value &lt; 0.001</td>
<td>T = 50.45, P-value &lt; 0.001</td>
<td>T = 73.47, P-value &lt; 0.001</td>
</tr>
</tbody>
</table>

Hypothesis 3.2.2 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 5.10. 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 5.7. As in the case of Hypothesis 3.1.2, 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 3.2.2.

**Figure 5.10** - New communication links between two individuals vs. average number of new interfaces between corresponding pair of module sets

**Table 5.7** - Regression model corresponding to Hypothesis 3.2.2

<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>113</td>
<td>465</td>
<td>580</td>
</tr>
<tr>
<td>Regression</td>
<td>( y = 0.734 + \frac{0.000562}{x} )</td>
<td>( y = 1.93 + \frac{0.000712}{x} )</td>
<td>( y = 2.67 + \frac{0.000261}{x} )</td>
</tr>
<tr>
<td>Regression (Normalized Data)</td>
<td>( y_n = 0.0093 + 0.155 \frac{x_n}{x} )</td>
<td>( y_n = 0.0122 + 0.213 \frac{x_n}{x} )</td>
<td>( y_n = 0.0332 + 0.159 \frac{x_n}{x} )</td>
</tr>
<tr>
<td>( R^2 )</td>
<td>0.079</td>
<td>0.158</td>
<td>0.061</td>
</tr>
<tr>
<td>T-test (slope)</td>
<td>( T = 3.08 )</td>
<td>( T = 9.31 )</td>
<td>( T = 6.13 )</td>
</tr>
<tr>
<td>P-value</td>
<td>( P = 0.003 )</td>
<td>( P-value &lt; 0.001 )</td>
<td>( P-value &lt; 0.001 )</td>
</tr>
</tbody>
</table>

Based on the strong support for the validity of Hypotheses 3.1.1 and 3.2.1 but weak
support for Hypothesis 3.1.2 and 3.2.2 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. [32] that in open source projects, the organizational structure affects the product structure but not vice versa.

5.3.2.3 Hypothesis 3.3

Based on the mirroring hypothesis and Conway’s law, Hypothesis 3.3 in this chapter 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 [34, 47] 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 [48]. 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 [34, 47].

The values of congruence measure for different versions of Drupal are shown in
Table 5.8. 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 [169]. 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 [170] to determine the core and peripheral participants. The core-periphery structure of the Drupal community for version 5 is illustrated in Figure 5.11. The number of participants and the congruence values in the core and the periphery are shown in Table 5.9. 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>
<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 5.11 - Core and peripheral nodes in the community (Version 5)

Table 5.9 - 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. [119] 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 [119] 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 [120] 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 \textit{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 chapter 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 [34, 47] 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 5.12, 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 5.13.
Figure 5.12 - 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 5.13 - Sizes of clusters based on the clustering of different networks

Section 5.4 - Discussion and Future Works

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 chapter, 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 dissertation 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. CHAPTER SIX - BUILDING SURROGATE MODEL TO REPRESENT COMPLEX BIPARTITE NETWORK

Section 6.1 - Motivation of Network Reconstruction and Resizing

Network science is an emerging research area involving the study of the structure and dynamics of complex networks [107]. Networks are represented as sets of nodes and arcs connecting those nodes. They are widely used to represent the topologies of a variety of real world systems such as a) the World Wide Web [105, 108], which is a system of interlinked hypertext documents accessed via the internet; b) metabolic networks [171], which are complex networks of various cellular constituents and reactions through which the processes that generate mass, energy, information transfer and cell fate specification are seamlessly integrated; c) neural networks [172], which are networks of biological neurons; d) author and paper networks [173], which are used to study the structure and evolution of scientific research and collaboration.

The network representations are used extensively in our research. From Chapter 3 to Chapter 5, the network representations are employed to represent product structures, community structures and participation. Both of the product structures and community structures can be represented as single node networks. In the product network, the nodes represent modules or components. The links represent the interfaces or physical connection among modules. In the community network, the nodes represent individuals, while the links represent communications among individuals. The participation, which describes who is working on which module, can be represented as a bipartite network. The Type A nodes represent modules or components, while the Type B nodes represent
individuals. The links between Type A and Type B nodes represent the assignments of tasks for individuals. The analysis of product structures, community structures and participations can be converted into the analysis of complex networks (single node or bipartite).

The analysis of complex networks is increasingly used to understand a range of social, physical, and engineering systems. Sociologists use network analysis to understand the behavior of social systems such as organizations, communities, online social networks, and their communication patterns [161, 174]. Computational biologists use network analysis to study gene regulatory networks, food networks, and neural networks. In the domain of engineering, network scientists analyze systems such as power infrastructure, transportation networks, the world-wide-web (WWW), the Internet, and cell phone communication networks. With the discovery of the common patterns followed by diverse networks [175], some researchers have predicted the emergence of a new science of networks [176, 177]. While sociologists and biologists are primarily focused on analyzing naturally occurring networks, engineers are mainly focused on the design of complex networks. With the growing interest in designing robust and resilient networks, there is an apparent need for developing general techniques to support the design of complex networks.

One of the challenges in the analysis and design of complex networks is their size and complexity. Complex networks range from a few thousand nodes to over a million nodes. For example, the synonyms network in Merriam Webster dictionary, in which two words connect to each other if they appear as synonyms, has 182,853 nodes and 317,658 links [178]. The metabolic network consists of 43 organisms, 18,482 substrates and
The number of routers normally recognized as nodes in the Internet was 88,000 in 1998. By 2008, the nodes increased five times [179]. Performing analysis and design of networks with large numbers of nodes and links is computationally expensive. As an example, the time complexity of insertion sort algorithm, which is commonly used in network analysis is $O(n^2)$ where $n$ is the number of nodes in a network. The time complexity of heap sort algorithm is $O(n \log(n))$. If the time required for analyzing a network with 1000 nodes using insertion sort algorithm is 0.07 seconds [180], the corresponding time grows up to 2083 hours for a network with 1,000,000 nodes [180]. Even with high performance computers, the analysis of the network structures and their evolution takes significant amount of time.

A potential approach to address the challenge of computational cost associated with the analysis and design of complex networks is to replace large-scale complex networks with smaller-sized networks with similar characteristics. The approach is similar to developing simplified models to replace complex physical systems in model-based design. The resized networks are abstractions of the original networks, which can be used for analysis and design exploration if the networks are equivalent in terms of the properties of interest. The question is: How can large complex networks be resized into smaller networks while maintaining similar characteristics?

Section 6.2- Proposed Approach for Bipartite Network Reconstruction and Resizing

6.2.1. Overview of the Network Reconstruction Process

The proposed approach supports the generation of resized networks from complex
bipartite networks. The emphasis is on achieving similar characteristics quantified by the social network metrics. Assume an original bipartite network with two types of nodes – type A and type B. From the bipartite graph, the following characteristics are extracted and used for reconstruction and resizing:

a) number of nodes of type A and type B,

b) number of links connecting nodes of type A with type B, and

c) degree distributions of type A and type B nodes.

Using these features from the bipartite network, the following three steps are followed to reconstruct and resize the network:

**Step 1: Degree distribution reconstruction.** Resizing of the network is achieved by reducing the number of nodes and links. First, the number of type A and type B nodes in the resized network is determined based on the number of nodes in the original network. Then, the nodes of both types are created. Based on the number of links and the degree distributions of both the node types in the original network, link-stubs are attached to type A and type B nodes. Link-stubs determine the number of links originating from different nodes. The outcome of this step is a set of nodes of both types with link-stubs that correspond to the target degree distribution.

**Step 2: Establishment of links between nodes.** In this step, the link-stubs from different node types are connected with each other to create links between nodes of type A and type B, thereby developing a bipartite network. The connection is based on the joint degree distribution between nodes of given degrees.

**Step 3: Deriving networks with uniform node type.** In this final step, the bipartite network is transformed into two networks with uniform node types.
Figure 6.1 displays the approach as a flow-chart. The details of the steps are presented in Sections 6.2.2 through 6.2.4.

\[
\text{Input: Original bipartite network with type A and type B nodes.}
\]

\[
\begin{align*}
\text{Step 1. Degree Distribution Reconstruction} \\
\text{a) Determination of the Target Numbers of Nodes and Links. (Section 3.2.1)} \\
\text{b) Assignment of Link-Stubs to the nodes. (Section 3.2.2)} \\
\end{align*}
\]

\[
\text{Step 2. Establishment of Links Between Nodes in the Bipartite Network} \\
\text{a) Node-based Random Attachment. (Section 3.3.1)} \\
\text{b) Link-Stub based Random Attachment. (Section 3.3.2)} \\
\text{c) Joint Degree Distribution Approach (Section 3.3.3)}
\]

\[
\text{Step 3. Deriving Networks with Uniform Node Types From the Bipartite Network (Section 3.4)}
\]

\[
\text{Output: Weighted Undirected Network for Type A and Type B Nodes.}
\]

**Figure 6.1 - Overview of the approach**

### 6.2.2. Step 1. Degree Distribution Reconstruction

In this step, we start with a bipartite network and a target for resizing the network expressed as a percentage of the size of the original network. The outcome of this step is a set of nodes with link-stubs that satisfies the degree distributions of original bipartite network. In this step, first the target number of nodes and links for the resized network are determined (see Section 6.2.2.1). Then, the distribution of link-stubs for each type is determined using the normalized degree distributions (see Section 6.2.2.2). The link-stubs are assigned to the nodes.
6.2.2.1. Determination of the Target Numbers of Nodes and Links

Assume $N_{A,o}$ and $N_{B,o}$ be the number of nodes of types A and B and $N_{L,o}$ be the number of links in the original bipartite network. The number of nodes and links are related to the degree distributions of both types of nodes. Figure 6.2 illustrates the degree distributions of type A nodes and type B nodes. Degree distribution of type A nodes describes the fraction of nodes of Type A as a function of their degrees. In the case of a bipartite network, the degree of Node A is the number of nodes of Type B connected to it. For example, a point $(x_i, y_i)$ in Figure 6.2(a) implies $y_i$ fraction of nodes of type A have a degree of $x_i$ (i.e., are connected to $x_i$ nodes of type B). Similarly, a point $(p_j, q_j)$ in Figure 6.2 (a) implies that $q_j$ fraction of nodes of Type B have a degree of $p_j$ (i.e., are connected to $p_j$ nodes of Type A). Based on the degree distribution,

$$
\frac{y_i}{N_{A,o}} = \sum_{x_i} \frac{y_i}{N_{A,o}} = \sum_{x_i} \frac{y_i}{N_{A,o}} = \sum_{x_i} \frac{y_i}{N_{A,o}} = \sum_{x_i} \frac{y_i}{N_{A,o}}
$$

![Figure 6.2 - Degree distribution of Type A and Type B nodes](image)

In order to rescale the network by a fraction $r$, the number of nodes and links is scaled by that fraction. These modified numbers of nodes and links are referred to as target values because these numbers are adjusted in the following steps to satisfy the degree distributions.

$$
\frac{y_i}{r N_{A,o}} = \frac{y_i}{N_{A,o}} \quad \frac{y_i}{r N_{A,o}} = \frac{y_i}{N_{A,o}} \quad \frac{y_i}{r N_{A,o}} = \frac{y_i}{N_{A,o}}
$$
6.2.2.2. Assignment of Link-Stubs to the Nodes

Having determined the target number of nodes and links, the next step is to determine the number of links that originate from each node. The knowledge of the degree distribution function is essential but inadequate for determining the number of links originating from nodes. Different networks with similar functions can have different densities of points \((x_i, y_i)\) and \((p_i, q_i)\) for different degrees in the degree distribution plots (Figure 6.2). Some networks may have more points at the low degree side than at the high degree side. Other networks may have greater number of points in the middle as compared to low and high degree sides. Hence, the knowledge of density of points in different parts of the degree distribution plot is also important for determining the number of links originating from nodes.

To account for the variation in density of points on the degree distribution plot, the degree distribution is discretized into a set of intervals and the percentage of points within each interval is calculated. This is illustrated in Figure 6.3, where the degrees are discretized into four intervals \(([0, 5], [5, 10], [10, 15], \text{and} [15, 20])\). 20% of the points on the degree distribution plot lie within the degree interval \([0, 5]\) and 10% lie within the interval \([15, 20]\). The number of intervals can be increased to increase the resolution. The density of points on the degree distribution plots of both node types is a key factor considered for the assignment of link-stubs to nodes.
Based on the information about the number of nodes ($N_{A,t}$ and $N_{B,t}$), the number of links ($N_{L,t}$), and the density of points on degree distribution plots, the objective is to determine a set of points on the degree distribution plot such that the following conditions are met as closely as possible:

$$
\sum_{l} y_{i,t} = N_{A,t} \quad \sum_{j} q_{j,t} = N_{B,t} \quad \sum_{l} x_{i,t} y_{l,t} = N_{L,t} \quad \sum_{j} p_{j,t} q_{j,t} = N_{L,t}
$$

Determining the points $(x_{i,t} \ , \ y_{i,t})$ and $(p_{j,t} \ , \ q_{j,t})$ that satisfy these equations is a discrete variable constraint satisfaction problem (CSP). We convert this constraint satisfaction problem into a single-objective optimization problem as follows:

**Figure 6.3 - The density of points in degree distribution of Nodes A**

Design variables:

$(x_{i,t} \ , \ y_{i,t})$ and $(p_{j,t} \ , \ q_{j,t})$

Subject to:

$$\varepsilon_1 = \sum_{l} y_{i,t} - N_{A,t}$$

$$\varepsilon_2 = \sum_{j} q_{j,t} - N_{B,t}$$

$$\varepsilon_3 = \sum_{l} x_{i,t} y_{l,t} - N_{L,t}$$
Minimize:
\[ z = |\varepsilon_1| + |\varepsilon_2| + |\varepsilon_3| + |\varepsilon_4| \]

We utilize the Simulated Annealing methodology [181] to solve the optimization problem. Simulated Annealing methodology is a flexible optimization method, suited for large-scale combinatorial optimization problems [182] such as the traveling salesman problem [182]. The concept of simulated annealing methodology is derived from statistical mechanics and motivated by an analogy to the behavior of physical systems in the presence of a heat bath [183]. In this approach, the initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution.

The outcomes of the optimization are two series of points on the degree distribution plots of type A nodes and type B nodes of a resized network that closely represents the original bipartite network. Each point \((x_{i,t}, y_{i,t})\) represents \(y_{i,t}\) nodes whose degree is \(x_{i,t}\). So, in the network, \(y_{i,t}\) nodes of Type A are created and \(x_{i,t}\) link-stubs are added to each node. Similarly, \(q_{i,t}\) nodes of Type B are created and \(p_{i,t}\) link-stubs are added to each node. These nodes with link-stubs are then connected in Step 2, discussed next.

6.2.3. Step 2. Establishment of Links Between Nodes In the Bipartite Network

In this step, we begin with the nodes and link-stubs and determine how to link the stubs of node type A with the stubs of node type B in order to generate a bipartite network. Figure 6.4 illustrates the linking of stubs. This linking can be carried out in three
ways. The simplest (and the least accurate) approach is a node-based random attachment process, which is discussed in Section 6.2.3.1. The second approach is link-stub based random attachment process, discussed in Section 6.2.3.2. Finally, the most accurate approach is based on the joint probability distribution, which is presented in Section 6.2.3.3. A comparison of the three approaches for the case study is presented in Section 6.3.4

Figure 6.4 - Combination of type A and type B nodes

6.2.3.1. Node-based Random Attachment

In the node-based random attachment process, pairs of type A and type B nodes are selected randomly. A link-stub from both the selected type A nodes is connected with a link-stub of the selected type B node if the two nodes are not already connected with each other and there are unconnected stubs available. The process is continued until all the link-stubs are connected to form the complete resized bipartite network. Note that the random mapping process is a simple process and does not utilize any other information from the original network.

6.2.3.2. Link-Stub based Random Attachment

A variant of the random attachment process is the link-stub based attachment. In this
case, instead of picking nodes at random, the link stubs from nodes A and B are randomly chosen and a link is established between the corresponding nodes if they are not already connected. The probability of selecting a node of a specific type is equal to the degree of that node divided by the total number of links. For example, the probability of a specific Type A node with degree $d_{i,A}$ being selected is $d_{i,A}/N_{L,o}$. The process induces an element of preferential attachment between nodes because the nodes with greater number of stubs have a higher probability of being chosen. Hence, the process results in greater connections among nodes with higher degrees.

6.2.3.3. Joint Degree Distribution Approach

The two approaches for linking nodes are based entirely on the degree distribution of networks. However, in many complex networks, degree distribution is inadequate to describe the network topology. It has been shown by Newman [184, 185] that many networks show assortative mixing on their degrees, i.e., high-degree nodes tend to link to other high-degree nodes and low-degree nodes tend to connect to low degree nodes. Some networks also show a reverse trend, disassortative mixing, where high degree nodes tend to attach to low degree nodes. For example, social networks show significant assortative mixing whereas technological and biological networks display disassortative mixing [185].

In order to take the assortative and disassortative nature of the networks into account, the third approach for linking nodes is based on the joint degree distribution of the network. The joint degree distribution, $P(k_1, k_2)$, is given by the probability that a randomly selected edge is connected to nodes with degrees $k_1$ and $k_2$. The joint probability function is extracted and utilized in the proposed approach as follows:
a) the nodes of type A and B are sorted based on their degrees;

b) the range of degrees for both types of nodes are independently split into a set of
degree ranges \([d_{l,A}, d_{l+1,A}]\) for nodes A and \([d_{j,B}, d_{j+1,B}]\) for nodes B;

c) For each combination of the degree ranges \([d_{l,A}, d_{l+1,A}]\) and \([d_{j,B}, d_{j+1,B}]\), the fraction
of links, i.e., the ratio of number of links between nodes within that degree range to the
total number of links, is calculated;

d) The number of links in the resized network is divided into each combination of
degree ranges based on the fraction of links within the corresponding degree range in the
original network;

e) Within each combination of degree subsets, the preferential attachment process is
utilized to link the nodes by connecting their link-stubs.

**Figure 6.5 - Division of network and components’ information about links’ combinations**

The approach is illustrated in Figure 6.5 where the range of degrees for both node
types are split into two subsets: \([d_{0,A}, d_{l,A}]\) & \([d_{I,A}, d_{2,A}]\) for node type A, and \([d_{0,B}, d_{l,B}]\) &
\([d_{1,B}, d_{2,B}]\) for node type B. In the figure it is shown that 75% of all the links in the
network are among nodes A with degrees between \([d_{0,A}, d_{l+1,A}]\) and nodes B with degrees
between \([d_{0,B} \ d_{j+1,B}]\). Within each combination (e.g., \([d_{0,A} \ d_{j,A}]\) and \([d_{0,B} \ d_{j,B}]\)), the probability of selecting a node is proportional to the degree of the node itself. Note that the degrees are split into two subsets for illustration only. The resolution of the approach can be increased by increasing the number of degree ranges for each node type.

6.2.4. Step 3. Deriving Networks with Uniform Node Types From the Bipartite Network

In bipartite networks many times the relationships between similar node types are also important. For example, in the author-paper bipartite networks the co-authorship relationship between different authors provides insights about the structure of the scientific community. The co-author relationship can be studied by deriving two weighted undirected networks from the original bipartite network: network of authors and network of papers. In the weighted authors network, a link between authors is established if they share at least one common paper. The weight on the link is set equal to the number of papers commonly authored by the authors. Similarly, papers in the weighted paper network are linked if they share common authors. The weights on the links signify the number of common authors. Figure 6.6 displays the transformation of a bipartite network into two weighted undirected networks with single type of nodes. Similarly, in open-source software development, the bipartite network consisting participants and projects can be used to derive the collaboration between different participants working on common projects. The participant network extracted from the bipartite participant-project network provides deeper understanding of the network characteristics.
The resized bipartite network obtained using the approach proposed in this dissertation should also be able to represent the characteristics of the derived networks. Hence, we also use the characteristics of the derived networks to compare the original network with the resized network.

Section 6.3 - Case Study – Reconstruction and Resizing of Open-Source Software Development Community Network

6.3.1. The Open-Source Software Development Network – Sourceforge.net

In this section we illustrate the approach presented in Section 6.2 using a bipartite network from open-source software development. The data used in this dissertation is obtained from a large open-source software community – SourceForge.net. SourceForge is the world’s largest open-source software development platform, with the largest repository of open-source code and applications available on the Internet [186]. The Sourceforge Research Data Archive (SRDA), located at http://zerlot.cse.nd.edu, is a collection of Open-Source Software (OSS) data and resources [187]. The repository is being used by various researchers from different fields for understanding the structure and dynamics of open source software communities [188]. In this chapter, our aim is to
illustrate the proposed approach for network resizing. We obtain the original bipartite network called "people-project network" by querying the SRDA using the standard SQL language. The two types of nodes in this network are people and projects.

6.3.2. Key Features From Original People-Project Network

After obtaining the original people-project network from the database, the following characteristics of the network are extracted:

a) total number of people: 112969,
b) total number of projects: 63695,
c) total number of links between projects and people: 154887,
d) people and project degree distributions, shown in Figure 6.7,
e) point densities in people distribution and project distribution, shown in Figure 6.8,
f) total number of points in people degree distribution: 24,
g) total number of points in project degree distribution: 86.

These characteristics are used to reconstruct an artificial network using the proposed approach detailed in Section 6.2. All of the data are obtained from SRDA by querying the database in May 2009. The degree distributions are plotted on a log-log scale. The linearity of the degree distributions on the log-log scale indicates the scale-free nature of the network.
6.3.3. Building Surrogate Bipartite networks of Equal-Size: Analysis of Results

Having obtained the key characteristics from the original people-project bipartite network, the three steps in Section 6.2 are followed to build a surrogate bipartite network with the same size as original bipartite network. The first step is the degree distribution reconstruction. The resulting degree distributions for the generated network are compared.
with the degree distributions of the original network in Figure 6.9. As shown in the figure, the people and project distributions of the surrogate network are very close to the distributions of the original network. The similarity between the surrogate degree distribution and original distributions demonstrates the usefulness of the proposed approach.

As mentioned in Section 6.2.3, three different approaches are used to establish links between the nodes with link-stubs: 1) node-based random attachment, 2) link-stub based random attachment, and 3) joint degree distribution approach. These approaches do not affect the resulting degree distributions. However, they influence other network characteristics listed in Chapter 2. We analyze how these approaches affect other network parameters in both bipartite networks and projected unimodal networks after the degree distributions are explicitly matched.

![Comparison of original and reconstructed degree distributions](image)

**Figure 6.9 - Comparison between reconstructed degree distributions and original ones**

Three key features: degree correlation, degree centralization, and density are used to compare the surrogate bipartite networks with the original networks. Five key features: clustering coefficient, diameter, average shortest path, average degree and density are used to compare the surrogate unimodal people-people networks with the original
unimodal people-people network. Figure 6.10 displays a comparison of degree correlation between the original bipartite network and surrogate bipartite networks with the same number of nodes.

**Figure 6.10 - Comparison of degree correlation between original and surrogate bipartite networks of identical size**

It is observed that in the original network, the average degree of neighbors of people increases with the increase in the degree of people. This implies that people with higher degrees normally link with projects with higher degree. However, on the project side, there are no significant differences of average degrees of neighbors among projects with different degrees. In the three surrogate networks, the average degrees of neighbors of both people and project increase with the increase of their degrees. The trend can be attributed to the manner in which the links stubs are connected. For example, the increase in the average degrees of neighbors with increasing node degree is due to the fact that when nodes are chosen randomly, there is a higher chance of higher degree nodes getting
selected earlier. Hence, the nodes with higher degrees are connected with each other first. This results in the positive correlation between the degrees of nodes and their neighbors.

In the joint degree distribution approach, the trend is likely due to the imposition of preferential attachment within each subset (see Figure 6.5). The degree correlation is closer to the actual trend for people nodes implying that preferential attachment is an underlying mechanism for people nodes, which is consistent with the observations in the literature [189, 190].

The degree centralization and density of the bipartite network and the characteristics of the 1-mode people-people projects are compared in Table 6.1. The parameters are obtained by applying the three approaches for the establishment of links. There is close agreement between the density of the bipartite and the surrogate networks. The degree centralization is of the same order of magnitude but the surrogate models have lower degree centralization than the original network. Comparing the 1-mode people network, the original network is a large-scale network with high clustering coefficient (0.785) and low density (0.00021).

Using the node-based random mapping process, the differences between the original people-people network and designed network are significant in terms of the diameter and the average shortest path. The difference in the clustering coefficient is small (0.785 in the original network vs. 0.758 in the designed network), which means the node-based random mapping process represents this characteristic well. The diameter is 15, as compared to the original diameter of 21, which means the size of network becomes smaller than the original one. Since the node-based random mapping process cannot well represent the allocation of links in the network because of lack of link allocation
information, the diameter is smaller than the diameter of the original network. In the node-based random attachment process, the allocation of links becomes homogenous since each pair of nodes has equal opportunity to get connected. However, in the original network, the allocation of links has significant heterogeneity. The average shortest path is 4.66 as compared to 6.62 in the original network, which means the distance between any two persons is a little smaller than the original one. This is consistent with the smaller diameter of the network. The density is 0.00023, which is close to the original one.

Table 6.1 - Comparison of key features of the networks obtained by different linking approaches

<table>
<thead>
<tr>
<th>Network</th>
<th>Original Network</th>
<th>Node-based random attachment</th>
<th>Link-stub based random attachment</th>
<th>Joint degree distribution approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Two subsets</td>
</tr>
<tr>
<td>Bipartite</td>
<td>Degree Centralization</td>
<td>0.0066224</td>
<td>0.0019137</td>
<td>0.0020521</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>2.16E-05</td>
<td>2.29E-05</td>
<td>2.29E-05</td>
</tr>
<tr>
<td>Clustering Coefficient</td>
<td>0.785</td>
<td>0.758</td>
<td>0.786</td>
<td>0.782</td>
</tr>
<tr>
<td>Diameter</td>
<td>21</td>
<td>15</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>Average shortest path</td>
<td>6.62</td>
<td>4.66</td>
<td>5.69</td>
<td>5.49</td>
</tr>
<tr>
<td>Average Degree</td>
<td>19.11</td>
<td>16.12</td>
<td>15.95</td>
<td>16.20</td>
</tr>
<tr>
<td>Density</td>
<td>0.00021</td>
<td>0.00023</td>
<td>0.00023</td>
<td>0.00023</td>
</tr>
</tbody>
</table>

Major improvements are observed in the diameter and the average shortest path by applying the link-stub based random mapping process. The diameter is 17 as compared to 15 in node-based random mapping process. The average shortest path is 5.69, which is closer to the original average shortest path as compared to the node-based random mapping process. These improvements demonstrate that the link-stub based random
mapping better represents the original network than the node-based mapping.

Finally, the joint degree distribution approach is utilized to establish the links. The approach involves discretizing the degree ranges into subsets and applying link-stub based random mapping within corresponding subsets. In Table 6.1, three scenarios are presented. In the first scenario, the degree ranges are divided into two subsets. The results from two subsets are close to link-stub based random mapping process. Hence, in the second scenario, we divide the degree range into four subsets. The results are a slight improvement over the first scenario. In the third scenario, the degree ranges are divided into sixteen subsets. In that scenario, the network characteristics are significantly closer to the original network. The clustering coefficient is 0.798 compared to the original 0.785. The diameter is 20 compared to 21 in the original network. The average shortest path is 6.54 as opposed to 6.62 in the original network. Further increase in the number of subsets within the degree range would increase the fidelity of the network but at the expense of additional computational cost of the linking process.

The degree correlations of original and surrogate bipartite networks are in the same order of magnitude, but do not closely match. The difference highlights the fact that matching the degree distributions does not necessarily imply that all other network characteristics will also match [191]. The proposed method for generating surrogate network is established with the goal of matching degree distributions only. We do not claim that the proposed approach will result in matching all other characteristics at the same time. The comparison is presented to determine which of the characteristics are closer to the original network, and which parameters show departure from the original. In the bipartite networks, the degree correlations of the people nodes are similar but the
degree correlations of the project nodes are different. In the project unimodal networks, some parameters like clustering coefficient, diameter, average shortest path and density are close. Other parameter like average degree is not. The result reiterates the assertions from Anderson et al. [192] and Li et al. [191] that the modeling of degree distributions may not relate to all network parameters and properties. Determining a comprehensive list of network parameters that are similar if the degree distribution of two networks is similar (via rigorous mathematical analysis) is still an open research issue.

In Table 6.1, the average degrees of the surrogate networks show some deviations from the original network. In order to determine the reason for this deviation, the network degree distributions for the people-people networks are plotted in Figure 6.11. It is observed that in the original people-people network, the degree distribution is not a straight line but shows significant scatter. However, the degree distributions of reconstructed and rescaled networks are constructed to closely follow a trend. For the open-source software development data, this trend is given by the power-law relations as shown in Figure 6.11. The variations around these power-law relations are not taken into consideration in the proposed approach. The variations induce the scatter observed in the degree distribution of people-people network. It is observed in Figure 6.11 that for number of people ranging from 100 to 10000, the degree distribution of the reconstructed network matches the original network well since it follows a straight line corresponding to scale-free networks. However, between the range [1 10], the variation in the degrees of the original bipartite network is significant. Since these variations are not taken into consideration, the degree distribution of resized people-people network deviates from the original degree distribution of people-people network. This is a limitation of the proposed
6.3.4. Resizing of the Bipartite Network: Analysis of Results

In this section, the results from the smaller surrogate networks are discussed. For brevity, the comparison of the networks obtained for different resizing levels are presented using only the joint degree distribution approach. The results using node-based random mapping and link-based random mapping are not presented because they are known to be inferior to the joint degree distribution approach. Three surrogate networks are developed with a target reduction to 75%, 50% and 25% size of the original network. The target numbers of nodes (people and projects) for the surrogate networks are shown in Table 6.2. These target numbers of nodes and the degree distributions are used in the optimization problem discussed in Section 6.2.2.2 to obtain the number of nodes, and link-stubs for each node.

The degree distributions for the resized network to 75%, 50%, and 25% of the...
original size are shown in Figure 6.12, Figure 6.13, and Figure 6.14 respectively. Based on these figures, it is observed that the degree distributions of the resized networks faithfully replicate the degree distributions of the original network.

Table 6.2 - Target numbers of nodes and links in the resized network

<table>
<thead>
<tr>
<th>People-Project bipartite network</th>
<th>Original Network</th>
<th>100%</th>
<th>75%</th>
<th>50%</th>
<th>25%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of People</strong></td>
<td><strong>Target</strong></td>
<td>112969</td>
<td>84727</td>
<td>56154</td>
<td>28242</td>
</tr>
<tr>
<td></td>
<td><strong>Obtained</strong></td>
<td>104973</td>
<td>73584</td>
<td>48389</td>
<td>23446</td>
</tr>
<tr>
<td><strong>Number of Projects</strong></td>
<td><strong>Target</strong></td>
<td>63695</td>
<td>47771</td>
<td>31847</td>
<td>15924</td>
</tr>
<tr>
<td></td>
<td><strong>Obtained</strong></td>
<td>56110</td>
<td>41036</td>
<td>27340</td>
<td>13684</td>
</tr>
<tr>
<td><strong>Number of Links</strong></td>
<td><strong>Target</strong></td>
<td>154887</td>
<td>116165</td>
<td>77443</td>
<td>38721</td>
</tr>
<tr>
<td></td>
<td><strong>Obtained</strong></td>
<td>135450</td>
<td>89906</td>
<td>58992</td>
<td>28621</td>
</tr>
</tbody>
</table>

Figure 6.12 - Comparison between reconstructed 75% degree distributions and original ones
Since it was shown in Section 6.3.3 that partitioning the degree range into sixteen components resulted in the networks closest to the original network, the same number of partitions is used for network resizing. Figure 6.15 presents comparison of degree correlation of the resized bipartite network with 75%, 50% and 25% sizes of the original bipartite network. Table 6.3 presents the key features of surrogate bipartite and 1-mode people-people networks from 100% to 25% size using the joint degree distribution approach.
Figure 6.15 - Comparison of Degree Correlation of Bipartite Network with 75%, 50% and 25% Sizes of the Original Bipartite Network using Joint Degree Distribution Approach

Table 6.3 - Key Features of the Smaller Surrogate Networks (100% - 25% size) using Joint Degree Distribution Approach

<table>
<thead>
<tr>
<th>Network</th>
<th>The Original Network</th>
<th>The Resized Network (16 Subsets)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>Bipartite</td>
<td>Degree Centralization</td>
<td>0.0066224</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>2.16E-5</td>
</tr>
<tr>
<td>1-mode People - People</td>
<td>Clustering Coefficient</td>
<td>0.785</td>
</tr>
<tr>
<td></td>
<td>Diameter</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Average shortest path</td>
<td>6.62</td>
</tr>
<tr>
<td></td>
<td>Average Degree</td>
<td>19.11</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.00021</td>
</tr>
</tbody>
</table>

In comparison of bipartite networks, it is observed that the degree correlations of the
surrogate networks are closer to the original network when the network size is reduced. The degree centralization shows a similar trend as observed in the previous section. The reduction of network size increases the degree centralization and brings it closer to the original one. The density of the surrogate networks increases (up to 9.16E-5) as the networks are reduced in size. On comparing the unimodal projected networks, it is observed that the clustering coefficient, diameter and average shortest path do not change significantly and can be accurately modeled using the joint degree distribution approach. The density of the resized network increases as the size of the network is reduced. The density of the 75% resized network is 0.00022, which is close to the density of the original network (0.00021). As the size of the network is reduced to 50% the density increases to 0.00029. For the network resized to 25% of the original size, the density increases to 0.00071. From the mathematical expression of density (see Chapter 2), it is observed that when the number of nodes and the number of links are simultaneously decreased by a factor of $k$, the density increases approximately by the same factor (for large networks). This is because in the proposed approach the target number of nodes and links are both reduced by the same amount (see Section 6.2.2.1). In order to maintain the same density as the network is resized, the target number of links must be reduced by $k^2$. While reducing the number of links by $k^2$ will help in bringing the density of the network closer to the original network, it will have an adverse effect on the clustering coefficient, which is based on the local topology of connections. This highlights the fact that when reducing the size of a network, it is not possible to preserve all the characteristics of a network. Depending on the objectives of the design problem, appropriate network parameters must be chosen and preserved in the surrogate network. Hence, an appropriate
reduction of the number of links has to be chosen depending on the characteristic (degree or clustering coefficient) that is of interest in a given network.

6.3.5. Validation of the Approach for Networks with Different Topologies

In this section, we validate the approach using networks with different topologies. Three bipartite networks with different topologies are generated for validation purposes. Networks I, II, and III have Poisson, exponential, and a scale-free degree distribution. Table 6.4 displays the basic parameters of corresponding networks including the number of type A nodes, the number of type B nodes, the number of links, and the mathematical expressions of corresponding degree distributions on both types of nodes.

Table 6.4 - Basic Parameters of Three Bipartite Networks

<table>
<thead>
<tr>
<th>Network No.</th>
<th>No. of Type A Nodes</th>
<th>No. of Type B Nodes</th>
<th>No. of Links</th>
<th>Degree Distribution of Type A Nodes</th>
<th>Degree Distribution of Type B Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>100</td>
<td>84</td>
<td>635</td>
<td>$\frac{4^4 e^{-4}}{4!} (\lambda = 4)$</td>
<td>$\frac{6^6 e^{-6}}{6!} (\lambda = 6)$</td>
</tr>
<tr>
<td>II</td>
<td>100</td>
<td>114</td>
<td>407</td>
<td>$0.5 e^{-0.5x} (\lambda = 0.5)$</td>
<td>$0.8 e^{-0.8x} (\lambda = 0.8)$</td>
</tr>
<tr>
<td>III</td>
<td>280</td>
<td>221</td>
<td>576</td>
<td>$0.5784 x^{-1.77}$</td>
<td>$0.5354 x^{-1.636}$</td>
</tr>
</tbody>
</table>

Since this section is for validation purposes only, smaller sized networks are generated. Surrogate networks are generated for the three networks using the proposed approach. The degree distributions of the original networks and equal-sized surrogate networks with three different topologies are displayed in Figure 6.16. As shown in the figure, the proposed approach can be used to match the degree distributions networks with different topologies also.
6.4 - Discussion and Future Work

In this chapter, we present an approach for constructing smaller-sized networks to replace large-scale complex bipartite networks for analysis and design. The proposed approach consists of three steps: a) degree distribution reconstruction, b) establishment of links between nodes, and c) deriving networks with uniform node types. Different ways
of linking nodes are presented. The approach is based on the premise that by replicating the degree distribution, it is possible to replicate key characteristics of complex networks. The following metrics are used to compare the resized networks with the original network: clustering coefficient, diameter, average shortest path, average degree, and density. The approach is illustrated using a complex bipartite network of people and projects from open source software development.

The strengths of the proposed approach are that a) it does not require the knowledge of the dynamics of network evolution, b) it is general enough for application to a wide range of network topologies, c) it can be used to scale down complex bipartite networks while preserving their key characteristics. Using this approach, efforts on the analysis and design of networks can benefit from the reduced computational costs associated with smaller scale networks. Hence, it is a step towards addressing the challenges in the field of complex network design, as discussed in Section 6.1.

In this chapter, we evaluate the generated networks based on five metrics only. However, a large number of diverse metrics are used for network analysis and design. Some examples include hierarchy, centralization, and connectivity. A detailed list of such metrics can be found in [193]. Further research is required to investigate the impact of resizing on other metrics. Based on the results presented in this chapter, it is observed that different network characteristics require different ways of resizing the number of nodes and links. This is discussed in the context of density and clustering coefficient in Section 6.3.5.3. We expect that different characteristics can be categorized into classes of metrics that have similar requirements and can be handled together. Further research is needed to categorize all the network characteristics that remain invariant under different
conditions. Finally, the proposed approach is limited in accounting for the variation in
degree distributions, as shown in Figure 6.13. These issues will be investigated in the
future work.
7. CHAPTER SEVEN - FUTURE WORK AND CLOSING

THOUGHTS: ANALYSIS OF OPEN SOURCE HARDWARE

7.1 - Introduction - Open Source Hardware

7.1.1. Definition of Open Source Hardware

While open source software has significant development during the past twenty years, open source hardware (OSH) development has also received significant attention recently. Since OSH development is still at its early stage, the studies and examples are limited. The definition of OSH development processes has been proposed by several researchers. From the Tucson Amateur Packet Radio (TAPR) Open Hardware License [194], the OSH is a term for tangible artifacts, including machines, devices and other physical things, whose design has been released to the public in such a way that anyone can make, modify, distribute and use those things. Wen [195] defines OSH development as a process allowing others to contribute to the development, and manufacture devices themselves, or even modify and create a new device. Seaman [196] describes three requirements for OSH characteristics: 1) interfaces are made public; 2) design is made public; 3) tools used to create the design are free. According to Acosta [197], OSH includes schematics, diagrams and design rules that can be used, studied and modified without restriction and can be copied and redistributed in modified or unmodified form either without restriction, or with minimal restrictions only to ensure that further recipients can do the same. From Wikipedia [198], OSH is defined as "Open source hardware (OSHW) consists of physical artifacts of technology designed and offered in the same manner as free and open source software (FOSS). Open source hardware is part
of the open source culture movement and applies a like concept to a variety of components." A board definition can be proposed as that OSH refers to a paradigm of community-led innovation where masses of individuals self-organize to product hardware or physical products.

7.1.2. Business Models

The business models of OSH are introduced by Menichinelli [199], Pomertantz [200], and Mellis [201]. Defined by McNamara [202], there are three levels of openness: 1) Open Interfaces, 2) Open Design, and 3) Open Implementation. Open interfaces refer to the scenario where the OSH provides interfaces which are documented explicitly so that everyone can design devices to integrate the OSH through these interfaces. A common example is the USB interfaces or series interfaces provided by computers. The documentation is publicly distributed and the interfaces are standardized so that individuals can develop their own devices to connect with these open interfaces. Open design refers to scenarios where the design of OSH is explicitly documented and can be obtained by public. Hence, everyone can access the design documentation and make modification. Examples of open design includes Open Source Car [4], Open Prosthetics [3], and Open CPU [81]. In these projects, the design documentation is publicly distributed online, so that individuals can obtain and modify them. Open implementation refers to the scenario where individuals can implement the OSH based on explicit documentation, which are available in public. At this level, the materials which are necessary to construct the devices are also available. One example is RepRap 3D printer [7].

Ferrera and Tanev [203] identify seven business models for OSH:
1) **Services**: In this model, the services can be offered based on customization of devices and consultation regarding to the usage and development of the OSH project.

2) **Manufacturing of owned or third party open hardware**: In this model, manufacturing can be offered by manufacturing companies for the open hardware.

3) **Manufacturing of proprietary hardware based on open hardware**: In this model, manufacturing companies can sell their proprietary hardware based on OSH assets developed in the OSH project they are participating in.

4) **Dual-licensing**: In this model, companies offer OSH product under either GPL license or a proprietary license.

5) **Proprietary hardware designs based on open hardware**: In this model, the hardware design is under a proprietary license. But the proprietary hardware design is derived from open source.

6) **Hardware tools for OSH**: In this model, companies sell development boards for testing and further development for OSH. Normally, the design of this board is entirely proprietary.

7) **Proprietary software tools for OSH**: In this model, companies sell proprietary software tools such as product management software for OSH.

### 7.1.3. Comparison between Open Source Software and Open Source Hardware

The key difference between open source software (OSS) and open source hardware (OSH) is that OSS is collaborative, while the OSH is derivative [201]. In open source software domain, individuals collaboratively work to develop refined versions of the software. The software management tool called concurrent versioning system [204] is
developed to record all modifications and bug fixes from individuals. Once the core developers believe that the refined version is ready for distribution, people can download this new version from the website. However, the development of OSH is different. In OSH domain, a core version of of the product is proposed and distributed. The improvements are not accumulated into a single new version OSH; instead, it becomes a highly customized product based on independent future design. The fundamental issue, which causes this difference between open source software and OSH is the cost associated with manufacturing [201].

A detailed comparison of product development processes between open source software and OSH can be divided into four phases: 1) design, 2) manufacturing, 3) distribution, and 4) upgrade.

In the design phase, both open source software and OSH can be viewed as information-based product consisting of requirements, functions, and proposed features. The details of design can be documented in files. In the open source software domain, UML [205] files can be used as a standardized language for developers to communicate. In the open source hardware domain, UML can also be applied as shown in recent studies [206]. In the open source software design, the standardized development and management tools including specific programming language, standardized development platform and CVS are well developed. In the OSH design several tools are also created. For example, SolidWorks can be used to develop shapes. AutoCAD can be used to create shape documentation. Ansys can be used to analyze stress and strain. Field-Programming Gate Array (FPGA) is used to develop electric circuits. Hence in the design phase, the OSH design and open source design share common characteristics.
In the manufacturing phase, open source software can be manufactured by compiling original codes. Due to the nature of software, every individual can manufacture the open source software with little cost. Besides, the software products manufactured by different individuals are identical. The manufacturing of OSH requires time, materials, money, and labor work. Once the design documentation is completed, individuals need to make efforts to manufacture the products. There are two ways to manufacture the products: 1) manufacturing by themselves, 2) manufacturing by third party companies. An example of a product that can be used for manufacturing at home is the Reprap 3D printer. Examples of open source products manufactured by third party companies include Arduino [207] and OSCar [4]. In both these approaches, the manufacturing process requires extra time and money. Besides, due to different selected materials, budgets, and manufacturing technologies, the products are different with each other for different individuals. A common solution for the manufacturing of OSH is that the core of the OSH is manufactured by core groups with the same criteria, while the peripheral parts can be manufactured by individual users.

In the distribution phase, OSS and OSH also have significant differences. With the development of information technologies, OSS can be distributed through internet with little cost. Once the new version of the open source software has been developed, it can be immediately distributed through the Internet. However, the distribution of OSH requires transition from one site to another site, which requires extra time and cost. With the development of logistics, the time and cost of the OSH product distributed is reduced. However, compared with the distribution of open source software, it is still a challenge when the OSH product is large and heavy.
In the upgrade phase, there are also significant differences between open source software and hardware. For open source software, upgrade can be done by modifying codes and re-compiling new codes, which has little cost due to the nature of software. For OSH, the upgrade of product requires re-manufacturing, which needs more money. Besides, the upgrade cycle for open source software is short compared with OSH. Once the new codes are well documented and tested, open source software can be upgraded. However, in OSH product, the upgrade involved more manufacturing and distributing time. Table 7.1 illustrates the difference between open source software and OSH.

Table 7.1 - Differences between Open Source Software and OSH

<table>
<thead>
<tr>
<th>Key Differences:</th>
<th>Open Source Software</th>
<th>Open Source Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Phase</td>
<td>Standardized tools, platforms.</td>
<td>Specific pools, platforms</td>
</tr>
<tr>
<td>Manufacturing Phase</td>
<td>Compile codes with little cost</td>
<td>Needs extra time, materials, energy and money</td>
</tr>
<tr>
<td>Distribute Phase</td>
<td>Distribute through internet with little cost</td>
<td>Transit from one site to another, need extra time and cost for logistics</td>
</tr>
<tr>
<td>Upgrade Phase</td>
<td>Re-compile new codes, short update cycles</td>
<td>Needs re-manufacturing and re-distributing, long update cycles.</td>
</tr>
</tbody>
</table>

7.1.4. History of Open Source Hardware Development

According to Acosta [197], it can be argued that the OSH was initiated in the early 1970's with Homebrew Computing Club. Computer engineers within the club shared designs and collaborated for their first personal computer. Based on the statement from Wozniak [208], the Apple I product can also be treated as OSH since the designs were
passed around freely, and the help was provided for building customized systems. At that time, OSH design relied on the ability of users to connect different components. Collaboration emerged at the "product" level because the components of the computers had been well developed as products by companies. From this standpoint, OSH developers collaborated at the "component" level after that. For example, Meade and Conway developed a method to design large scale integrated circuits in 1970s. The development of components, including integrated circuits, main board, and other peripheral computer devices emerged in 1980s. The OSH development at the "component" level began to require manufacturing. Since 1990s, the rapid development of proprietary design tools obstacles the development of OSH since users found difficulties to share knowledge, and design, and collaborate with each other at different platform. Since 2000, the rapid development of open source software products has led the developers to focus on OSH product again. Besides, the standardization of design methodology and tools enables individuals to collaborate for the OSH development. The significant reduction of manufacturing cost and logistics cost due to the globalization enables OSH communities to not only design, but also manufacture and distribution OSH products. Due to the openness of technology information, the design of OSH is not limited in the computer peripheral components. Instead, the OSH development of key components, including CPU and single board computer has emerged. Besides, the OSH framework is no longer limited in the computer hardware. Instead, the development of OSH physical products including camera, 3D printer s emerged. Figure 7.1 displays the development of OSH over different decades.
7.1.5. Challenges in OSH Product Development

Although OSH has significant development from 1970s, there still exist challenges for future development. The challenges can be categorized into seven aspects, namely: 1) manufacturing involvement, 2) development tools, 3) quality control, 4) specifications, and 5) development life cycle.

The first challenge of OSH product development is the manufacturing involvement. Compared with OSS, the OSH development needs financial resources for manufacturing. Hence, the development of OSH requires manufacturing expertise. Besides, the manufacturing involvement of OSH requires physical modifications, which can cause difficulty in upgrading. Finally, challenges also exist in manufacturing technologies. Different technologies are associated with different manufacturing time, cost and quality.

The second challenge of OSH product development is the lack of development tools. In the open source software domain, CVS is developed as a version management tool.
The lack of standardized development tools and management systems requires individuals to spend more time and energy to develop and manage the projects.

The third challenge of OSH product development is the quality control issue. Since manufacturing processes are required for OSH product development, different manufacturing technologies, materials, environment, and processes affect the quality of products. Hence, documentation recording details of manufacturing processes is necessary to maintain high quality. Besides, the quality monitoring and control process is also necessary to maintain the same quality for products manufactured in different location with different technologies.

The fourth challenge of OSH product development is that more specifications are required in development processes. The specifications include three aspects: 1) more specific job requirements. For example, the OSH development includes design in different aspects, such as Digital Filter design, digital system design, analog system, interface design and power supply design. The different design aspects require participants with different skills and experience. 2) OSH product development requires the integration between hardware design and software design. The communication between software developer and hardware developer is important to ensure the success of integration. 3) The intellectual property issues are revealed in the product development when applied to commercial unites in the OSH design.

The fifth challenge is the development cycle for OSH product development. The product development life cycle is longer for OSH compared to open source software development. The reason is the involvement of distribution in the life cycle. Different from open source software, which can be distributed online, the distribution of OSH
requires extra time and cost. Hence, the product life cycle is longer for OSH, which may reduce the upgrade speed. In Figure 7.2 the challenges for OSH development are illustrated.

![Figure 7.2 - Challenges for OSH Product Development](image)

### 7.2 - Example - Arduino

In this section, we select an example – Arduino [207], to illustrate the development of OSH in details. In Section 7.2.1, an introduction of Arduino is presented. In Section 7.2.2, we discuss the challenges we listed in Section 7.1.5.

#### 7.2.1. Introduction

According to the introduction on Arduino homepage, Arduino is an open source electronics prototyping platform based on flexible, easy-to-use hardware and software. The Arduino system is a single-board microcontroller, containing an Atmel AVR processor and on-board inputs/outputs. The product can sense environment from inputs and can affect its surrounding by its outputs (e.g. light, motors, and actuators, etc.). The key idea behind Arduino is to develop an initial standardized board, and then to distribute it to participants for re-design and development. The initial standardized board can be
purchased at an affordable price (25-30 dollars) or built by hand based on the documentation on the Arduino website. The product began in Ivrea, Italy in 2005. The initial idea is to lower the price for prototyping systems. Founders for this project are Massimo Banze and David Cuartielles [207]. The project is rapidly developed over time. According to Menichinelli [199], by 2006, Arduino had sold 5000 units, 30000 in 2007, and 60000 in 2009. Torrone [209] estimated that there are about more than 300000 Arduino units, and half a million people participate in the project design. In 2011, Google has chosen Arduino for the "Android Open Accessory" kit [209].

7.2.2. Solutions for Challenges

To maintain the successful development of the project, the development team has proposed and applied solutions for the challenges listed in Section 7.1.5.

To solve the challenges in manufacturing involvements, they proposed three strategies. The first strategy is to standardize units, which can be purchased at low cost worldwide, or it can even be built by hand. Besides, the board contains standardized components (e.g. AVR processor, memory and input/output). These standardized components are integrated in the board so that the requirement of special manufacturing technologies is reduced. The second strategy is the good design of the board, which is the core for the platform. The board is well designed in such a manner that the platform can perform basic functions and is easy to upgrade for complex application. By designing the board in this manner, the manufacturing involvement can be reduced to a minimum when participants have limited knowledge and only desire basic functions. Besides, the advanced participants can also develop complex application from the board. The third strategy is that when upgrading the system, the cooperating companies will provide
upgraded versions of the board as soon as possible.

The team resolves the second challenge regarding to the development tools based on two methods. The first method is that participants can mainly develop the systems in the component level. At this level, new components are purchased by participants and integrated with the board by standardized interfaces. The second method is to develop a uniform programming language called Arduino programming language. Besides, the Arduino development environment is also developed. The software is free to download with detail tutorials.

In order to solve the third challenge of quality control, the team selects particular companies to manufacture the boards, and then sell them at a low price. The quality of the board, which is the key component of the platform, can be maintained by the selected manufacturing companies. The quality of the further development by participants is determined by individuals.

The team solves the fourth challenge that more specifications are required by preparing the enriched documentation. The enriched documentations support the specifications of developments. For example, the Book named "Practical Arduino - Cool Projects for Open Source Hardware" about the Arduino development is published. Besides, learning examples, libraries, guidelines, manual, curriculum and forums are provided from the Arduino website. To resolve the intellectual property issues, the team provides simplest but complete version of board, which does not have intellectual property problems.

For the fifth challenge, which is the longer product life cycle, the team uses derived based process in the distribution. The development cycles are controlled by individuals
who re-design the board for applications. Besides, they provide three main versions of the board to accelerate the distribution process. For each version, there are four different sizes - mega, uno, mini and mano. The different types of interfaces, such as USB, serial, and blue tooth are also provided.

7.3 - Research Issues

The OSH product development is still at the early stage. Hence, the research issues related to the OSH product development can be divided into five aspects: 1) study of OSH product structure, 2) study of OSH community structure, 3) study of coordination between OSH product and community, 4) development of OSH management tools, and 5) modeling the OSH product development processes and proposing guidelines. In this section, we discuss these five aspects of research issues in detail.

7.3.1. Study of OSH Product Structure

The study of OSH product structures contains six parts: 1) modeling product structures, 2) comparison of product structure among different products, 3) analysis of product structures, 4) analysis of evolution of product structures, 5) measurement of complexity, and 6) features of product structure.

In the first part, the research questions raised as:

a) How to model the OSH product structure?

b) How to define the interfaces?

In the second part the research questions are:

a) Can we compare product structures among different products?

b) How to standardize the strength of interfaces?
c) How to compare different product structures based on standardized strength of interfaces?

In the third part, the research questions are:

a) What is the physical meaning of network metrics when describing OSH product structure?

b) How to explain OSH product structure by proposed measurements, including propagation cost, cluster cost, DOM, DSM, etc?

In the fourth part, the research question is how to identify the consecutive versions of product?

In the fifth part, the research questions are

a) How to identify complexity in design phase, manufacturing phase, distribution phase and upgrading phase respectively?

b) How to quantify and standardize complexity in each phase then propose a comprehensive measurement of complexity of product for OSH processes?

In the last aspect, the research question is: what is the product structures and evolution in the successful OSH product?

In Table 7.2, the research questions in the studies of OSH product structures are summarized.

<table>
<thead>
<tr>
<th>Areas</th>
<th>Research Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Product Structure</td>
<td>-How to model OSH product structure?</td>
</tr>
<tr>
<td></td>
<td>-How to define interfaces?</td>
</tr>
<tr>
<td>Comparison of Product Structures</td>
<td>-Can we compare product structure among different product? If yes, then:</td>
</tr>
<tr>
<td></td>
<td>-How to standardize the strength of interfaces?</td>
</tr>
<tr>
<td></td>
<td>-How to compare product structures based on the standardized interfaces?</td>
</tr>
<tr>
<td>Analysis of Product</td>
<td>-What is the physical meaning of network metrics in OSH</td>
</tr>
</tbody>
</table>
Structure product structure?  
- How to explain OSH product structure by proposed measures, (e.g. propagation cost, clustered cost, DOM, and DSM, etc)?  
- Can we propose more metrics to describe the OSH product structures?

<table>
<thead>
<tr>
<th>Analysis of Evolution of Product Structure</th>
<th>- How to identify the consecutive versions of the product given the derivative nature of OSH?</th>
</tr>
</thead>
</table>
| Measurement of Complexity                 | - How to identify complexity in design phase, manufacturing phase, distribution phase, and upgrading phase, respectively?  
- How to quantify and standardize complexity in each phase, then propose measures of complexity of the product? |
| Features of Product Structure             | - In a successful OSH product, what is the desired product structure?  
- In a successful OSH product, what is the evolution of product emerged? |

7.3.2. Study of OSH Community Structure

The OSH community is modeled in the same way as open source software community by complex networks with nodes and links, where nodes represent individuals and links represent communication. The research questions in the studies of OSH community structure exist in two areas: 1) modeling the community structures and 2) the features of community structures.

In the first area, the research questions are: how to identify the communication among individuals in the OSH process, which have the derived nature? How to determine the strength of communication based on different communication approaches? In the second area, the research questions are: what is the desired community structure for the OSH product development? How is the evolution of community emerged in a successful OSH project? In Table 7.3, the research questions for the studies of OSH community structures are presented.

Table 7.3 - Research Questions for Studies of OSH Community Structures

<table>
<thead>
<tr>
<th>Areas</th>
<th>Research Questions</th>
</tr>
</thead>
</table>

181
7.3.3. Study of Coordination between OSH Product and Community

The studies of coordination between OSH product structure and community structure are focused on five aspects: 1) modeling of coordination, 2) analysis of coordination, 3) comparison of coordination among different products, 4) analysis of evolution of coordination, and 5) the trends of coordination.

In the first aspect, the research question is how to identify which individual works on a specific product component? In the second aspect, the research questions are how to distinct the coordination in each phases? How to capture the coordination trends? How to test the mirroring hypothesis when analyzing the coordination? In the third aspect, the research questions are can we compare the coordination among different products? If we can, then is there any standard measure for coordination for comparison? In the fourth aspect, the research question is how to identify the consecutive versions of coordination? In the fifth aspect, the research questions are: will the mirroring hypothesis hold in a successfully OSH product? What kind of trend is emerged if the mirroring hypothesis is not held? Table 7.4 describes the research questions focused on the study of coordination between OSH product and community.

Table 7.4 - The Research Questions for Studies of Coordination between OSH Product Structure and Community Structure

<table>
<thead>
<tr>
<th>Areas</th>
<th>Research Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling the Community</td>
<td>- How to identify the communication among individuals in OSH processes, which have</td>
</tr>
<tr>
<td>Structures</td>
<td>derived nature?</td>
</tr>
<tr>
<td></td>
<td>- How to determine the strength of communication based on different communication</td>
</tr>
<tr>
<td></td>
<td>approaches?</td>
</tr>
<tr>
<td>The Features of Community</td>
<td>- What is the desired community structure for a successful OSH product?</td>
</tr>
<tr>
<td>Structure</td>
<td>- What is the evolution of community emerged in a successful OSH project?</td>
</tr>
</tbody>
</table>
Modeling of Coordination
- How to identify which individual works on a specific product component?

Analysis of the Coordination
- How to distinct the coordination in each phases?
- How to capture the coordination trends?
- How to test the mirroring hypothesis when analyzing the coordination?

Comparison of Coordination among Different Products
- Can we compare the coordination among different products?
- If we can, then: is there any standardized measures for coordination, which can be used in comparison?

Analysis of Evolution of Coordination
- How to identify the consecutive versions of coordination?

The Result of Coordination
- Will the mirroring hypothesis hold in a successful OSH project?
- What kind of trend is emerged if the mirroring hypothesis is not held?

7.3.4. Development of OSH Management Tools

The studies of the development of OSH management tools are focused on three aspects: 1) product modification management, 2) documentation of communication, and 3) assignment of tasks for individuals.

In the first aspect, the research issue is to develop a comprehensive management tool to record component modifications in the development processes for each phase. In the second aspect, the research issue is to build communication platforms so that the communication among individuals can be recorded. Also, the research question rises as how to record face-to-face communication when individuals are working in the same place? In the third aspect, the research issue is focused on development of management tools to record who works on which components. Table 7.5 illustrates the research issues for the development of OSH management tools.

<table>
<thead>
<tr>
<th>Areas</th>
<th>Research Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Modification Management</td>
<td>- Develop a comprehensive management tool to record component modifications in the development process (included different phase)</td>
</tr>
</tbody>
</table>
7.3.5. **Modeling the Process and Proposing Guidelines**

The research of modeling the OSH product development processes and proposing guidelines is focused on two aspects: 1) simulation of OSH product development processes, 2) proposing the guidelines.

In the first aspect, the research questions are how to build a computational model to simulate the OSH development processes? How to predict the further results of OSH development processes based on a simulation? In the second aspect, the research questions are whether we can propose guidelines to ensure the success of OSH development processes based on the analysis and modeling of the OSH development processes. Can we propose suggestions or advise for OSH communities when they start to design an OSH product? Table 7.6 presents the research question for modeling the OSH processes and proposing the guidelines.

**Table 7.6 - The Research Questions for Modeling**

<table>
<thead>
<tr>
<th>Areas</th>
<th>Research Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation of OSH Development Processes</td>
<td>- How to build a computational model to simulate the OSH development processes?  &lt;br&gt;  - How to predict the further results of OSH development processes based on a simulation?</td>
</tr>
<tr>
<td>Proposing the Guidelines</td>
<td>- Can we propose guidelines to ensure the success of OSH development processes based on the analytical results?  &lt;br&gt;  - Can we propose suggestions or advise for OSH communities when they start to design an OSH product?</td>
</tr>
</tbody>
</table>
7.4. - Closing Thoughts

Research in the field of open source processes is in its early stages. Successful applications of open source require a deeper understanding of the dynamics of participation and resulting product evolution. Three key factors affecting the Open Source Processes are 1) product structure, 2) community structure, and 3) coordination between product and community structure. In this dissertation, we analyze 1) the effect of product structure on the open source processes, 2) the evolutionary characteristics of product structures, and 3) the coordination between product structure and community structure. The analytics and modeling results display proposed for the corresponding research questions. In this dissertation, the RQ1 is answered in Chapter 3. The RQ2 is answered in Chapter 4. The RQ3 is answered in Chapter 5.

From the results we can conclude that:

1) The product structure has effect on the Open Source Processes; the higher DOM of the product, the lower completion time for the Open Source Processes.

2) The evolutionary characteristics of product structures are determined by the node-level mechanisms. By modifying the corresponding node-level mechanisms, different evolutionary patterns emerge.

3) The coordination between product structure and community structure emerges at single node-level, pair node-level and cluster level. The coordination between product structure and community structure follows the mirroring hypothesis proposed by existing research.

The case studies of Open Source Processes are based on Open Source Software Development projects because of their highly development, mature community, and
completeness of documentation. The studies of Open Source Hardware Development are still in the early stage. There are few successful projects, immature communities, and also lack of documentation for rigorous analysis. In Section 7.3, the potential research issues related to Open Source Hardware Development are discussed in detail. The key future research issues can be divided into two categories: 1) how to apply the knowledge obtained from studies of Open Source Software projects to the development of Open Source Hardware processes, and 2) how to study the Open Source Hardware process based on its unique features. The first future research issue is important because 1) the emergence of Open Source Hardware development in recent years, 2) the common characteristics shared by Open Source Software development and Open Source Hardware development, and 3) the benefit of developing successful Open Source Hardware project by using knowledge obtained from software domain. The second future research issue is important because 1) the Open Source Hardware development is based on derived, which is different from Open Source Software development. This unique feature affects all four phases in the development process and 2) the Open Source Hardware development requires physical manufacturing and distribution, which causes additional energy, money and labor.

The first future research issue involves four steps: 1) define the common characteristics between Open Source Software and Hardware processes, 2) obtain the key knowledge of product development in Open Source Software domain, 3) investigate how to design guidelines for Open Source Software development processes based on the obtained knowledge, and 4) investigate how to apply the proposed guideline to the Open Source Hardware domain. By taking these four steps, the common features between Open
Source Software and Hardware development can be well understood. The guidelines of how to develop both Open Source Software and Hardware based on the knowledge of Open Source Software projects can be well defined.

Because the Open Source Hardware process is still in the early stage, the lack of documentation and data prevent us from deeper analysis of Open Source Hardware process. With the growth of successful Open Source Hardware projects, the data can be obtained for analytics and modeling purposes. The second key future research issue can be conducted then. The second key future research issue contains broader research topics. As discussed in Section 7.3, the research topics can be divided into five categories: 1) studies of unique characteristics of product structure, 2) studies of unique characteristics of community structure, 3) studies of unique characteristics of coordination, 4) development of unique tools, and 5) modeling of the process based on the common characteristics of Open Source Software and Hardware processes as well as the unique characteristics of Open Source Hardware processes. By conducting studies in these five categories, the dynamics of Open Source Processes, including the Open Source Software and the Open Source Hardware processes can be well understood. The core knowledge shared between Open Source Software and Hardware processes, as well as the unique knowledge in corresponding processes can be well framed. Detailed guidelines of Open Source Software development and Open Source Hardware development can be proposed and documented. The success of the development of Open Source Software projects and Open Source Hardware projects can be ensured based on the knowledge and guidelines.
8. BIBLIOGRAPHY


Concerns in Open Source Software (QACOS) and Open Source Software and Product Lines (OSSPL).


Advances in Physics, 51(4), pp. 1079-1187.
Physical Review E, 63(6), pp. 066123.
Networks," Proceedings of the National Academy of Sciences of the Unite States 
of America, 97(11), pp. 149.
112. Bianconi, G., and Barabási, A.L., 2000, "Competition and Multiscaling in 
113. Kumar, R., Raghavan, P., Rajagopalan, S., Sivakumar. D., Tomkins, A., Upfal, E., 
2000, Stochastic Models for the Web Graph, 41st Annual Symposium on 
Foundations of Computer Science: Los Alamitos, CA.
Product Modularity and Interorganizational Relations in the Air Conditioning 
117. Hoetker, G., 2006, "Do Modular Products Lead to Modular Organizations?," 
Consequences of Fit and Misfit between Formal and Informal Networks in Multi- 
1356-1370.
Congruence Have an Effect on Software Build Success? A Study of Coordination 
307-324.
120. Bolici, F., Howison, J., and Crowston, K., 2009, "Coordination without 
Discussion? Socio-Technical Congruence and Stigmergy in Free and Open


Qize Le was born in Xiamen, China in 1985. He grew up in Xiamen, China and received his high school education at No. 1 Middle School of Xiamen until 2003. He attended the Xiamen University at Xiamen, China from 2003 to 2007 and earned his Bachelor of Science degree in Automation. He joined the graduate program in Mechanical Engineering at Washington State University, Pullman, WA in 2008. He received his PhD in Mechanical Engineering in 2012. After his PhD, Qize is interest in a position that emphasizes data analytics and mining.