Chapter I Introduction

1.1 Research Motivation

Nowadays, many software development projects focus on customer satisfaction, quick adaptaion to changes, and flexibility. Therefore, software product line development has become popular because it responds well to frequent changes in user requirements. Software product line shares a common set of features and are developed based on the reuse of core assets have been recognised as an important paradigm for software systems engineering. Recently, a large number of software systems are being developed and deployed in this way in order to reduce cost, effort, and time during system development. Various methodologies and approaches have been proposed to support the development of software systems based on software product line development.

Although software product line development is criticized as having difficulties, it has been more popular. Some difficulties are concerned with the

(a) necessity of having a basic understanding of the variability consequences during the different development phases of software products,
(b) necessity of establishing relationships between product members and product line artifacts, and relationships between product members artifacts,
(c) poor support for capturing, designing, and representing requirements at the level of product line and the level of specific product members,
(d) poor support for handling complex relations among product members, and
(e) poor support for maintaining information about the development process.

As the core of software product line disciplines is constructing new product member by composing the existing software product line architecture rather than building from scratch. In particular, the composing or applying the existing software product line architecture is to identify and reuse the existing software artifacts of the product line. This method, software reuse, is a crucial concern in today's world of complex software products and is one of the major goals of software engineering research. It has been widely accepted that software reuse is an effective way to improve

the quality and increase the productivity of software projects, and to reduce the development cost and time [Caldiera, G. and V.R. Basili. 1991][Frakes, W.B. and K. Kang. 2005].

Generally, the software artifacts that can be reused are requirements specifications, designs, source codes, test plans, architectures and documents. Regarding the evolutionary history of a software system, a number of reusable software artifacts are continuous growing. Hence, the software repositories play the central role in software reuse for storing and maintaining these reusable software artifacts. For effective reuse of these artifacts, two essential activities of the software reuse process [Almeida, E.S. etc. 2005], the development for reuse and the development with reuse, have to be adopted. Generally, the basic phases of software reuse process are acquisition, classification, retrieval, understanding or assessment, adaptation, and integration of the software artifacts. The software developers can create new system effectively based on these processes.

This research thus introduces the mining UML-based SRD repository to cover all important modules and the related infrastructure of the software product line systems by analysis of social network. Also, the research presents the supporting in collecting UML-based software artifacts and identifying the potential reusable software artifacts of software product line systems for reuse in the new software systems. The research is aimed to alleviate the difficulties previously mentioned i.e. (b) and (e).

1.2 Problems Statement

The problems tackled in this research are that:

1. The reuse of software artifacts created during software product line development becomes more difficult.

2. Software developers lack of tacit knowledge of identifying software artifacts from software product line repository in practical.

1.3 Research Objective

The objective of this research project is to identify the reusable software artifacts created during software product line development.

1.4 Scope of Work

1. In this research, we set up a team of software developers which includes a software engineer, system analyst, and programmers.
2. Software engineer and system analyst have well-experience in object-oriented analysis and design. Particularly, they design a system based on object-oriented techniques e.g. by applying UML use case diagrams for software analysis.
3. Programmers have well-experience in object-oriented programming, e.g. in java language.
4. In this research, we establish a case study of software development that encompasses two software projects and apply the analysis of social network.
5. Those software projects have some similar and different requirements.

The remainder of this report is organized in five chapters as described below:

Chapter 2 presents the review of the UML-based software artifacts, business process modeling, and social network analysis and the background of software product line.

Chapter 3 describes the research method applied in the research.

Chapter 4 contains a description of the experiments and analyses the experiences on the case study.

Chapter 5 discusses the conclusions and directions for future work.

Chapter II Research Background

This chapter provides the research background including the UML-based software artifacts, business process modeling, and social network analysis.

2.1 Social Network Analysis (SNA)

In this research, the social network and its analysis method, called the social network analysis or SNA, are focused on. The following subsections describe the origin and importance of SNA, the social networks, and the measurements of SNA.

2.1.1 The Origin and Importance of SNA

Social network analysis (SNA) is a research methodology widely involved in the area of anthropology, sociology, economics and medicine lately. It was firstly proposed by the U.S. anthropologist, J.A. Barnes [Barnes, J.A. 1954], in 1954 to analyze the relationships between actors. SNA is based on an assumption of the importance of relationships among actors (or interacting units). The relationships between actors are channels for transfer or flow of resources either material or nonmaterial. SNA provides both a visual and a mathematical analysis of relationships. The three mathematical foundations of network analysis methods are graph theory, statistical and probability theory, and algebraic models. Moreover, SNA is descriptive rather than predictive. The sociological explanation from social relation angle is more convincing than the explanation simply from individual attribute angle.

In the past, the networks available for study were small and few. Currently, huge amount of data on large social networks are available from the Internet or the organizations. The information may come from blogs, knowledge-sharing sites,

collaborative-filtering systems, online gaming, social-networking sites, newsgroups, chat rooms, and so on. Therefore, SNA are utilized in several studies of word-of-mouth marketing [Domingos, P. 2005], movie analysis [Weng, C., W. Chu and J. Wu. 2007], collaboration of authors or research [Rueda, G., etc. 2007][Cheatham, M. and K. Cleereman. 2006][Tang, J. etc. 2007], relationships among websites [Xiu-Min, Y. and F. Zhu-Qing. 2007], software development [Lopez-Fernandez, L. etc. 2004][Ohira, M., T. etc. 2005.][Souza, C. etc. 2005][Madey, G. etc. 2002.], and so on. A set of any interacting units can form a network.

2.1.2 Social Networks

A *social network* is a set of *social entities* connected by a set of *social relationships* [Scott, J. 2000][Wasserman, S. and K. Faust. 1994]. Examples of the social entities (or actors) are people, groups, organizations, web sites, or other information entities, depending on the level or focus of analysis. The social relationships can be the authority relationships (who reports to whom), the actual communication and information exchange (who communicates with whom), the structuring and flow of work (who depends on whom), or the social relationships (who likes whom, who is similar to whom, etc).

The data of both the social entities and the social relationships can be collected from the questionnaires, direct observation, written records, experiments, or derivation. The social entities data can be the interested attributes of entities, e.g. the attributes of people are name, gender, age, etc. In addition, social entities data can be what part or sub-part of a network the social entity falls in, can arise from the position of social entities measured in the network, e.g. closeness centrality score, and can also be information about the relationships between two types of social entities, e.g. affiliation. The relationships data is the interested attributes of relationships, e.g., the attributes of friendship relationship between people are being friend (is friend or is not friend) or the strength of friendship (closed friend, friend, or far friend).

The social networks' type depends on the social entities and social relationships being studied. It can be a *one-mode* (has single kind of entities) or *two-mode* (has two kinds of entities or has single kind of entities and single kind of events) network. Additionally, it may be a *simplex* (has single kind of relationships) or *multiplex* (has multiple kinds of relationships) network. From these properties, it can be divided into two important types of networks: *homogeneous* and *heterogeneous*. The homogeneous network (single-relational network or SRN) has single node types and link types. The heterogeneous network (multiple-relational network or MRN) has multiple node types and/or link types.

Moreover, a social network has a set of relations of ties which can be viewed in two different ways: *ego-centered networks* and *whole network*s. The ego network is a network of an individual focal entity (ego) that consists of such ego and all of its connected entities. The ego-centered network focuses on an individual at the network center whereas the whole network focuses on the whole network but based on some specific criterion of boundaries called subgroups. Subgroups are subsets of actors among them whom there are relatively strong, direct, intense, frequent, or positive ties. Within the subgroups, social structure and the embeddedness of individual can be also analyzed.

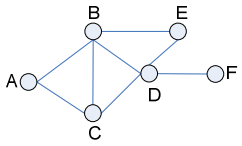
2.1.2.1 Representation of Social Networks

To represent information about the social network data, *graphs* (sometimes called *sociograms*) and *matrices* are useful ways. A graph is used for visually presenting and quantifying important structural properties of social networks. A matrix contains exactly the same information as a graph, but is more useful for computation and computer analysis.

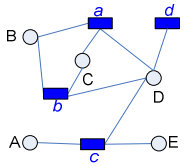
- *Graph*: A graph composes of nodes and edges. The nodes correspond to the social entities and the edges correspond to the social relationships between social entities. The nodes connected to each other by *directed* edges (edges with arrows) or *undirected* edges (edges without arrows) that can have or have not weight associated with

them. The weight in this case may represent strength, cost, probability, or qualitative type of a relationship. A graph can present either one-mode or two-mode and either simplex or multiple network by using different symbols of nodes and edges. Figure 2.1 illustrates examples of sociograms. For a small graph this may be adequate, but usually the networks data are too complex for this relatively approach.

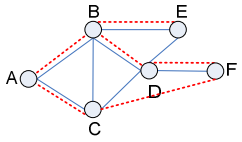
- *Matrix*: Another way to represent and summarize information about social networks is using matrices, especially the *adjacency matrices*. Within an adjacency matrix, a given cell $X_{i,j}$ contains a value of 1 if node *i* is adjacent node *j*, otherwise is 0. The matrix can be either *symmetric* (undirected graph) or *asymmetric* (directed graph). In a symmetric matrix, $X_{i,j}$ is equal to $X_{j,i}$, whereas in an asymmetric matrix, $X_{i,j}$ and $X_{j,i}$ are not necessarily equal. If the network is weighted, it can be represented in a matrix form with weighted values instead of 1s.



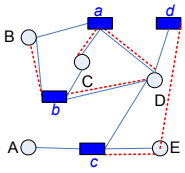
a) One-mode, Simplex



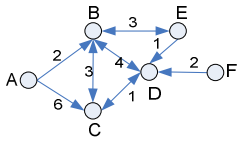
b) Two-mode, Simplex



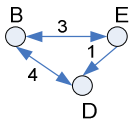
c) One-mode, Multiplex



d) Two-mode, Multiplex



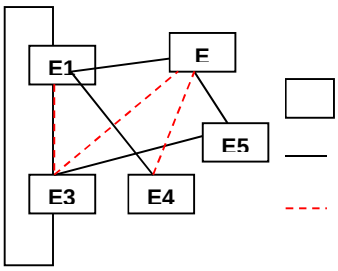
e) One-mode, Simplex, Directed, Weighted



f) Egonet of node E from the whole graph (graph e)

Figure 2.1 Examples of Sociograms

Figure 2.2 illustrates two examples of social networks, i.e., organization network and citation network, represented by both graphs and matrices. The organization network has the employee as the social entity and the co-worker and friendship as the social relationships between employees. Therefore, it is a one-mode and multiplex network. Its graph and matrix representation are shown in Figure 2.2. Generally, one matrix represents a simplex network. For the multiplex networks, a set of adjacency matrices (or slices) is used to represented, one for each relationship. For the citation network, its social entities are author and publication whereas the social relationship is the publish relationship between the author and the publication.



An Adjacency Matrix of Co-worker Relationship

	E1	E2	E3	E4	E5
E1	0	1	0	1	0
E2	1	0	0	0	1
E3	0	0	0	0	1
E4	1	0	0	0	0
E5	0	1	1	0	0

Figure 2.2 Examples of the Representation of Social Networks

2.1.2.2 Fundamental Properties of Social Networks

There are some properties of social networks that are very important in social network analysis. These properties can be considered into two aspects with respective to the concept of graphs. In the first aspect, the edges between a pair of nodes in terms of whether the nodes are adjacent or not are considered. The fundamental properties relate to this aspect are *degree*, and *density*. In the second aspect, two nodes can be linked by indirect routes that pass through the other nodes in the graph. The fundamental properties relate to this aspect are *walk*, *trail*, and *path*. The *length* of a walk, a trail, and a path is simply the number of edges in them respectively. A property that

whether there is a path between a pair of nodes or not, called *reachability*, is also very important. In addition, using the definition of paths the *distance* between any pair of nodes and *diameter* of a graph can be found. Other important property is whether or not a graph is connected. If a graph is connected, all pairs of nodes are reachable. This can be known by considering the *components*. All of these properties and their description are summarized in Table 2.1.

Table 2.1 Fundamental Properties of Social Networks

Properties	Description
Degree	It is the number of directed edges a node has.
Density	It is the proportion of all possible edges that are actually present. It is the ratio of the number of edges present to the maximum possible number of edges.
Walk	It is a sequence of nodes and edges that begins and ends with nodes. It can involve the same nodes or the same edge multiple times.
Trail	It is any walk that includes a given edge no more than once. The same other nodes can be part of a trail multiple times.
Path	It is a walk in which each other node and each other edge may be used at most one time.
Reachability	If there is a path between node n_i and n_j, then n_i and n_j are said to be reachable.
Distance	It is, also called the *geodesic distance*, the length of any shortest path between two nodes. The term *geodesic* means a shortest path between two nodes.
Diameter	It is the length of the largest geodesic between any pair of nodes in a graph.
Component	It is a maximal connected subgraph. If there is only one component in a graph, the graph is connected. If there is more than one component, the graph is disconnected.

Many of the concepts of graph theory have been used as the foundation of many theoretical concepts of social network analysis. To compute the fundamental properties of social networks, the matrix operations such as permutation, transpose, addition, subtraction, multiplication, or powers, are needed.

2.1.3 Measurements of SNA

Since social network data consists of relationships among actors, the presence of relationships has implications for a number of measurement issues. To analyze the social networks, the analysts have to know their purpose of what they are looking for in existing network data. A social network data can be observed at a number of different levels: individual level, group level, and whole network level. The *individual level analysis* concerns the identification of the most important actors in a social network and the measures for individual roles. The *group level analysis* concerns the identification of cohesive substructures or groups of actors within a network. Lastly, the *whole network analysis* concerns measures that focus on entire network. In addition to the aspect of connection, another aspect of research studies is similarity. Table 2.2 summarizes the measurements of social network analysis according to the levels of analysis, the connection of within a network, and the similarity between individual or between subgroups.

Table 2.2 Measurements of Social Network Analysis

	Individual Level	**Groups Level**	**Network Level**
Connection	- Centrality (e.g., degree, closeness, betweenness, etc.); - Prestige (e.g., proximity, rank)	- Density; - Group Centralization; - Group Prestige ; - Cohesive substructure (e.g. cliques, n-clique, n-clan, k-plex, LS sets, Lambda sets, etc.);	- Density; - Average Distance; - Centralization; - Connectedness; - Prestige
Similarity	Similarity (valued relations, binary relations)	Equivalence Classes (e.g. structural, automorphic, regular)	-

2.1.3.1 Individual Level of Analysis

The *individual level analysis* concerns the identification of the most important actors in a social network and also concerns the measures for individual roles. Actors who are the most important or the most prominent are usually located in strategic locations within a network. An individual has power if he can dominate others. For role of a node, it is used to describe the behavior of a node in relationship to its neighbors and to the whole network [Scripps, K. etc. 2007]. For example, roles of a person can be connectors, mavens, leader, ambassador, isolates, etc. There are a number of measures that can be used to determine the roles of nodes in a network. A node that plays role as *centrality* or *authority* in a network is important, in which it has significant impact in a network. The *centrality* measure [D'Souza, D.F. and A.C. Wills. 1999][Sabidusi, G. 1996] can be used to assess a node's popularity or centrality. The *degree*, *closeness*, and *betweenness* are major methods of centrality. The *rank* is a measure of authority within a network. The major methods of rank measures are *HITS* [Kleinberg, J.M. 1999] and *PageRank* [Page, L. etc. 1998.] algorithms.

The similarity of actors in a network has to be measured by indexing the similarities/dissimilarities based on their relations to other actors. The relations between actors can be measured as valued and as binary. The valued relations refer to the strengths or weights of two actors. The common measures of similarity based on valued relations are *Pearson correlation coefficients*, *covariances*, and *cross-products*. An alternative approach to linear correlation is to measure the dissimilarity between the tie profiles of each pair of actors. Examples of the approach are Euclidean, Manhattan, and squared distances. For binary relations, there are several useful measures of tie profile similarities based on the matching idea, e.g. Jaccard, Hamming, Exact, etc.

2.1.3.2 Group Level of Analysis

Groups in a network are those of nodes that are closer to one another than they are to other groups. The group level of analysis involves ways of describing how the

actors in a network are divided into sub-groups on the basis of their patterns of relations with one another. Using the connectivity properties, social network analysis can define various types of substructures or subgroups such as cliques [Luce, R.D. and A.D. Perry. 1949][Festinger, L. 1949], n-cliques [Luce, R.D. 1950][Alba, R.D. 1973], n-clans and n-clubs [Mokken, R.J. 1979], k-plexs [Seidman, S.B. and B.L. Foster. 1978], k-cores [Seidman, S.B. 1983a], LS sets [Seidman, S.B. 1983b], Lambda sets [Borgatti, S.P. etc. 1990], and so on, where a given level of cohesion exists and depends on proximity, frequency, affinity or other properties. The result of group level analysis is a list of the specified cohesive subgroups in the network and the actors who belong to each subgroup. To measure how centralized and prestigious the group of nodes is, one can use a group-level measure of either *group centralization* or *group prestige*.

Moreover, within a network there are a lot of groups of nodes that share similar properties. These roles are defined by regularities in the patterns of relations among nodes, not attributes of the nodes themselves. That is, if two nodes have similar patterns of relations, they fall into the same substructure or *equivalence class* [Hanneman, R.A. and M. Riddle]. There are three types of equivalence; structural, automorphic, and regular, used for defining the equivalences of two nodes.

2.1.3.3 Network Level of Analysis

The measures of the network level analysis can be done with respect to only the connection within a network. The density, the average distance, the centralization, the connectedness, and prestige are measures at this level.

2.1.4 Centrality and Prestige Measures

Identification of the most important or prominent nodes in social networks is one of the primary measures in SNA. The prominent nodes may be more influential or may be more influenced by others. To determine which of the nodes in a network or a group are prominent, one needs to examine not only all relationships made by a node and

all relationships received, but indirect relationships as well. The *centrality* and *prestige* are measures of the prominence of the nodes in a social network [Knoke, D. and R.S. Burt. 1983]. They are first defined at the level of the individual node, then, they can be aggregated over all nodes in a directed network to obtain a group-level measure of either centralization or group prestige.

2.1.4.1 Centrality Measures

The *centrality measure* [D'Souza, D.F. and A.C. Wills. 1999][Sabidusi, G. 1996] was designed for uncovering prominence of nodes that are extensively involved in relationships with other nodes in a network. Three major methods of centrality are degree, closeness, and betweenness centralities [Freeman, L.C. 1979]. The majority of the centrality concepts presented in the following are for undirected networks, where there is no distinction between receiving and sending nodes. They can be extended to directed networks.

- *Degree centrality*. The simplest definition of node centrality is that central nodes must be the most active in the sense that they have the tightest edges to other nodes in the networks. The tighter edges a node has, the more power it has. The degree centrality of node *i*, $C_D(n_i)$, for an undirected network is defined in formula (1) where x_{ij} and x_{ji} is 1 if an edge between node *i* and *j* exists, otherwise is 0. The maximum value of degree for each node in a network is *k-1*. Therefore, a standardization of this measure, as defined in formula (2), is the proportion of nodes that are adjacent to node *i*. The quantity varies between 0 and 1. The higher degree centrality index a node has, the more power it has.

$$C_D(n_i) = \deg(n_i) = \sum_{j=1}^{k} x_{ij} = \sum_{j=1}^{k} x_{ji} \qquad 1)$$

$$C_D^{'}(n_i) = \frac{\deg(n_i)}{k-1} \qquad 2)$$

• *Closeness centrality*. The node which is able to reach other nodes or which is more reachable by other nodes with shorter distance paths is central to quickly interact with all other nodes. The closeness centrality measures [Hartree, D. 1949] how close a node is to all the other nodes in a set of nodes. The centrality is inversely related to distance, that is, if the length of geodesics increase, the centrality of the nodes involved will decrease. Therefore, the closeness centrality index of the particular node *i*, $C_c(n_i)$, is defined as formula (3) where *k* denotes the size and $d(n_i, n_j)$ denotes the length of geodesic linking node *i* and *j*. When the node *i* is adjacent to all other nodes, the $C_C(n_i)$ index reaches $(k\text{-}1)^{-1}$. When the node *i* are not reachable to one or more nodes, the $C_C(n_i)$ index reaches the value of 0 in its limit. Therefore, the $d(n_i, n_j)$ should be infinity if the node *i* is not reachable. The standardization index of $C_C(n_i)$ can be computed with formula (4).

$$C_C(n_i) = \left[\sum_{j=1}^{k} d(n_i, n_j)\right]^{-1} \quad (3)$$

$$C_C^{'}(n_i) = \frac{k-1}{\left[\sum_{j=1}^{k} d(n_i, n_j)\right]} = (k-1)C_C(n_i) \quad (4)$$

• ***Betweenness centrality.*** The nodes that lie in the shortest path between other nodes are relevant as intermediate nodes for interaction with the rest. They can control the interaction between the two nonadjacent nodes. The betweenness centrality is defined as formula (5) where $G_{jg}(n_i)$ denotes the number of geodesics linking two nodes *j* and *g* containing node *i*, G_{jg} denotes the overall number of geodesics linking node *j* and *g*, and $i \neq j \neq g$. A higher betweenness value for a node is an indication of the node's importance. The standardization of this index, $C_B^{'}(n_i)$ in formula (6), is specified like other node centrality indices.

$$C_B(n_i) = \sum_{j<g} \frac{G_{jg}(n_i)}{G_{jg}} \quad (5)$$

$$C_B^{'}(n_i) = C_B(n_i) \Big/ \left[(k-1)(k-2)/2\right] \quad (6)$$

• ***Eigenvector centrality.*** Eigenvector centrality is a measure of the importance of a node in a network by taking into account the entire pattern in the network. Let $G= (V, E)$ be a graph representing a social network that consists of vertices (or nodes) V and edges E. Let A be the adjacency matrix for this graph; a_{ij} = 1 if vertices i and j are connected by an edge and a_{ij}=0 if they are not. More generally, the entries in the matrix A can be real numbers representing the weight or strength of relationships. The principle of eigenvector centrality is the eigenvector of the largest eigenvalue of an adjacency matrix could make a good network centrality measure. Power iteration is one of many eigenvalue algorithms that can be used to find this dominant eigenvector. Equation (7) describes eigenvector centrality X as a matrix equation where λ is the largest eigenvalue of A.

$$\lambda X = AX \qquad 7)$$

The i^{th} component of the related eigenvector gives the centrality score of the i^{th} vertex (or node) in the network.The centrality score of the i^{th} vertex, as shown in equation (8), is proportional to the sum of the centrality score of the vertices to which it is connected with the largest eigenvalue λ where n is the total number of vertices.

$$x_i = \sum_{j=1}^{n} a_{i,j} x_j \;\; / \;\; \lambda \qquad i = 1,2,...,n \qquad 8)$$

In general, vertices with high eigenvector centralities are those which are connected to many other vertices which are, in turn, connected to many others (and so on). Hence if a vertex has a great centrality value, it will be more important.

These discussed measures can be calculated for directed networks. The degree and closeness indices are easily applied to directed networks, while the betweenness indices are not because of their reliance on undirected paths and geodesics. However, centrality indices for directed network generally focus on outbound. To examine the inbound, the prestige indices are generally used both direct and indirect. The example of degree centrality index for directed networks is an outdegree centrality index. The $deg(n_i)$ comes from the summation of all outgoing edges from the node i. For degree

centralization of a directed network, the dominator of this index is $(k-1)^2$. Other indices can be calculated as suggested in undirected networks but with directed edges.

2.1.4.2 Prestige Measures

The prestige measure is used when the network is directed. The prestige of a node increases as the node becomes the object of more edges but not necessarily when the node itself initiates the edges. That is, we must consider at edges directed to a node to measure the node's prestige. The simplest individual-level measure of prestige is *degree prestige* of each node, especially the indegree. Another prestige measure is *proximity prestige*. It measures how proximate the node *i* is to the nodes in its influence domain. The most popular measure of prestige is *rank*. It used to measure authority within a network.

- *Degree Prestige*. The nodes that are prestigious tend to receive many selections. Therefore, the degree prestige of a node *i*, $P_D(n_i)$, is defined as the sum of incoming degrees to the node. Its standardization form, $P_D'(n_i)$, is the proportion of nodes whose choose node *i*. The larger this index, the more prestigious is the node.

$$P_D(n_i) = \deg_I(n_i) = \sum_{j=1}^{k} x_{ij} \qquad 9)$$

$$P_D'(n_i) = \frac{\deg_I(n_i)}{k-1} \qquad 10)$$

- *Proximity Prestige*. Proximity is the closeness that focuses on distances *to* rather than *from* each node. The average distance other nodes are to node *i* is $\sum d(n_j, n_i)/I_i$, where the sum is taken over all node *j* in the influence domain of node *i* and I_i is the number of nodes in the influence domain of node *i*. For group-level measures, the variance of proximity prestige, S_P^2, is used.

$$P_P(n_i) = \frac{I_i/(k-1)}{\sum d(n_j, n_i)/I_i} \qquad 11)$$

$$S_P^2 = \left[\sum_{i=1}^{k} \left(P_P(n_i) - \overline{P_P} \right)^2 \right] \Big/ k \qquad 12)$$

- *Rank Prestige*. There are two popular algorithms of rank, *HITS* [Kleinberg, J.M. 1999] and *PageRank* [Page, L. etc. 1998.]. *HITS (Hyperlink-Induced Topic Search)* is an iterative algorithm based on the linkage of the documents on the web that is used to rate Web pages. It determines two values of a page: *authority* and *hub*. The authority value estimates the value of links to the page while the hub value estimates the value of its links to other pages. They are defined in terms of one another in a mutual recursion. The authority value is computed as the sum of the scaled hub values that point to that page. A hub value is the sum of the scaled authority values of the pages it points to. *PageRank* is an algorithm that assigns a numerical weighting to each element of a hyperlinked set of documents, such as the WWW, with the purpose of measuring its relative importance within the set. The numerical weight that it assigns to any given element E is also called the PageRank of E.

2.2 Software Product Line

Software product line is originally introduced to serve the reuse practice in an organization having a large number of products, which drives issues such as highly expensive, complex, and tedious tasks. The idea of product line was motivated by the need to systematize a number of products more effectively and the fact that these products have a certain set of common and special functionalities. For example, a mobile-phone company has created a mobile-phone family that contains a set of mobile-phones. Some lower end mobile-phones have similar basic functionalities but different hardware capacities to offer competitive price. Region-based mobile-phones are designed for different transmission and signaling standards, depending on regional diversity; thereby, the company provides different functionalities of mobile phones to support different regions.

2.2.1 Activities of Software Product Line Development

The main activities of software product line development i.e.

(a) domain engineering, and

(b) application engineering.

Domain engineering is a systematic process for the creation of the core assets [Northrop, L. M. 2002]. There are three steps for domain engineering:

(i) domain analysis is the process of identifying, collecting, organizing and representing the relevant information in a domain, based upon the study of existing systems and their developing histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain [Kang, K., at el. 1990];

(ii) domain design is the process of developing a design model from the products of domain analysis and the knowledge gained from the study of software requirements or design reuse and generic architectures [Garlan, D. and M. Shaw. 1993.]; and

(iii) domain implementation is the process of identifying reusable components based on the domain model and generic architecture [Clements, P., and L. Northrop. 2004].

Application engineering is a systematic process for the creation of a product member from the software artifacts created during the domain engineering [Northrop, L. M. 2002]. The application engineering process is composed of three steps:

(i) requirements engineering focuses on identifying, collecting, organizing and representing requirements of a product member. The major difference between requirements engineering of an individual product and a product member is that stakeholders do not only focus on the specific product but also on the scope of product family;

(ii) design analysis is to analyse and design the architecture for a product member. Design analysis in application engineering must be consistent with the concept of design analysis in domain engineering; and

(iii) integration and testing is a process of taking reusable components then putting them together to build a complete system, and of testing if the system is working appropriately.

The activities in domain engineering involve the creation of core assets which are expected to be used for all product members. As the activities in application engineering involve the reuse of the software artifacts to create a product member. As shown in Table 2.3, different types of software artifacts are created during the activities of software product line development. For example, reference requirements are created during domain analysis and represented for the requirements of a product line. Software product line architecture is created during domain design and represented for the architecture of the software product line while the architecture of a specific product is generated during design analysis of application engineering. As shown in Table 2.3, software artifacts are generated during the development of software product line systems.

Table 2.3: Software artifacts created during software product line engineering

Software artifact	**Activity**	**Description**
Reference requirements	Domain analysis	Defining the products and their requirements of a family.
Software product line architecture	Domain design	Representing the architecture of software product line.
Reusable software components	Domain implementation	Being integrated with other reusable software components for a particular product member.
Specific-product requirements	Requirements Engineering	Specifying the requirements of a particular product member based on the reference requirements
Specific-product architecture	Design analysis	Representing the architecture of a particular product member based on the software product line architecture
Specific-product configuration and particular product member	Integration and testing	Integrated and configured reusable components that are conducted to be a particular product member

2.3 FRAMEWORK OF SOFTWARE PRODUCT LINE ARTIFACTS

In this section, the framework of software product line artifacts is presented.

2.3.1 Use Case Description

Many approaches proposed to apply use case description in the activities of software product line development. Moreover, we found some approaches that extend use case description for software product line engineering. In [Fantechi, A., at el. 2004], the authors proposed to express the requirements of product members of a product family by extending the use case definition given by Cockburn. The variability is expressed in use cases by using special tags. The tags indicate the requirements of a product family that need to be specialised for a product member. They proposed three types of tag:

(i) alternative tag, which specifies variable requirements with a predefined set of requirement variants;

(ii) parametric tag, which requires specifying of parameters to fill in a requirement for a product member, and

(iii) optional tag, which represents an optional requirement which can be instantiated. In [John, I., and D. Muthig. 2002], the authors extended use case specification by adding constructs for representing variant points for variable requirements and applied the decision model to express the relationships and dependencies between the variable requirements.

2.3.2 UML Modeling

Many existing approaches and methods apply UML modelling in software product line engineering. Some approaches such as [Clauss, M. 2001][Gomaa, H., and M. E. Shin. 2004] are proposed to adapt UML diagrams for modeling software product line.

Gomaa [Gomaa, H., and M. E. Shin. 2004] proposed Product Line UML-based Software engineering (PLUS) by using UML modeling for the development of software product line. The author applied UML diagrams to represent the commonality and variability of software product line. In [Clauss, M. 2001], they use a UML class diagram to represent software product line architecture. They define three types of stereotypes for representing variability in a product family:

(i) variationPoint, which is used for a generalized class;
(ii) variant, which is used for a specialized class; and
(iii) optional, which is used for a class.

They applied two types of relationships to assist representation of variability:

(i) generalization/specialization, which associates between classes typed of variationPoint and variant; and
(ii) association with cardinality 0...1, which associates between any class and a class typed of optional.

Some work proposed the combination of patterns and discriminants to support representing of commonality and variability in software product line architecture. A pattern is represented by class and object diagrams and a discriminant is a feature representing a requirement that differentiates a system from another. They defined three types of discriminant:

(i) single discriminant, which represents a set of mutually exclusive features;
(ii) multiple discriminants, which represent a set of optional features which are not mutually exclusive; and
(iii) option discriminant, which is a single optional feature that may or may not be used.

The single discriminant represents an inheritance hierarchy that consists of a generic class called base class and a set of subclasses called realm. A realm is used to represent variability in a product family. For the single discriminant, a product member can be specified with a subclass in a realm. The multiple discriminants also represent an

inheritance hierarchy that consists of a base class and realm. However, a product member can be specified with one or more subclasses in a realm. The optional discriminant is represented by two classes with a 0..1 association. A product member, which has been specified with a class, may or may not be specified with another class.

2.3.3 Feature Modeling

This technique was initially proposed in FODA [Kang, K., at el. 1990] to assist the activity of domain analysis. Many approaches apply and extend the definition of a feature model to support the development of software product line. However, feature modelling has not yet been standardised comparing with UML modelling which standard has been known. We describe below different aspects of feature modeling technique that are applied in existing approaches i.e.

(i) types of a feature in a feature model,

(ii) notation, and

(iii) relationships between features in a feature model.

In general, there are three types of feature: mandatory feature [Bosch, J. 1998][Clements, P., and L. Northrop. 2004][Griss, M. L., at el. 1998][Kang, K., at el. 1990][Kang, K. C., at el. 1998] is compulsory for product members of a family; optional feature [Bosch, J. 1998][Clements, P., and L. Northrop. 2004][Griss, M. L., at el. 1998][Kang, K., at el. 1990][Kang, K. C., at el. 1998][Svahnberg, M., at el. 2001] may exist in a specific product member or not; and alternative feature [Bosch, J. 1998][Clements, P., and L. Northrop. 2004][Kang, K., at el. 1990][Kang, K. C., at el. 1998] or variant feature [Griss, M. L., at el. 1998], is a possible feature that can be selected for a specific product member. Moreover, [Svahnberg, M., at el. 2001] define an extra type of feature external feature as a requirement not available in the system but need to be satisfied by the external system. A feature may be depicted as a round or a rectangle with its name inside. Some approaches have applied UML notation for expressing features [Griss, M. L., at el. 1998]. Moreover, different types of a feature i.e. mandatory, optional, and alternative are represented in different notations.

Regarding the relationships between features in a feature model, ideally, features are atomic units that can be put together in a product without difficulty. However, features are generally not independent and several types of relations can exist between them. According to [Gibson, P., at el. 1997], feature interaction is defined as a characteristic of a system whose complete behaviour does not satisfy the separate specifications of all its features. The types of relationships express the rules of feature interaction. These relationships are considered for selecting features. They represent which features must be selected together and which features must not.

A feature model becomes a powerful, practical, extensible, and simple technique in domain analysis process. On the other hand, UML diagrams, particularly class diagram, has been applied in domain design process due to its maturity, compliance; and practicality. We investigate how to map a feature model into UML class diagram as the following section.

2.4 Summary

This chapter has provided background of software product line and waterfall approach. The framework of software product line artifacts is presented and mapping different perspectives between feature model and UML diagrams is also discussed.

Chapter III Approach

This chapter presents the Actor-Artifact Network that we apply to create semantic relations between artifacts and actors. Those relations are later used to identify the reusable software artifacts. The next sections we present the characteristics of UML-based software artifacts. Next, the idea of UML-based social networks is described. Lastly, the preparation of UML-based social networks is proposed.

3.1 Actor-Artifact Network

We propose to apply Artifact-Actor-Networks that are an approach to connect social networks and artifact networks, with the goal to create more meaningful semantic connections between artifacts and actors. To connect artifacts and actors under and between each other, semantic relations are required. Every relation in the network connects objects by a semantic context. The Actor-Artifact Networks participate in the life cycle of artifacts as well as significant connections to involved actors will be outlined. Actor-Artifact Networks are consolidating multilayered social networks and artifact networks in an integrated network.

Therefore, we consider the communication and collaboration with each tool (e.g. chat, e-mail and documents) as a single layer of the respective network. We unite these single layers in both social and artifact networks to consolidated networks that contain all actors and artifacts respectively. While in the consolidated social network we can only make statements concerning the relations between actors and in the consolidated artifact network we can only analyse the relations between artifacts, Actor-Artifact Networks (figure 3.1) also contain semantic relations between actors and artifacts. This connection enables a new retrieval method for relevant artifacts and persons in complex data sets. As shown in Figure 3.1, the semantic relations are identified between consolidated social network (lower layer) and consolidated artifact network

(upper layer). With the Actor-Artifact Network with semantic relations, it is easy to relate all artifacts to a person it was involved with. On the other hand one can easily find relevant artifacts and involved persons based on each artifact. As our research scope, we set up a team of software developers which includes a software engineer, system analyst, and programmers. Thus it becomes more facile to determine experts to certain questions and to find problem related data. With the Actor-Artifact Networks, we apply it as a feasible tool to reproduce the dynamic evolution of the networks and to gain new insights in group dynamics and working processes within organizations.

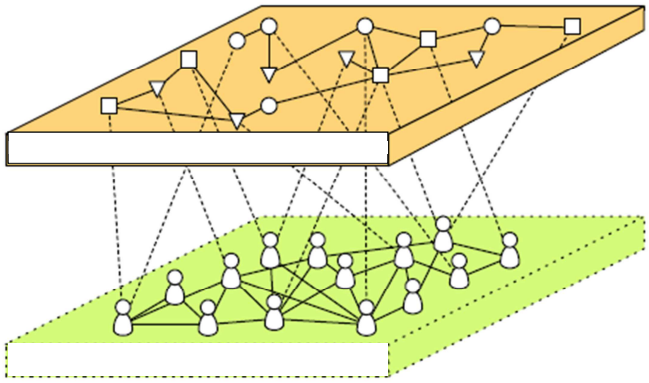


Figure 3.1 Actor-Artifact Network with semantic relations between actors and artifacts

3.2 UML-Based Software Artifacts

In the software product line development taking care of several software product members, each product may be divided into several related systems depending on its size and complexity. Each system is produced differently according to its functional objective. A system which is object-oriented analyzed and designed using UML certainly has many types and levels of diagrams. UML-based software artifacts of use case diagrams can be depicted as shown in Figure 3.2. These diagrams contain elements which consist of model entities and relationships among model entities as summarized in Table 3.1.

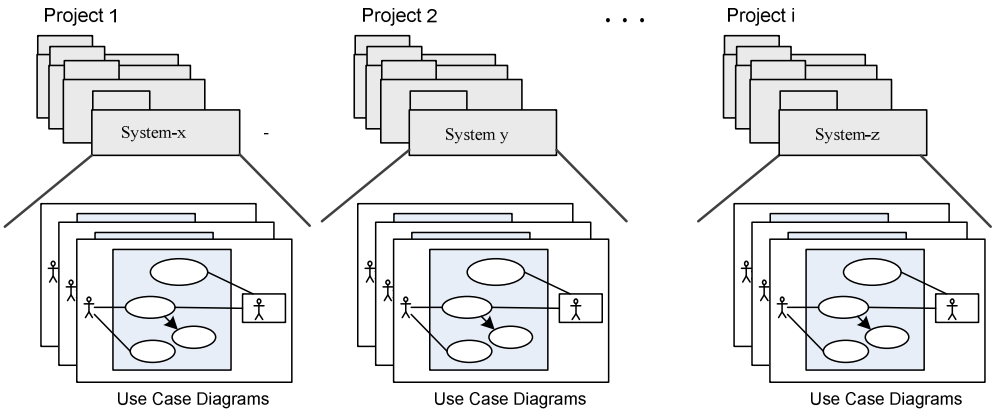


Figure 3.2 A Collection of UML-based SRD Artifacts

Table 3.1 Elements of UML Use Case Diagram

Elements of UML Use Case Diagram	
Model entities	- Use case - Actor - System
Relationships	- Association between use case and actor, - <<extends>> relationship between use cases, - <<uses>> relationship between use cases, - Generalization relationship between use cases or between actors, - Interaction between actor and system

Within the same system or software product member, different levels of UML use case diagrams also have relationships between model entities. That is, the model entities in the top level of a diagram relates to the model entities in the lower levels. Moreover, there are relationships between model entities across different product members. These relationships may happen in case of the model entities of different product members perform same functionality or behavior.

The model entities also have more details data provided in their description template. For example, the description template of a use case [Booch, G. etc. 2005]

typically includes the following data: goal, actors list, constraints a use case operates (pre-condition, invariants, post-conditions), scenarios which are the flow of events that occur during a use case instance, and additional requirements applicable to the use case.

3.3 UML-Based Social Networks

We apply a social network to represent a collaborative communication paths among contributes that are related to software artifacts. In this work , the proposed social network is called the SPL social network for software product line systems. It is a structure where a node represents i) a stakeholder, which can be displayed in different roles e.g. system analyst, designer, programmer, tester, manager etc., and ii) a software artifact, which can be created during software product line process. Those roles are applied and identified each stakeholder when a software system is being developed. In this work, we focus on software requirements artifacts, particularly, use case diagrams. Moreover, we apply a graph to represent the social network in which each directed edge connecting two individual nodes indicates the communication between a pair of nodes. The SPL social network displays only contributors who are involved with the implementation of the same requirements. The SPL social network can capture and explore relevant information to provide awareness on the current activities and stakeholders working on the requirement. In particular, it needs to display i) individual roles, e.g. system analyst, designer, programmer, tester, manager etc., with access to addition information on their activities related to the requirement, ii) the direction of the communication flow; iii) the amount of information flowing from one individual to another, from one individual to software artifact, or from software artifact to an individual; iv) frequency of the information exchange.

The information may be rendered as a visual graph as follows: a) individual roles and software artifact types can be shown using different node shapes and colour; b) directed edges connecting individual nodes indicates the presence and direction of the

communication flow; and c) tag of number can be used to represent the frequency of the information exchange.

The SPL social network can be generated using information from task assignment in a project plan when it is available. If a project plan is not available, a simple diagram can be manually created to indicate the presence of a requirement and the key people assigned to the project. As the requirement is implemented, a contributor is added to the SPL social network based on communication patterns among contributors. If a contributor consults with a programmer about the selected requirement, then the programmer is added to the network and an edge connects the two contributors.

To generate the SPL social network, we can analyze early requirements artifacts and extract key words. We also perform tracking communication patterns among contributors and then updating the SPL social network. For example, one can discover with whom contributors contact when implementing a requirement and maintain a record of expertise regarding this requirement. By observing the SPL social network, a project manager may become more knowledgeable about who is involved in the development of the requirement, and may become more aware of a contributor's current work and skill set. A participant in the project, whether he is a manager or a contributor, can easily see who is working on a particular requirement, and can easily seek information from an appropriate expert if required. Moreover, a manager or participant is encouraged to further explore information on the status and history of the requirement's implementation by adding this information to the SPL social network.

Moreover, the SPL social network can facilitate communication among contributors by providing contact information for each contributor who is working on the same requirement. A contributor can also view the network to observe each team member's involvement in the requirement, and therefore be aware of what the team member is doing. A requirements engineer can use the SPL social network diagram to retrieve a list of everyone involved in the project so that he can easily send messages that are relevant to the entire team, such as requirements change notifications. A project

manager can use the network to identify where there may be a lack of communication between two contributors who should be working together and improve the communication infrastructure to rectify the issues.

Additionally, the data from UML-based software artifacts is heterogeneous since it has multiple types of model entities and multiple types of relationships. The essence of this data depends not only on the features or data of model entities but also on the behavior of relationships among model entities. It is quite possible to think of such data in the same ways as conventional data. That is, the rows are lists of model entities and the columns are data of each model entity. However, various types of the relationships among model entities also needed to be assembled.

The sociological idea about the data in a collection of UML-based SRD artifacts is to represent such data based on the mentioned data and relationships in terms of social networks. These representations are called the *UML-based social networks*. According to the model entities and relationships of use case diagrams as summarized in Table 3.1, the possible social networks of a collection of use case diagrams are listed in Table 3.2.

From the table, the UML-based social networks are defined by considering types of model entities, types of relationships among model entities, and weight and direction of relationships. The networks of single type of model entities are one-mode networks and of two types of model entities are two-mode networks. The networks of single type of relationships are simplex networks and of more than one type of relationships are multiplex networks. The method to represent them practically is graphs or matrices.

Table 3.2 A List of UML-Based Social Networks of Use Case Diagrams

UML-Based Social Networks	Model Entities	Relationships	Mode Type	Simplex or Multiplex	Weighted	Directed
System Network	Systems	Relationships between systems via interaction with the same actor	One-mode	Simplex	Yes (No. of actors)	No
Actor Network	Actors	Relationships between actors (i) association with the same use case, and (ii) generalization relationship	One-mode	Simplex	Yes (No. of use cases) No	No Yes
Use case Network	Use cases	Relationships between use cases: (i) association with the same actor, (ii) uses relationship, and (iii) extends relationship (iv) generalization relationship	One-mode	Multiplex	Yes (No. of actors) No No No	No Yes Yes Yes
System-Actor Network	Systems, Actors	Interactions between systems and actors	Two-mode	Simplex	No	No
System-Use case Network	Use cases, Systems	Containment of use cases in a system	Two-mode	Simplex	No	No
Use case-Actor Network	Use cases, Actors	Associations between actors and use cases	Two-mode	Simplex	No	No

3.4 Preparing UML-Based Social Networks

This section proposes the preparation of UML-based social networks. The input of this preparation is the data in a collection of UML-based software artifacts. We only extract information from the collection of artfifacts. The steps of preparing the UML-based social networks consist of three functions: formulate the initial UML-based social network, find the similar model entities, and adjust the initial social network, as illustrates in Figure 3.3. We described the detail of each functions in the following sections.

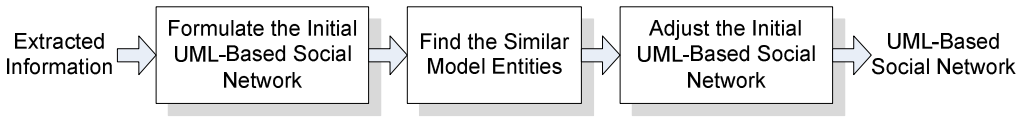


Figure 3.3 The Step of Preparing the UML-based Social Networks

3.4.1 Formulate the Initial UML-Based Social Networks

According to the representation of a social network, a graph and matrices are used to represent the UML-based social networks. In the *graph G= (V, E)*, *V* denotes the set of model entities and *E* denotes the set of edges or relationships among the model entities. The pattern of the initial UML-based social network will be formulated in terms of a graph based on its definition as summarized in Table 3.2.

The following example, shown in Figure 3.3, focuses on formulating the initial use case network. Suppose there are two use case diagrams come from different systems. The abbreviations *ST*, *AT*, and *UC* are system, actor, and use case, respectively. Eight nodes of the corresponding graph are formulated from eight use cases in two use case diagrams. The edges of nodes are formulated from the following relationships: direction relationship, association with the same actor, and containment within the same system. If the type of edge $e_{ij} \in E$ between vertex $v_i \in V$ and vertex $v_j \in V$ is direct relationship, e_{ij} is an unweighted directed edge. If the type of edge e_{ij} is the interaction with the same actors, e_{ij} is a weighted undirected edge where the weight is the number of

common actors associating with two use cases. Lastly, if the type of edge e_{ij} is the containment within the same system, e_{ij} is an unweighted undirected edge.

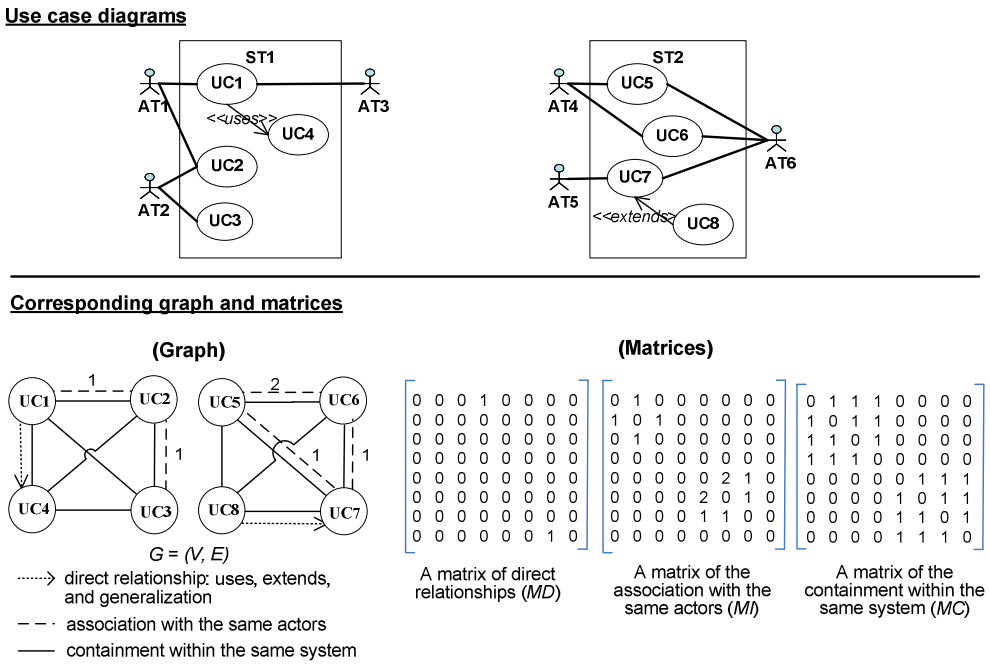


Figure 3.4 Example of Use Case Diagrams with the Corresponding Graph and Matrices

From the Figure 3.4, a use case *UC1* has the <<uses>> relationship to a use case *UC4*, it has direct relationship to *UC4* (dash arrow line). The use case *UC1* associates with an actor *AT1* which is the actor that also associates with a use case *UC2*, therefore, *UC1* and *UC2* linked each other with the weight value 1 (dash line). Moreover *UC1* and *UC2* are contained in the same system or use case diagram, they also link each other with one more relationship (normal line). It is noted that there is no relationship between nodes across system.

Another view of this graph can be represented by three *matrices* according to the number of different types of relationships. In the *matrix of direct relationship MD*, if the use case UC_i uses, extends, or is a part-of the use case UC_j, the element of cell (i, j) in this matrix is 1 otherwise is 0. For the *matrix of the association with the same actors MI*, the number of common actors associating with two use cases UC_i and UC_j is the element of cell (i, j).

Lastly, if use cases UC_i and UC_j are contained in the same system, cell (i, j) in the *matrix of the containment within the same system MC* contains 1, otherwise 0.

3.4.2 Find the Similar Model Entities

From the formulated initial UML-based social networks, it is noted that some model entities may be created in several applications, software projects, or systems. For example, the "*Login*" use case can be produced in more than one system. In this case, they could be considered as a uniqe "*Login*" use case. Moreover, there are some model entities of one system created with different name but perform as same or similar functions in other systems. The similar model entities can be specified by taking the extracted model entity description features into consideration by means of their exact keywords, synonyms of keywords, or pre-defined keywords. Such a model entity is called the *similar model entity* in this research. The overall steps of finding similar model entities are given in the following algorithm.

Algorithm: Find Similar Model Entities
Input: *ME* – a set of model entities consisting of description features D
Output: *S* – a set of the groups of similar model entities
Steps:

1. For each model entity $me_i \in ME$
2. Tokenize words from description feature $D_i \in D$
3. Let T_i = set of tokens
4. Remove stop words from T_i
5. End for
6. For each pair of model entities me_i and me_j, where $i \neq j$
7. Let $s_1 = sim(stem(T_i), stem(T_j))$
8. Let $s_2 = sim(synset(T_i), synset(T_j))$
9. Let $s_3 = sim(pred(T_i), pred(T_j))$
10. Let $simVal = max(s_1, s_2, s_3)$
11. Assign the pair (me_i, me_j) and the similarity value *simVal* to *P*
12. End for
13. Let *mean* be the mean value of *simVal* in P
14. Let *sd* be the standard deviation value of *simVal* in *P*
15. For each data $p_i \in P$
16. If $simVal_i >= (mean + sd)$ then $S_k = \{me_x \in p_i, me_y \in p_i\}$
17. End for
18. Return S

Figure 3.5 Finding the Similar Model Entities Algorithm

First of all, the extracted description features of each model entity are tokenized and then the stop-words are removed. In assigning the similar model entities into the same group, at step 6-12, two model entities are similar in a variety of ways based on the similar real tokens, the similar pre-defined terms of some tokens, or the similar synonyms of tokens. The similar model entities can be found by employing similarity measure. The *dice's coefficient* [Dice, L.R. 1945] in equation (1) which is defined as twice the shared information over the combined set is used in this work.

$$sim(T_i, T_j) = \frac{2 \bullet |T_i \cap T_j|}{|T_i| + |T_j|} \quad (1)$$

For the similarity based on real tokens, every token has to be stemmed. Stemming the token T_i, $stem(T_i)$, can be done by utilizing the Porter's Stemmer algorithm [Porter, M.F. 1980]. For the similarity based on the synonyms of a token, the synonyms of token T_i, $synset(T_i)$, can be generated by utilizing the lexical library of WordNet [Miller, G.A. 1995]. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. The synsets of a token are generated by finding the verb synonyms of the first token and then finding the noun and adjective synonyms of the remainder tokens. For the similarity based on the pre-defined terms of tokens, $pred(T_i)$, some terms have to be defined in the domain dictionary. The domain dictionary is prepared as the backend tool because some terms are similar but are not found with WordNet. Only the maximum value of these three similarity values is selected to compare with the mean plus standard deviation of similarity. If it is greater and equal to, the pair of model entities are defined to have high similarity and then assigned into the same group. Each group of the similar model entities is the representative of similar model entities.

In order to more understand what the algorithm does, Figure 3.6 shows the input and output of steps in the algorithm.

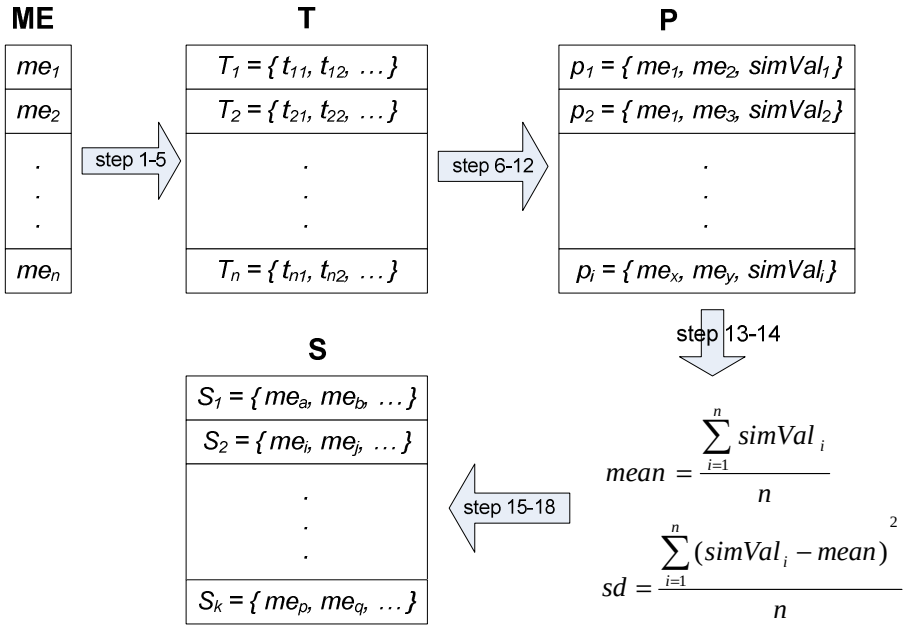


Figure 3.6 The Step Results of Finding the Similar Model Entities

3.4.3 Adjust the Initial UML-Based Social Networks

The set of the groups of similar model entities will be used to adjust the initial UML-based social networks into the new ones. Therefore, the corresponding new graph $G' = (V', E')$ or new matrix M' will be generated. First of all, the set of model entities V' have to be generated. Given the UML-based social network $G = (V, E)$ and the set of the groups of similar model entities $S = \{ S_i \mid S_i$ is a group of similar model entites and $i = 1, 2, \ldots k \}$, the new set of model entities, V', is formed as formula (2). The case of adjusting the nodes in V to V' can be shown in Figure 3.7.

$$V' = \{ x \mid x \in V \wedge x \notin S \} \cup \{ y_i \mid y_i \text{ is the similar node index of } S_i \} \quad (2)$$

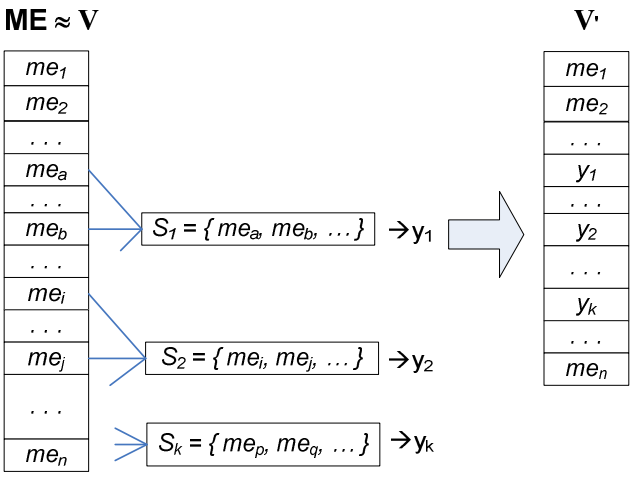


Figure 3.7 The Results of Adjusting Nodes

Afterwards, the set of edges E' will be generated. The first consideration is the edges within E. If nodes me_p and me_q are normal model entities and linked each other, their edges $e_i \in E$ also belong to E'. But in case that they are similar model entities, the edges in E' are different. In order to illustrate the way to adjust the edges of similar model entities me_p and me_q for E', let consider Figure 3.8.

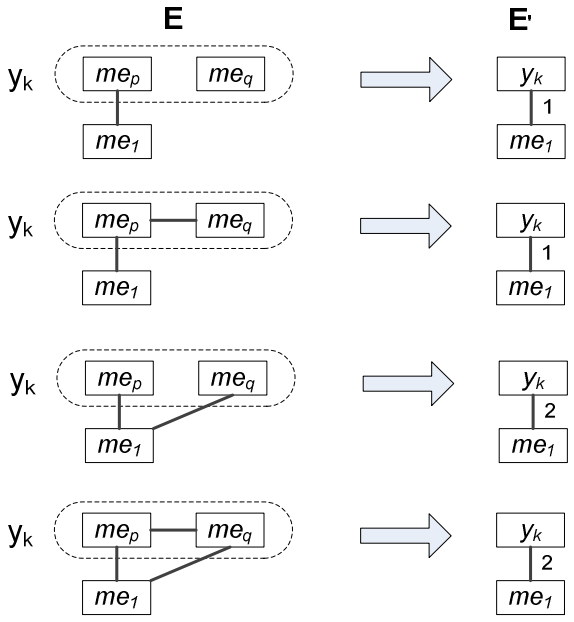


Figure 3.8 The Results of Adjusting Edges of the Similar Model Entities

The second consideration is the edges within E'. Let e_i and e_j be edges linking me_x, $me_y \in V'$, the merged edge $e_k \in E'$ such that e_k represents e_i and e_j is defined. The weight of e_k is defined in equation (3). However, this work uses the min-max normalization to perform a linear transformation on weight value of each relationship type to the range [0, 1] before performing the addition operation.

$$w_k = w_i + w_j \tag{3}$$

Figure 3.9 illustrates the same example as shown in Figure 3.4, but consisting of similar model entities UC_1 and UC_5 ($S_{1\&5}$) and similar use cases UC_3 and UC_7 ($S_{3\&7}$). The similar use cases $S_{1\&5}$ and $S_{3\&7}$ are used as the similar node indices in V' instead. Therefore, the number of nodes in the new corresponding graph is decreased. For the new set of edges, all types of relationships relate with a similar use case will also be related to its corresponding similar nodes in V'. Therefore, there are relationships across different system within the graph. All types of edges among the nodes are merged according to formula (3). Moreover, the merged graph can also be represented with the merged matrix M' as shown step by step in the figure. This adjusted use case network is symmetrized and then be used in the identification process. Each actor is assigned with various roles as described earlier and it depends on the task assignment.

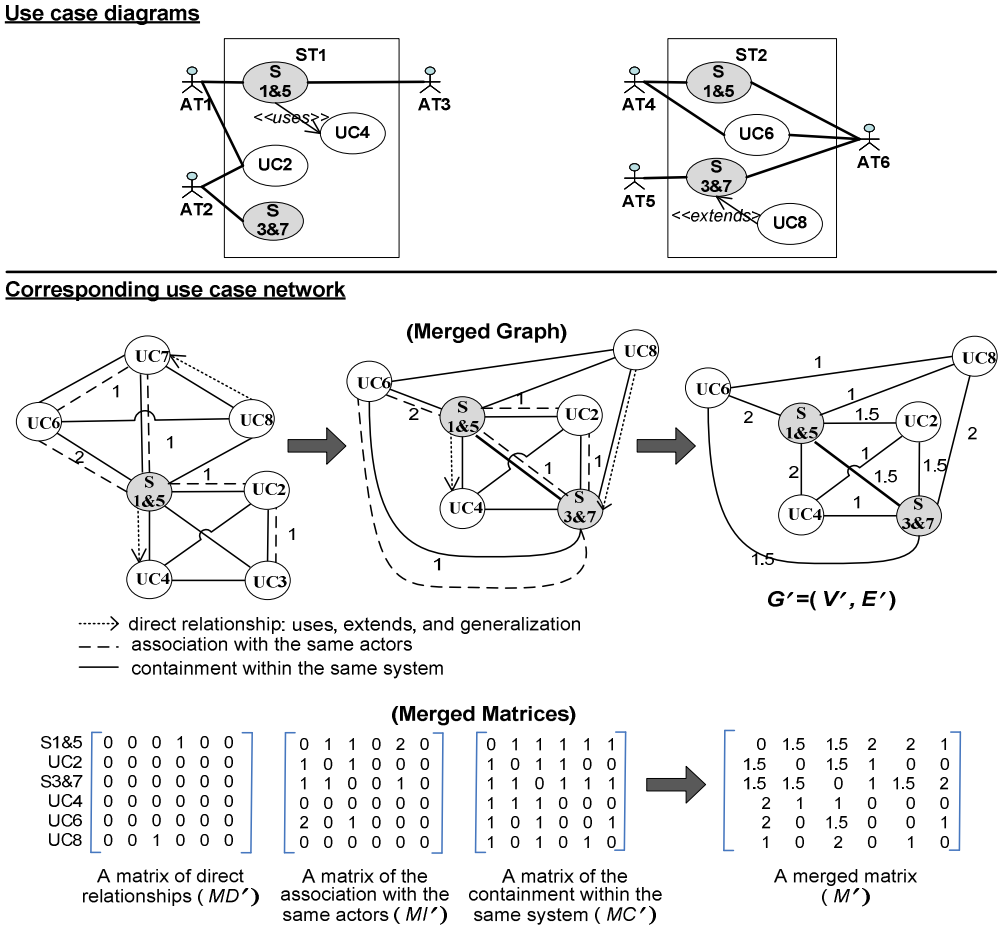


Figure 3.9 Example of Use Case Diagrams and the Corresponding Use Case Network

3.5 Analysis of UML-Based Social Networks

We found that the *frequency* and *eigenvector centrality* of a model entity in a UML-based social network will be used as the indicators to identify whether it is the potential reusable SRD artifact or not. The *frequency* shows that a model entity is frequently created in various systems or not. The frequency in this case considers the name and common function of the model entity. It supports the definition (i) in which the reusable SRD artifacts are frequently created in various system of the software projects. For the *eigenvector centrality*, it is used to measure the importance of a model entity in a social network by taking into account the entire pattern in the network. It support the definition (ii) in which the reusable SRD artifacts are a model entity has low coupling

with other groups and has high cohesion within its group or not. In this section, how to compute the frequency and eigenvector centrality of each model entities in the UML-based social networks will be described.

3.5.1 Find the Frequency of Model Entities

The frequency can be found based on both the set of similar model entities and the remaining model entities. Given a set of groups of similar model entities $S = \{S_1, S_2, \ldots S_n\}$ where n is the total number of groups, S_j is a group of similar model entities, i.e., $S_j = \{me_1, me_2, \ldots me_m\}$, and m is the number of similar model entities of the group j. For any model entities me_i of the UML-based social network, its frequency f_i is

$$f_i = \begin{cases} |S_j|, & if\ me_i \in S_j \\ 1, & otherwise \end{cases} \quad (1)$$

3.5.2 Calculate the Eigenvector Centrality

For calculating the eigenvector centrality, the relationships between model entities are employed. Let M' be a matrix of the adjusted UML-based social network, the eigenvector centrality of a model entity me_i is computed as follow.

$$ev_i = \frac{1}{\lambda}\sum_{j=1}^{n} m_{ij} ev_j \ , \quad i = 1, 2, \ldots, n \quad (2)$$

where λ is the largest eigenvalue of M', $m_{ij} \in M'$, and n is the number of model entities.

3.6 Identifying the Potential Reusable SRD Artifacts

This section aims to identify the potential reusable SRD artifacts based on the analyzed results. It consists of three steps: filtering the model entities, combining the filtered model entities, and choosing the reusable model entities.

3.6.1 Filter the Model Entities

The higher the frequency and eigenvector centrality of a model entity, the better the reusability is. Therefore, filtering the model entities based on high values of frequency and eigenvector centrality are proposed firstly. In this case, the set of frequent model entities and the set of important model entities based on eigenvector centrality will be generated. Given the set of model entities *ME*, the set of frequent model entities Ψ can be generated by considering the model entities that their support satisfies the minimum support *minsup*. The support of a model entity $me_i \in ME$ is the proportion of its frequency f_i to the total number of systems X.

$$\Psi = \{me_i \in ME \mid support(me_i) > minsup\} \quad (3)$$

$$support(me_i) = \frac{f_i}{X} \quad (4)$$

For the set of important model entities based on eigenvector centrality, Ω, it can be generated by considering the model entities that satisfies the following criteria.

$$\Omega = \{me_i \in ME \mid me_i \in top\ \%pv\ of\ A\} \quad (5)$$

where *A* is a set of model entities with positive percent variance of eigenvector centrality and %pv is the specified percent variance of eigenvector centrality.

$$A = \{me_i \in ME \mid pv_i > 0\} \quad (6)$$

$$pv_i = \frac{(ev_i - \mu)}{\mu} * 100 \quad (7)$$

where ev_i is the eigenvector centrality of me_i, and μ is the mean eigenvector centrality. Since both the set of frequent model entities Ψ as well as the set of important model entities based on eigenvector centrality Ω have high reusability, the boundary of reusable model entities can be encircled with both Ψ and Ω. Figure 4.2 illustrates the boundary of reusable model entities, $\Gamma = \Psi \cup \Omega$.

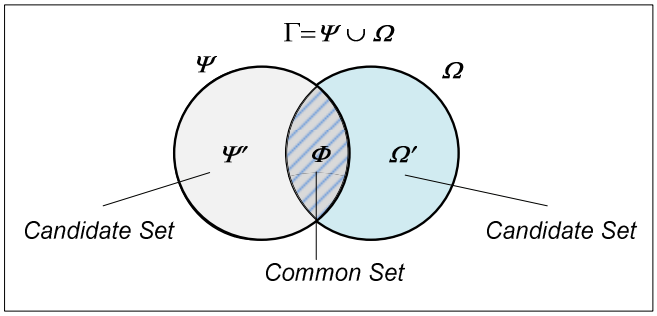


Figure 3.10 The Boundary of Reusable Model Entities

3.6.2 Combine the Filtered Model Entities

However, the reusability can not be decided with either frequency or eigenvector centrality values. Therefore, the combination of these two values are considered. The way to compute the combination of frequency and eigenvector centrality values for the model entity $me_i \in ME$ is

$$c_i = \alpha f_i + \beta ev_i \tag{8}$$

where α and β are the weight values of the frequency and the eigenvector centrality respectively. Their values are in the range [0, 1] and $\alpha = 1-\beta$. Generally, α and β are equal, i.e., their values are 0.5. It means that the frequency and eigenvector centrality are equally important to indicate the reusability. If α is greather than β, the frequency is more important than the eigenvector centrality and vice versa. Now, all the model entities within the boundary have the combined value.

3.6.3 Choose the Reusable Model Entities

One important thing is that some model entities passed the filtering step can belong to both Ψ and Ω. Therefore, they can be said the *common reusable model entities*. Other model entities which belong to only one set of Ψ or Ω can be the *candidate reusable model entities*. From the boundary, let Φ is the set of common reusable model entities and Ψ' and Ω' are two sets of candidate reusable model entities which are complement of Φ based on frequency and eigenvector centrality respectively. It should be noted that Φ will take precedence over every candidate reusable model entities. Therefore, Φ is considered

as the first order set of reusable model entities, whereas Ψ' and Ω' are considered as the second order sets. To choose the best reusable model entities, the first order set with the highest combined value (rank number 1) is considered. The priority sequence of the potential reusable model entities starts from low to high rank numbers. That is, the sequence of ranked reusable model entities starts from the ranked common reusable model entities in Φ followed by other ranked candidate reusable model entities in Ψ' and Ω'. With these results, the software developers can know the important model entities and can use them in the new system development to model the business processes.

3.7 Summary

The characteristics of data in a collection of UML-based SRD artifacts, the idea of UML-based social networks, and the preparation of UML-based social networks are presented in this chapter. The UML-based social networks are prepared by formulating the initial social networks, finding and eliminating the similar use cases in the initial social networks, and then adjusting the initial social networks.

Chapter IV Experiments and Results

This chapter presents the experimental results according to the proposed approach. The dataset used are collected from diverse sources and are separated for using in each experiment. The experiments are divided into three parts in order to reach the objectives.

4.1 The Objectives of Experiments

The objectives of experiments compose of two parts:

1.) To study the possibility of applying the structure of social networks to a collection of UML-based artifacts captured from software process during product line development.

2.) To identify the reusable software artifacts based on the reusability indicators of model entities in the UML-based social networks.

4.2 The Dataset

The UML-based dataset used in this study consists of two domains: hotel management system and jewelry management system.

4.2.1 Dataset 1

Dataset 1 was collected from a case study of software project with the domain of hotel management system. According to the case study, the product line consists of three product members (i) *room service, (ii) restaurant and bar,* and *(iii) ballroom management* systems. The numbers of model entities in the dataset, i.e., sub-systems, use

cases, and actors of the system i, ii and iii are {4, 7, 2}, {6, 2, 12, 6, 2} and {2, 2 ,4} respectively as shown in Table 4.1.

Table 4.1 Information of the dataset 1

Product Member	**(Number of Use Cases)**	**Actors**
(i) Room Service Management	Booking (4) Cleaning Service (7) Laundry Service (2)	*6 Actors*: Receptionist, Manager, Cleaner, Washer, Technician, Customer
(ii) Restaurant and Bar Management	Booking (6) Cleaning Service (2) Catering (12) Stock Logistics (6) Delivery (2)	*7 Actors*: Waiter, Customer, Cashier, Chef, Cooker, Bar Manager, Task Chief, Clerical Staff, Cleaner
(iii) Ballroom Management	Booking (2) Cleaning Service (2) Stage Management (4)	*5 Actors*: Waiter, Customer, Manager, Technician, Musician

4.2.2 Dataset 2

Dataset 2 was collected from a case study of software project with the domain of jewelry management. According to the case study, the product line consists of four product members. The number of product members, actors, and use cases are 4, 10, and 29 respectively as summarized in Table 4.2.

Table 4.2 Information of the dataset 2

Product Member	Use Cases	Actors
(A) Production	*9 Use cases*: Search design 1, Search design 2, Update catalog 1, Update catalog 2, Made-to-order, Produce-in-line, Supply materials 1, Supply materials 2, Supply materials 3	*4 Actors*: Technician, Designer, Supplier, Quality Control Checker
(B) Marketing	*5 Use cases*: Update catalog 1, Update catalog 2, Contact customers 1, Contact customers 2, Contact customers 3	*3 Actors*: Marketer, Customer, Seller
(C) Selling	*8 Use cases*: Selling 1, Selling 2, Selling 3, Search design 1, Search design 2, Credit card management, Cheque management , Cash management	*2 Actors*: Seller, Customer
(D) Accounting	*7 Use cases*: Authenticate, Check clearing, Account Balancing, Accounting 1, Accounting 2, Accounting 3, Reporting	*3 Actors*: Audit, Accountant, Owner, Seller

4.3 Experiment I: Study capturing of a collection of UML-based artifacts from software process during product line development

According to the objective, to study the possibility of applying the structure of social networks to a collection of UML-based artifacts captured from software process during product line development, the hotel management project dataset is manipulated.

4.3.1 Experimental Results

The methodology of this experiment consists of extracting features of the use case models, constructing the UML-based social networks based on the definition described in previous chapter. In this attempt, UML-based social networks corresponding to the use case models, i.e., system network, actor network, use case network, system-use case network, and actor-use case network, are developed by a prototype tool and describes here what we found.

We found that the structure of social networks can be applied to represent the relationships of model entities in UML use case diagrams. The UML-based social networks show the software developers how the dependencies among model entities of UML models are. The influential model entities of each type of UML-based social networks which might be the candidate of reusable SRD artifacts are explored. The software developers can utilize these influential model entities for reuse later.

For a complex and large scale software system, however, visualization may not be easy to view and comprehend for specifying the potential reusable SRD artifacts. Therefore, deeply measure the importance of each model entity in the UML-based social networks in order to find the potential reusable SRD artifacts could be studied.

For the *system network*, we apply a node to represent an area which belongs to the product member, *Room Service Management, Restaurant and Bar Management, and Ballroom Management*. The area of each node shows the number of systems related with the particular system. The percentage of thickness of the edge represents the number of same actors the two product members associating with. Some product members i.e. *room booking* sub-system of room management product member system and the *restaurant management* sub-system of restaurant and bar management product member system have the large areas and the thick link together with the thickness edges. It refers that they have influence on other product members in the product line (Hotel Management System).

For the *actor network*, we apply a node to represent an actor. The area of each node shows the number of actors related with a particular actor. The percentage of thickness of the edge represents the number of same use cases the two actors interacting with. Some actors i.e. manager and receptionist seem to have the largest size of node area and also have the thickness edges to others. It refers that they have influence on other actors in the product line.

For the *use case network,* we apply a node to represent a use case. The area of each node shows the number of use cases related with a particular use case. The percentage of thickness of edge represents the number of same actors the two use cases

interacting with. However, according to the experiment, the size of node area nd the thickness of edges do not show significant different between each other. Therefore, it refers that considering only the number of links can't specify the important ones.

For the *system-use case network*, we found that the network is based on the relationships between product members and use cases are separated into several groups. There is no such significant number to show the difference between them. It refers that the numbers of groups are equal to that of the number of sub-systems.

For the *actor-use case network*, we apply a node to represent a use case. The area of a node represents the numbers of associations. The *Manager*, *Receptionist*, and *Customer* actors seem to be important ones because the numbers of associations with use cases are high. They also have relationships via their associated use cases.

4.4 Experiment II: Identify the reusable software artifacts based on the reusability indicators of model entities in the UML-based social networks

This experiment used the Jewelry Management System dataset to perform the objective (2), that is, to identify the reusable SRD artifacts based on the reusability indicators of model entities in the UML-based social networks and on the definition of reusable SRD artifacts. This experiment focused on the use case models of Jewelry Management System dataset which will be represented the use case network in terms of undirected graph.

4.4.1 Experimental Results

We apply each node to represent a use case and the thickness of each edge to represent the relationship between use cases. Then, the frequency and eigenvector centrality of model entities are computed.

The analysis shows where the frequency and eigenvector centrality values are normalized in range (0, 1] and also sorted in descending order of the normalized

frequency and eigenvector centrality sequentially. From Figure 4.1, a use case with high values of frequency and eigenvector centrality tends to be a good reusable use case candidate. However, many use cases have low values of eigenvector centrality even though their frequencies are high, and vice versa. Therefore, the limitation of high values of frequencies and eigenvector centralities is necessary for identifying the potential reusable use cases. In Figure, we represent the potential use cases are reusable in order significant factors and renamed them in order.

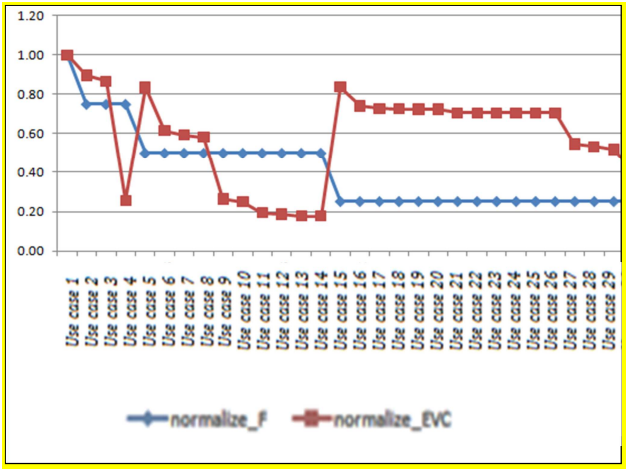
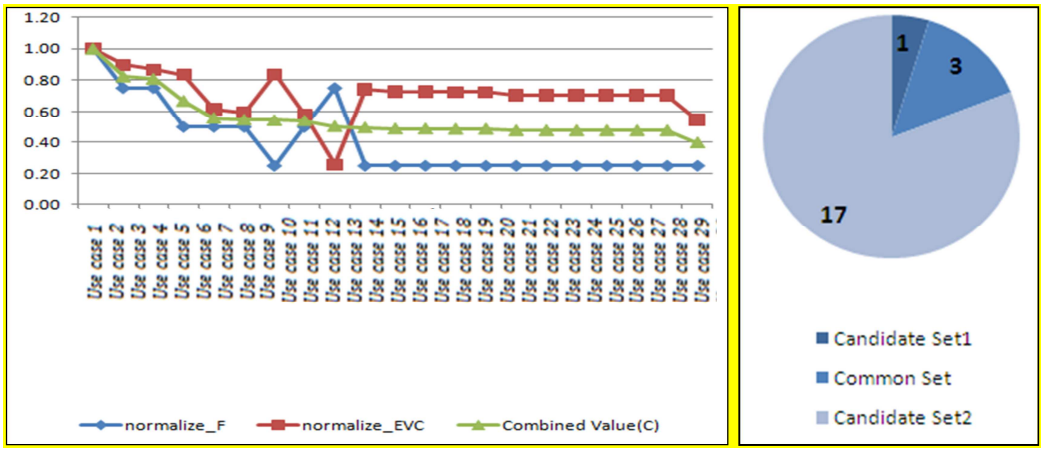


Figure 4.1 The Comparison of Analysis Results of Use Cases in the Jewelery Management System

In the step of filtering in order to generate the set of frequent use cases and the set of important use cases based on eigenvector centralities, 50% of *min_supp* and 95% of *pv* are specified Then, these two filtered sets of use cases will be used to compute the combination values. Figure 4.2(a) depicts all values of use cases sorted in descending order of the combined values which are computed with the equal weight values of α and β, 0.5. The number of reusable use cases in each set of the boundary is illustrated in Figure 4.3(b).



(a) Three values of reusable use cases (b) Boundary

Figure 4.2 Reusable Use Cases of the Jewelry Management System

Figure 4.2(a) shows the list of filtered use cases based on three values: frequency, eigenvector centrality and combined value. The order of the combined values of use cases depicts the important level of the filtered use cases. That is, a use case with the highest combined value is the best candidate of reusable use case. For Figure 4.2(b), the list of filtered use cases is divided into three sets, i.e., the common set of use cases (Φ) has the first priority and the candidate sets (Ψ' and Ω'), according to the proposed identification method. The proportion of common reusable use cases is low in comparison with the candidate reusable use cases because of the diversity of producing use case diagrams in each system.

In the step of choosing the potential reusable use cases from the boundary, the common set of use cases (Φ) has the first priority and the candidate sets (Ψ' and Ω') have the second priority. Their rank number comes from the combined value. Table 4.3 illustrates a list of first ten reusable use cases including the analysis results, the combined values, the rank numbers, and the chosen set.

Table 4.3 A List of the First Ten Reusable Use Cases of the Jewelry Management System

Reusable use case	Frequency (F)	Eivenvector Centrality (EVC)	Combined Value (C)	Rank no.	Chosen Set
1. Use case 1	4	39.86	1.0000	1	common(**Φ**)
2. Use case 2	3	35.76	0.8235	2	common(**Φ**)
3. Use case 3	3	34.56	0.8084	3	common(**Φ**)
4. Use case 4	2	33.31	0.6678	4	candidate (**Ω′**)
5. Use case 5	2	24.51	0.5575	5	candidate (**Ω′**)
6. Use case 6	2	23.68	0.5470	6	candidate (**Ω′**)
7. Use case 7	1	33.40	0.5439	7	candidate (**Ω′**)
8. Use case 8	2	23.17	0.5406	8	candidate (**Ω′**)
9. Use case 9	3	10.25	0.5036	9	candidate (**Ψ′**)
10. Use case 10	1	29.55	0.4957	10	candidate (**Ω′**)

In general, software developers searching for reusable components use different types of knowledge to guide their search. The use case from Table 4.3, for example, has the first rank of reusability according to the experimental results. The software developers can develop their new software systems by considering it as the first priority.

Based on the experimental results, the number of use cases in each set is summarized in Table 4.4. A set of 29 use cases consisting of 9 use cases (those with the high scores of combined values with higher than 0.5) identified as reusable use cases and 10 use cases (those with the low scores of combined values with lower than 0.5) identified as non-reusable use cases.

Table 4.4 Set of Use Cases in the Jewelry Management System

Set of use cases	#No. of use cases
Original use cases	29
Similar use cases	13
Use case network	19
Frequent use cases	4
Important use cases based on eigenvector centrality	10
Reusable use cases in the boundary	11
Percentage of the identified potential reusable use cases = 30.88%	

The percentage of the identified reusable use cases is 31.034%. The use cases in this set will be evaluated by evaluators later whether they are reusable or not. In the other word, the identified non-reusable use cases have to be evaluated whether they are non-reusable or not in the evaluators point of views. It is expected that these use cases is beneficial for the software developers to reuse in business process modeling in their new software projects. The result shows that the reusable use cases can be automatically identified based on our approach.

4.5 Summary

This chapter presents the experiments firstly in order to apply the structure of social networks to a collection of UML use case diagrams and to observe the dependency patterns among use cases in the use case social networks. The results reveal the dependencies among use cases. Moreover, the important use cases can be found. The second experiment's goal is to identify the reusable use cases based on the proposed mining process or the identification method. From the experiment, each use case will be analyzed and filtered. The set of filtered use cases can be considered as the candidates for reuse.

Chapter V Conclusions and Future Work

This chapter provides the conclusions, some useful suggestions for future study and, future work of this research. Section 5.1 presents the overall conclusions. The future work are described in Section 5.2.

5.1 Conclusions

The objectives of this research is to propose the social network analysis for mining of the potential reusable software artifacts created during software product line in order to assist the software developers in identifying the potential reusable software artifacts. The research motivation comes from the difficulty in finding, understanding, and choosing the reusable software artifacts of software developers from a large size and complexity of software repository created during softwar product line development. The existing identification methodologies for reusable software artifacts are not automatic, costly, and remain unexplored. Most of commercial software repositories support the software developers in collecting, locating, retrieving, maintaining, and monitoring the reusable software artifacts, but there is a lack of tools that assists the software developers in identifying the potential reusable software artifacts.

This research focuses on analysis of the social network technique to identify the potential reusable software artifacts in software product line repository. We apply a graph to show the social network which relate UML model elements created from the software artifacts, particularly UML use case diagrams. The mining process of reusable software artifacts was proposed. With the current version of the tool, we have performed two experiments to achieve the objectives: i) to study the possibility of applying the structure of social networks to a collection of UML-based artifacts captured from software process during product line development; and ii) to identify the reusable software artifacts based on the reusability indicators of model entities in the UML-based social networks.

The potential reusable use cases identified based on the approach are beneficial to software developers. The software development organizations can apply this outcomes and their approach to other types of software artifacts. In particular, this research helps promote the idea of software reuse of software product line development.

5.2 Future Work

The following issues are interesting directions for future work:

1. Automatic process

   At present, the model is applied with using several software tools, depending on its availability. Still, the proposed approach has been performed in a semi-automatic way. More specifically, some activities are done by applying with software tools and some are manually performed. The automatic process is expected to support the activities of approach. Moreover, the prototype tool needs to fully support the visualization of social networks in order to facilitate and automatically identify potential reusable software artifacts to new software projects..

2. Extension to small and medium-sized software projects

   The techniques and approaches for software product line development should be further extended to allow establishing software product line for small- and medium- sized projects. In the future work, we plan to gather more data from the projects in order to develop statistic evaluation of comparison between small or medium-sized and large-sized software projects.

   Moreover, for further work, multiple software projects and different application domains will be included in the software repository to have more information to support in identifying the potential reusable software artifacts. Other types of UML diagrams such as class and sequence diagrams can be extended and considered as the software artifacts. The preparation of UML-based social networks, and the mining process for identifying the potential reusable software artifacts for these types of diagrams still be the same.

References

Alba, R.D. (1973). A Graph-Theoretic Definition of a Sociometric Clique. *Journal of Mathematical Sociology*, Vol. 3, pp.113-126.

Almeida, E.S., A. Alvaro, D. Lucredio, V.C. Garcia, and S.R. de Lemos Meira. (2005). *A Survey on Software Reuse Processes,* pp. 66-71.

Barnes, J.A. (1954). Class and Committees in a Norwegian Island Parish. *Human Relations*, Vol. 7, pp. 39-58.

Booch, G., J. Rumbaugh, and I. Jacobson. (2005). *The Unified Modeling Language User Guide*, 2nd Ed., Addison-Wesley.

Borgatti, S.P., M.G. Everett and P.R. Shirey. (1990). LS Sets, Lambda Sets, and Other Cohesive Subsets. *Social Networks*, Vol. 12, pp. 337-358.

Bosch, J. (1998). *Product-Line Architectures in Industry: A Case Study*. Pages 544 - 554. The 21st International Conference on Software Engineering. IEEE, Los Angeles, USA.

Caldiera, G. and V.R. Basili. (1991). *Identifying and Qualifying Reusable Software Components*. IEEE Computer, Vol. 24, No. 2, pp. 61-70.

Chang, C. and C. Liu. (1995). A Hybrid Approach to Object Library Classification and Retrieval. *Proc. IEEE Computer Software and Applications", COMPSAC-95*, pp. 278-283.

Cheatham, M. and K. Cleereman. (2006). Application of Social Network Analysis to Collaborative Team Formation. *Intl. Symp. on Collaborative Technologies and System*, May, pp. 306-311.

Clauss, M. (2001). *Modelling variability with UML*. GCSE 2001 - Young Researchers Workshop.

Clements, P., and L. Northrop. (2002). Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA, USA.

Clements, P., and L. Northrop. (2004). A Framework for Software Product Lines Practice. http://www.sei.cmu.edu/productlines/framework.html

Dice, L.R. (1945). Measures of the Amout of Ecologic Association Between Species. Ecology, Vol. 26, No. 3, pp. 297-302.

Domingos, P. (2005). Mining Social Networks for Viral Marketing. IEEE Intelligent Systems, Vol. 20, No. 1, pp. 80-82.

D'Souza, D.F. and A.C. Wills. (1999). Objects, Components, and Frameworks with UML: The Catalysis Approach, Addison-Wesley, Reading, MA, USA.

Fantechi, A., S. Gnesi, G. Lami, and E. Nesti. (2004). A Methodology for the Derivation and Verification of Use Casees for Product Lines. Pages 255-264. The 3rd International Conference, SPLC 2004. Springer Verlag, Boston, MA, USA.

Festinger, L. (1949). The Analysis of Sociograms using Matrix Algebra, Human Relations, Vol. 2, pp. 153-158.

Frakes, W.B. and K. Kang. (2005). Software Reuse Research: Status and Future. IEEE Transactions on Software Engineering, Vol. 31, No. 7, July, pp. 529-539.

Freeman, L.C. (1979). Centrality in Social Network: I. Conceptual Classification. Social Networks, Vol. 1, pp. 215-239.

Garlan, D. and M. Shaw. (1993). An introduction to software architecture. In Advances in Software Engineering and Knowledge Engineering, Volume I. World Scientific Publishing Company.

Gibson, P., B. Mermet, and D. Méry. (1997). Feature Interactions: A Mixed Semantic Model Approach in O'Regan and Flynn, eds. 1st Irish Workshop on Formal Methods (IWFM97), Dublin.

Gomaa, H., and M. E. Shin. (2004). A Multiple-View Meta-modeling Approach for Variability Management in Software Product Lines. Pages 274-285. 8th International Conference (ICSR 2004). Springer Verlag, Madrid, Spain.

Griss, M. L., J. Favaro, and M. d. Alessandro. (1998). Integrating feature modeling with the RSEB. the 5th International Conference on Software Reuse. IEEE Computer Society Press, pp. 76-85.

Hanneman, R.A. and M. Riddle. Introduction to Social Network Methods, Riverside, CA: University of California, Riverside (published in digital form in http://faculty.ucr.edu/~hanneman/ (retrieved November 30th, 2012).

Hartree, D. (1949). Calculating Instruments and Machines. The University of Illinois Press, pp. 112.

Hassan, A.E. (2008). The Road Ahead for Mining Software Repositories. 24th IEEE Intl. Conf. on Software Maintenance, pp. 1-6.

Houhamdi, Z. and S. Ghoul. (2001). A Reuse Description Formalism. Intl. Conf. on Computer Systems and Application, June, pp. 395-401, Beirut, Lebanon.

Jirapanthong, W. (2005). Supporting Product Line Development through Traceability. 12th Asia-Pacific Software Engineering Conference (APSEC 2005), Taipei, Taiwan.

Jirapanthong, W. (2008). An Approach to Software Artefact Specification for Supporting Product Line Systems. the 2008 International Conference on Software Engineering Research and Practice (SERP'08), Las Vegas, Nevada, USA.

Jirapanthong, W., and A. Zisman. (2009). XTraQue: traceability for product line systems. Software and System Modeling 8(1): 117-144.

John, I., and D. Muthig. (2002). Tailoring Use Cases for Product Line Modeling. REPL'02, Essen, Germany.

Kang, K., S. Cohen, J. Hess, W. Novak, and A. Peterson. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Kang, K. C., S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. (1998). FORM: a feature-oriented reuse method with domain-specific architectures. Annals of Software Engineering 5: 143-168.

Kleinberg, J.M. (1999). Authoritative Sources in a Hyperlinked Environment. Journal of the ACM, Vol. 46, No. 5, pp. 604-632.

Knoke, D. and R.S. Burt. (1983). Prominence. R.S. Burt and M.J. Minor eds., Applied Network Analysis, Newbury Park, CA: Sage, pp. 195-222.

Lopez-Fernandez, L., G. Roble, and J.M. Gonzalez-Barahona. (2004). Applying Social Network Analysis to the Information in CVS Repositories. Proc. of the 1st Intl. Workshop on Mining Software Repositories, pp. 101-105, May.

Luce, R.D. and A.D. Perry. (1949). A Method of Matrix Analysis of Group Structure. Psychometrika, Vol. 14, pp. 95-116.

Luce, R.D. (1950). Connectivity and Generalized Cliques in Sociometric Group Structure. Psychometrica, Vol. 15, pp. 159-190.

Madey, G., V. Freeh, and R. Tynan. (2002). The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory. 8th Americas Conf. on Information Systems, pp. 1806-1813.

McIllroy, M.D. (1976). Mass Produced Software Components. In J.M Buxton, P. Naur, and B. Randell, edt., Software Engineering Elements and Techniques, 1968, NATO Conference on Software Engineering, pp. 88-98.

Miller, G.A. (1995). WordNet: A Lexical Database for English. Communications of the ACM, Vol. 38, No.11, pp. 39-41.

Mokken, R.J. (1979). Cliques, Clubs and Clans", Quality and Quantity, Vol. 13, pp. 161-173.

Northrop, L. M. (2002). SEI's Software Product Line Tenets. IEEE Software, Vol. 19, pp. 32-40.

Ohira, M., T. Ohoka, T. Kakimoto, N. Ohsugi, and K. Matsumoto. (2005). Supporting Knowledge Collaboration using Social Networks in a Large-Scale Online Community of Software Development Projects. Proc. of the 12th Asia-Pacific Conf. on Software Engineering, Dec., pp. 6-10.

Page, L., S. Brin, R. Motwani and T. Winograd. (1998). Pagerank Citation Ranking: Bringing Order to the Web, Technical Report, Stanford University.

Porter, M.F. (1980). An Algorithm for Suffix Stripping. Program, Vol. 14, No. 3, pp. 130-137.

Prieto-Diaz, R. (1991). Implementing Faceted Classification for Software Reuse. Communications of the ACM, Vol. 34, No. 5, May, pp. 89-97.

Rueda, G., P. Gerdsri and D.F. Kocaoglu. (2007). Bibliometrics and Social Network Analysis of the Nanotechnology Field. Proc. of the Portland Intl. Center for Management of Engineering and Technology, August, pp. 2905-2911, Portland, Oregon, USA.

Sabidusi, G. (1996). The Centrality Index of a Graph., Psychometirka 31, pp. 581-606.

Scott, J. (2000). Social Network Analysis: A Handbook, 2nd edition, Sage, London.

Scripps, J., P. Tan and A. Esfahanian. (2007). Node Roles and Community Structure in Networks. Proc. of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, pp. 26-35, NY, USA.

Seidman, S.B. and B.L. Foster. (1978). A Note on the Potential for Genuine Cross-Fertilization Between Anthropology and Mathematics. Social Networks, Vol. 1, pp. 65-72.

Seidman, S.B. (1983a). Network Structure and Minimum Degree. Social Networks, Vol. 5, pp. 269-287.

Seidman, S.B. (1983b). Internal Cohesion of LS Sets in Graphs. Social Networks, Vol. 5, pp.97-107.

Smith, E., A. Al-Yasiri, and M. Merabti. (1998). A Multi-Tired Classification Scheme for Component Retrieve. Proc. of the 24th Euromicro Conf, August, pp. 882-889, Vesteras, Sweden.

Souza, C., J. Froehlich, and P. Dourish. (2005). Seeking the Source: Software Source Code as a Social and Technical Artifacts. Proc. of the 2005 Intl. ACM SIGGROUP Conf. on Supporting Group Work, pp. 197-206.

Svahnberg, M., J. Gurp, and J. Bosch. (2001). On the Notion of Variability in Software Product Lines. Pages 45-55. The Working IEEE/IFIP Conference on Software Architecture.

Tang, J., D. Zhang and L. Yao. (2007). Social Network Extraction of Academic Researchers. *Proc. of the 7th IEEE Intl. Conf. on Data Mining*, October, pp. 292-301.

Wasserman, S. and K. Faust. (1994). *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge, UK.

Weng, C., W. Chu and J. Wu. (2007). Movie Analysis Based on Roles' Social Networks. *IEEE Intl. Conf. on Multimedia and Expo*, July, pp. 1403-1406.

Xiu-Min, Y. and F. Zhu-Qing. (2007). The Research of Link Relationships Based on SNA. *Proc. of the Intl. Conf. on Service Systems and Service Management*, June, pp. 1-6.