2010

Efficient techniques for partitioning software development tasks

Samyukta Soothram
Iowa State University

Efficient techniques for partitioning software development tasks

by

Samyukta Soothram

A thesis submitted to the graduate faculty

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Systems

Program of Study Committee:
Zhengrui Jiang, Major Professor
Yoshinori Suzuki
Michael Crum

Iowa State University

Ames, Iowa

2010

DEDICATION

I dedicate this thesis to my mother, Mrs. S.Subhadra and my father, Mr.S.V Ramana for valuing education enough to get me where I am today. Their support and words of wisdom helped me to complete this work. I would also like to thank my brother, Anurag and my grandfather, Mr.T.N.Sethumadhavan for their constant motivation and encouragement during my graduate student life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This research examines the problem of assigning software development tasks to teams. The goal of this study is to model the most efficient way of module assignments in order to reduce the communication and coordination delays among software teams that arise from the improper distribution of software modules. The study quantifies the module interactions using software coupling design measure and models these interactions using Linear Programming and Cluster Analysis techniques. The performance of the two techniques is evaluated to find the one that offers the most favorable set of module assignments that can be used by software practitioners in the real world. The results obtained from this research suggest that though Linear Programming is the most optimal technique for obtaining the solution, it cannot provide solutions for large problems. With an increase in the number of modules, the computational time required for Linear Programming model increased considerably. Cluster Analysis, on the other hand, provided solutions which were not as optimal as Linear Programming but generated module assignments for large module count problems. Two types of Cluster Analysis techniques, namely agglomerative clustering and partitional clustering were implemented in this research. Of the two, agglomerative cluster analysis technique offered the most efficient and practical solution for module assignments. This research is an attempt to improve the decision making capabilities of software practitioners who often make use of intuitions and their past experiences in the process of assigning modules to software development teams.

CHAPTER 1 INTRODUCTION

An Information System can be defined as an aggregation of practices and methodologies to translate data into information and knowledge that is useful to organizations and/or other entities. An Information Systems development life cycle refers to the process of evolution of the information system and consists of four phases- planning, analysis, design and implementation. This research focuses on the design phase of the Information Systems development life cycle.

Software design defines *how* the system functions by determining the hardware, software and network infrastructure required for operation (Alan Dennis, 2005).The software design activity starts with structuring high-level system design platform, system architecture, deployment environment, hardware and software resource requirements and extends to finer details such as module design, data flow, communication flow and so on. Technological innovations, evolving programming models and complex development environments have altered the traditional methods of software design used by practitioners. One of the software design approaches that have been inherently used is the *structured design* approach that advocates the philosophy of configuring an information system as a set of components. Each component is structured to address at least one of the functionalities of the information system. The collection of all components constitutes the information system as a whole. Structured design refers to the "art of designing the components of a system and the interrelationship between those components in the best possible way"(Edward Yourdon, 1975). Structured design can be implemented by dividing software code into discrete functional units known as modules, based on the similarity of purpose and commonality of data. The modular structure of software results in some immediate benefits such as ease of implementation, efficient maintenance and reduced cost of modification. The cost of implementing an information system with modular design minimizes system faults and

system costs because the parts of problem(s) to be handled become small and solvable separately (Edward Yourdon, 1975).

A module represents the basic work item of an information system design. A module can be tailored to represent a subsystem, an object or a class. In object oriented systems, a module represents the smallest unit of code that can be developed independently. The object-oriented software design approach involves designing modules by grouping the data and processes such that they address at least one of the functional requirements of the software. Once the modules have been determined, the next task in the design process is to allocate these modules to different software teams so that they can be developed independently. Though each module addresses certain functionality, since the software package is a collection of all these individual modules, there is some level of dependency among the modules. Even though the modules are developed individually, they cannot operate in complete isolation. The interdependency among modules complicates the task of disconnected software development since dependent modules share data, memory and/or processes. In order to deal with the module interdependencies, constant communication and coordination between teams is required. The absence of effective communication can lead to inefficient software development.

Managing functional dependencies between modules during early stages of software design ensures better software quality (Hung-Fu & Lu, 2009) . The goal of this effort is to construct a methodology that can be used by software practitioners for assigning software modules (development work items) across concurrent development teams. This will streamline communication flow across teams and allow software development projects to be completed faster, and with a lower risk of project failures.

1.1 Motivation

Though the concept of structured design has been around in the industry for few decades, the dawn of globalization has added a new dimension to the validity and significance of this approach. Globalization has resulted in geographically distributed development sites, creating new challenges and complexities for software practitioners. Dynamic business needs and availability of resources have changed the structure of software production. The nature of global software projects has led to distributed development sites (Mockus & Weiss, 2001), with several remotely located teams, placing high demands on the level of communication and coordination. It is highly challenging to configure and control the organizational makeup of software teams. Improper communication and collaboration result in negative productivity and longer production intervals.

Practices influencing research and proven methodologies (Damian & Moitra, 2006) have helped in cultivating global software development. Quantitative research (Damian & Moitra, 2006) has shown that work items in distributed environments appear to consume larger chunks of time, about 2.5 times of time required for completion of work items which are developed together at one site. Delays in productivity can be attributed to glitches caused by lack of communication and coordination, along with the complexity and size of the project (Bin et al., 2007).

The software design activity for distributed environments is ingenious. Software managers often determine the distribution of software module designs to development teams based on the assignment decisions made in past projects and their intuition. Developing an optimal methodology of partitioning modules in a software package improves the decision making capability of software managers. It helps them in comparing their learning, knowledge of work item (module) distribution and intuition with the scientific approach

developed as a part of this study. The benefits of the software modules partitioning techniques proposed in this research include

- practical and optimal module assignment logic,
- reduced communication and collaboration delays with respect to software development within and across teams,
- increased productivity in distributed software development environments,
- improved management of shared data and processes in a software project and
- increased accountability of developers for the assigned modules

Coupling is a software design measure that assesses the strength of the interconnection or the extent of dependency between modules. It reflects how much change is required in one module, after changing some other module in the system. Module coupling helps in determining understandability, testability, maintainability and reliability of software (Hla Myat, Nan Si, & Ni Lar, 2005).

While proposing software design, the aim is to develop a system consisting of loosely coupled modules. Such design provides the advantage of manipulating a module without disturbing much of the configuration of other modules in the set (collection of modules, software package). This research is based on the theory of coupling and its application in the partition of software modules.

1.2 Summary Problem Statement

The primary objective of this research is to devise an efficient technique for grouping software modules with an aim of assigning these modules across development teams. Object-oriented systems form the environment and software modules constitute the variables monitored as part of this study. Literature search in the field of modularization and coupling reveals that no methodology has been developed in the area of software engineering to help

practitioners in optimal decision making process of allotting the software modules (software development tasks) to teams. The research questions that need to be addressed are:

- What are the efficient techniques for partitioning software modules?
- How can these techniques be implemented?
- Which technique offers the best solution?

Linear Programming and cluster analysis are examined as the techniques for partitioning software modules. Linear Programming approach can produce optimal grouping of modules while cluster analysis offers a fast heuristic approach. The results from both these approaches are examined to find the optimal and practical solution that can be used in the real world by software practitioners. The results from this research can be used in distributed software development environments to aid software productivity and reduce communication setbacks.

CHAPTER 2 LITERATURE REVIEW

This section summarizes the literature study conducted for formulating the research methodology.

2.1 Modularization

Modularization is the main characteristic of modern business applications. It is a technique of separating software code into chunks of 'domain modules' (Sarkar et al., 2009) such that interactions among modules is minimized. Frequent changes, upgrades and maintenance in requirements due to evolving markets and varying customer preferences result in the need for flexible and comprehensible systems (Parnas, 1972), where every domain module can be changed without a need to change other modules and every module in the software can be studied one at a time. System modularization does not imply random division of a system into modules instead; it involves a well-defined approach to generate segments which are meaningful in isolation and also communicate with other modules in the system. Such modules simplify reassembly, replacement and other maintenance activities without a need for changing the whole system. Thus, modularization of software helps in achieving maintainability and flexibility.

A domain module is a basic functional unit of software development and maintenance (David, Gerald, & Frank, 1985). A module is the fundamental component of any structured software design. By configuration, a module can be a subsystem or a set of data and processes wrapped together. Several modules may share all or part of data and processes. A good module (Sarkar et al., 2009) provides access to data and processes which satisfy a specific functional requirement. The extent to which data and processes are shared among modules, determines the extent of interdependency between those modules.

The definition of module can be manipulated to fit into different levels of software design. At high-level of software design, a module can assume the meaning of a sub-system or any other high-level functional unit of an information system whereas at low-level of software design, a module can be a class, an object or a collection of these. Successful attempts have been made in implementing modularization at high level design as well as decomposing modules into sub-modules (Michael & David, 1996) to facilitate convenient reuse and remodeling.

The guidelines for modularization discussed in literature (David et al., 1985; Parnas, 1972; Sarkar et al., 2009) are similar but the approaches differ in the way they are implemented. One of the approaches examines modularization at two levels (Parnas, 1972); first wherein the module design is based on the structure of software design flowchart and the second that is based on 'information hiding', wherein the modules are generated such that the information within a module is hidden from other modules. Modularization based on flowchart might be easy to achieve since there is little transparency in the flow of data and inter-module procedural calls but it does not capture the difficult design decisions. Thus, modules should be designed based on the data and collection of code and not the steps corresponding to the flowchart (Parnas, 1972). Modularization based on relaxing the standards of information hiding (Michael & David, 1996) limits every sub-module component to one design choice such as data structure, data type and time of binding, etc. Such decompositions promote the idea of limiting the module to a fewer number of decisions (Garlan, Allen, & Ockerbloom, 2009; Kiczales, 1996).

The goal of managing dependencies is essential to modularization. The process of software design needs to be carefully implemented in order to identify and create modules that manage dependencies and provide traceability. The Axiomatic Design approach (Hung-Fu & Lu, 2009) helps in tracing the different design decisions by constraining the number of decisions and dependencies that should exist as a part of good software design.

One of the recent approaches to modularization suggests a three-layered architecture for examining modularization: application, domain and infrastructure (Sarkar et al., 2009). The modules are formed such that they serve as access points to all the three layers of a business application. Each layer consists of modules that provide specific functionality. This approach limits the direction of communication among modules. The modules within a layer are designed to communicate with other modules in the same layer using an application program interface. The modules residing in the upper layers of the system design can communicate directly with the modules in the lower layers. However, the modules in the lower layer should be designed in such a way that they cannot communicate with modules in the higher layers.

The literature study on modularization reveals that the primary challenge in modularizing applications is in finalizing the criteria consisting of an appropriate mix of module strength, module size, information hiding and design decisions. Although smaller module size costs more than larger module size (David et al., 1985), choosing only module size is the least convincing strategy for optimal modularization. The resultant set of modules in every structured design initiative should have minimal fault rate and low cost. A great responsibility lies with the developers in implementing the modular design structure to ensure that the benefits of modularization, evident in the software design are carried forward in their entirety and remain valid in the software implementation phase as well. Module size and module strength should always be considered in parallel while modularizing the system. This will not only ensure lower cost but also lower fault-rates, provided the programmers are suitably encouraged to write high-strength modules. The real test of modularization lies in striking a balance between module strength and information hiding. Hence, practitioners need to put in a lot of effort to propose a structured software design that satisfies all modularization principles.

2.2 Coupling

Coad and Yourdon (Yourdon, 1991), define a good software design as "one that balances trade-offs to minimize the total cost of the system over its entire lifetime". Software design quality in an object-oriented system can be evaluated using a pre-defined set of criteria. The set of criteria primarily consist of coupling and cohesion (Alan Dennis, 2005).While coupling measures the interdependency between modules, cohesion measures the closeness of the processing elements of the modules data and functions taken together.

Coupling exposes the visibility of interactions among various modules of the system. It helps in identifying and understanding the set of modules that would be impacted together, due to high degree of dependency. In case a change is required in one module, coupling allows identification of those modules that need to be changed in order to maintain the software system design. Consider two modules under observation, Module A and Module B. Module A is said to be tightly-coupled with Module B if Module A is highly dependent on Module B and vice-versa. This implies that changes in Module A can cause significant changes in Module B. Thus in order to understand the functioning of Module A, the functioning of module B should be known.

The degree of coupling is directly proportional to the extent of interactions among modules. The magnitude of coupling can be specified in quantitative as well as qualitative terms. The stronger is the interaction among modules, the higher is the degree of coupling; and the weaker is the interaction among modules, the lower is the coupling. The cost of developing a modular software system is largely determined by the existence of the type of coupling and the extent of coupling between modules (Edward Yourdon, 1975; Mockus & Weiss, 2001).

There exists a conflict between the opinions of software engineering gurus and practitioners when it comes to identifying the modularization drivers. Related work in this

area (Brito e Abreu & Goulao, 2001) suggests the existence of other design criteria (semantic organization) apart from coupling and cohesion, which serve as modularization drivers and are more convincing to the practitioners.

The debate in identifying modularization drivers might place relatively less weight on coupling and cohesion as the panacea, but these measures help in predicting the behavior of the modules. In modular systems, there is a need to clearly understand the direction of information flow, the frequency of communication, the amount and type of shared data, common processes etc. These factors influence the performance of the system. Efficient design includes benefits derived from coupling and cohesion. Since this research examines the issue of partitioning software modules, it makes use of only coupling to model the software module assignments.

Research studies prove that coupling and cohesion should be examined together and not in isolation in the context of software maintenance (Darcy, Kemerer, Slaughter, & Tomayko, 2005). Wood's task complexity model offers a solution to capture coupling complexity by accounting for the sources of task complexity, namely component, coordinative and dynamic. Though coupling and cohesion are two different structural complexity measures, there exists an 'interaction term' (Darcy et al., 2005) that associates coupling and cohesion and helps in better understanding of each program unit in terms of design, development and maintenance.

Since this study focuses on determining module assignments to teams based on the interactions between modules, it is assumed that excluding cohesion design metric will not lead to biased results. Coupling strengths between modules forms the basis of this research.

One of the prime cost reduction targets of the traditional SDLC model (Software Development Life Cycle) is the software development phase. The costs of development are directly affected by the software design, skills of software developers and the resources allocated for development (infrastructure, time to go-live, etc). Better designs lead to better

performance and economical budget allocations. Downstream phases in SDLC are directly impacted by the decisions taken in the upstream. After requirements gathering, software design is the next most important phase of SDLC. Subsequent phases of SDLC base their assessments and performance out of phase. Continuous refinement of the decisions made in the software design phase is imperative for identifying opportunities for cost reduction and for system improvement opportunities.

Poisson regression models (Briand & Wust, 2001) using coupling, cohesion and class size have been used in determining the software development efforts. Coupling and cohesion explain development effort to a limited extent. The main driver for development effort was found to be class size. The management of coupling (Cain & McCrindle, 2002) in the software design helps in improving team dynamics and system productivity. When software is being developed by teams, it is important to understand the distribution of tasks and module development plans in order to ensure good structured software system. This study focuses on coupling and examines how it influences the module assignments in the process of software development and improves the productivity of the system.

The structure of teams helps in improving and accelerating the development of any project under observation. Clustering techniques (Paul & Jack, 2000) have been used successfully in the allocation of tasks for concurrent teams operating in the engineering domain. The task allocation approach used for engineering teams cannot be directly applied to software teams because of the difference in the nature of engineering tasks and software development tasks.

Cluster analysis (Brito e Abreu, Pereira, & Sousa, 2000) has been used to account for the quality of modularization in software systems. Cluster analysis can be used to group modules into distinct clusters. Each cluster consisting of finite set of modules can be allocated to different software development teams. Such task assignment helps in reducing communication and coordination glitches across teams.

The literature review suggests the need to improve quality, performance and productivity of software systems using design criteria, predominantly coupling, since it classifies modules based on their interactions with other modules in the system. This study focuses on areas not explored by previous research and formulates methodologies that partition modules into different groups. It uses clustering and Linear Programming techniques to group modules into clusters such that highly interactive modules are grouped in one cluster.

CHAPTER 3 RESEARCH METHODOLOGY

The aim of this study is to devise a methodology for optimal classification of modules into groups depending on the strength of module interactions. The modules are partitioned such that the modules which are highly dependent on each other are grouped into a single cluster and modules in one cluster are less dependent on modules of the other cluster. The problem formulation for the research involves addressing the following goals:

- developing efficient techniques for grouping software modules by preserving software design policy of reduced coupling and enhanced cohesion among modules, and
- comparing the suggested techniques in order to find the most feasible technique that can be implemented by software practitioners to solve large scale module assignment problems

The steps that were followed in order to realize the goals mentioned above are as follows:

- identifying modular software systems and quantifying the coupling data between modules,
- investigating the traditional techniques to classify/group data observations,
- modifying the traditional methods to suit the needs of the research problem,
- generating the sample data for the research,
- comparing the results by applying different techniques to sample data, and
- suggesting the most optimal method for use in real world

The results obtained from this research are expected to help software practitioners in assigning the software modules to development teams such that the communication and coordination delays arising across teams due to improper assignments are reduced resulting in quicker development cycles.

3.1 Research Description

This research is specifically targeted at modular systems developed as a part of object-oriented system design. This research issue of module assignments is handled by considering initially, a system composed of a small set of modules. To start with a module count of 5 is considered for formulating the partition methodology. This was done with the purpose of ease of problem formulation for a smaller module count. The module count is gradually increased till the stage where results could be easily computed using Linear Programming and Clustering. The limit on the highest number of modules considered for this research reflects the complexity of the research issue and the extent of difficulty in modeling and achieving the most favorable module assignments.

As mentioned in the previous sections, coupling can be applied at different levels of system design; module, class, sub-system etc. In this research, coupling is studied at module level, though it can be extended to class or sub-system level. A change in the level of coupling changes the method of quantifying coupling coefficients. The methodology suggested in this paper for module partition can be applied directly to scenarios using coupling at different levels since the suggested methodology is independent of the coupling level under observation.

The coupling coefficients are used as inputs for this research. Coupling coefficients are calculated depending on the coupling categories and their designated weight schemes. If coupling is examined at a different level of software design, apart from module level, such as sub-system etc then the scheme used in this research for calculating coupling coefficient cannot be applied directly. In such cases a new method of measuring coupling coefficient capturing is required.

The output of the research consists of subsets or groups of modules. The groups combine highly related modules together. The research ensures that the module assignments

generated in the output preserve the software design philosophy of low coupling. At the higher level of system design, the proposed method can be modified to study the optimal groupings of sub-systems which could be developed together, while at lower level of system design, the proposed method can be used to identify the classes which should be developed together by teams present at same work location. The steps undertaken for conducting this research are listed in the sequential order:

- preparing the interdependence matrix,
- transforming the interdependence matrix into distance matrix,
- obtaining the Linear Programming results(objective function value and grouping assignments) by using interdependence matrix as an input to the Linear Programming model (solved by Premium Solver excel add-in),
- computing the clusters using SAS program by using distance matrix as input,
- testing for the objective function value by plugging in the cluster results (grouping assignments) derived from SAS, and
- comparing the change in objective function value and group assignments for Linear Programming and Clustering

3.2 Quantifying Coupling

Measuring the quality of software design is essential for building a robust Information System. Estimating software quality from software design helps in avoiding expenses that result from bad architecture or design of the software systems. Post implementation changes demand huge investments. Software design offers a good platform to estimate the system performance.

Software design quality is examined by identifying the components that make up the system and understanding the interrelationships between the modules. The interrelationship between modules exposes the frequency of interactions, type of data movement (call-by-

reference, call-by-value etc) between modules, locus of impact (Offutt, Abdurazik, & Schach, 2008), relationship between modules and other interaction details. The information on these interactions and interrelationships is well-captured by coupling. Coupling-based structural metrics involve examination of the quality of coupling as a design measure.

The approach for measuring software design quality using coupling has evidenced significant changes. Literature review on the measurement of software quality suggests a change in the level at which coupling is examined. The common principle of measuring software design quality is to examine the call traffic between modules (using graphs and sub-graphs). However, the approaches of measuring software design quality differ in levels at which coupling is studied. The design measures overlap in dimensions of quality measurement (Lionel, Jurgen, John, & Porter, 2000). The measures coincide to predict the same set of quality attributes of a software design.

Before proceeding with the measurement of coupling-based software quality, it is important to choose the level of coupling at which the quality metrics need to be defined; class-level, system level, and/or Application Program Interface (API) level and so on. Couplings can be studied at different levels; at module-level or API level (Sarkar, Rama, & Kak, 2007) by segregating aspects of software design at architectural and structural level. Techniques have been developed (Offutt et al., 2008) for evaluating coupling of classes that are not exposed until runtime. These techniques analyze the source code to identify the existence of coupling and to make use of criteria such as number of classes, types of coupling to effectively construct the quality metric. The structural metrics that quantify coupling (for modules residing in API Application Program Interface) and the mode in which they communicate with different modules in the program reflect the evolving nature of software architecture.

The quality of modularization at API level is affected by object-oriented design characteristics such as inheritance, base-class design and others (Sarkar, Kak, & Rama,

2008). The procedure for measuring the software quality design is carried out by identifying and separating modules in the environment in which they exist and function. For example, one of the methods that capture the coupling-based structural metrics includes examining the modules that reside within an API by using 'module interaction index' (Aruna, Devi, & Deepa, 2008) that measure the frequency of interactions between modules within the API with the ones outside the API.

All coupling measurements capture the strength of module interactions. This research does not propose a theory for measuring module coupling but uses pre-existing methodologies for quantifying coupling.

One of the ways of quantifying coupling is by measuring of the strength of interconnections between modules (Edward Yourdon, 1975). The following three categories measure the strength of module interconnections:

- Highly coupled,
- Loosely coupled, and
- Decoupled/No coupling

Two or more modules are said to be highly coupled if they have a strong degree of interdependence. If modules are highly coupled then a change or maintenance activity in one module will cause changes in other dependent modules. This increases software development and maintenance problems and leads to poor software performance. The total cost of highly coupled modular systems is high.

Loosely coupled modules are characterized by weak degree of interdependence. Loosely coupled modules are less dependent on each other. This reduces the frequency of communication. The level of interaction is less in loosely coupled modules as compared to highly coupled modules. Loosely coupled modular systems reflect a good system design and increase maintainability, flexibility and reliability levels.

The modules in the decoupled/uncoupled modular structure do not interact. They function independent to each other. Such modular design is difficult to incorporate in real world applications. In scenarios wherein modules share data and processes, this modular structure is not implemented.

Coupling can also be quantified based on the object-oriented properties of inheritance and interaction (Yourdon, 1991). Inheritance is a feature of object-oriented programming wherein new classes are formed using the pre-defined or existing classes. Inheritance coupling measures the closeness of classes in the inheritance hierarchy. Problems with inheritance are due to the features in object-oriented programming that help in violating encapsulation and information hiding (Alan, 1986). Interaction coupling refers to the coupling that exists due to the flow of data or messages between modules. It is the coupling type that exists when the modules (methods or objects) communicate using message passing. The Law of Demeter (Lieberherr & Holland, 1989) suggests minimizing the number of interactions the module has with other modules in the program. The Law of Demeter states that an object should send messages to only either itself, one of its superclasses, an object that is passed as a parameter to a method, an object created by a method, or an object that is stored in a global variable

Every case mentioned above increases the degree of interaction coupling which is not considered healthy from a system design perspective. The six types of interaction coupling (Jones, 1988; Myers, 1978), in the diminishing order of effect (good to bad) they have on systems are:

- No direct coupling,
- Data coupling,
- Stamp coupling,
- Control coupling,
- Common or Global coupling, and

- Content or Pathological coupling

No direct coupling is the highly desired type of coupling in which modules of the system are not dependent on one another. Hence, there are no interactions among the modules constituting the system. Data coupling is the type of coupling where one method passes a variable to another method. The method that passes the variable is known as calling method and the method that receives the variable is known as the called method. Stamp coupling is a slight variation of data coupling. In data coupling, if the calling method passes a composite variable to the called method then the called method uses the entire composite variable to complete its function. However, in stamp coupling, only a portion of the composite variable will be used by the called method. In Control coupling the value of the control variable, passed by the calling method, determines the execution of the called method. When the methods refer to a 'common' or 'global' data area then methods are said to be common or globally coupled. Content or Pathological coupling is the least desirable form of coupling. The concept of 'friends' in object-oriented paradigm promotes the philosophy wherein a method of one object can refer to hidden parts of another object. Content or pathological coupling makes use of this feature.

The direction of information movement between modules influences interaction coupling (Hla Myat et al., 2005). The two broad types of interaction coupling based on direction of information flow are import coupling and export coupling. Import coupling counts the messages *received from* modules whereas export coupling counts the messages *sent to* modules. The broad categories of import and export coupling include call coupling, scalar coupling, stamp coupling and tramp coupling.

This research uses coupling coefficients to group modules. The weights assigned to coupling categories are nominal, limited to a scale of 0-5, 0 signifying least interdependence and 5 signifying maximum interdependence. The coupling coefficients are quantified by

considering the type of connection between modules, the complexity of interface, and the type of information flow among the connection.

Two sub-types of connection between modules were identified, namely minimally connected and normally connected. A minimally connected module has least level of coupling and is the most desirable form of module design. Modules which are minimally connected have least interdependency. A normally connected module is more coupled than minimally connected module and is the least desirable form of module design. Modules which are normally connected have high interdependency. A pathological connection is established when a reference to an entity within a module originates from outside the module. A good software design does not accommodate pathological connections in module interactions. Thus, it is not considered for the purpose of this research.

The complexity of the interface is measured by the type of the parameter passing method. When two or more modules interact there is exchange of data. This data can be transferred from one module to another using different parameter passing methods such as call by reference and call by value. An implicit reference is received in call by reference parameter passing style and it demands a greater communication potential between the interacting modules. Call by value, on the other hand, passes arguments using a copy of the value, which limits the communication between the two interacting modules.

The frequency of the calling method determines the extent of use of the module in the software package. The more the frequency of calling, the greater is the level of interactions. If a module is called frequently by another module in the software package, then the frequency of interaction of this module pair is considered to be high.

When two modules interact, there is flow of information; data and/or control. This feature of module interaction was assigned weights depending on the most desirable to least desirable form of coupling interaction. The types of interaction couplings quantified for this research are data coupling, control coupling and common coupling. If one or all of the

contents of one module are included in the contents of another then it is known as content coupling. It is least desirable form of module interaction design. It is assumed that the modules under observation for the purpose of the research align with the guidelines of a good software design; hence this form of coupling is not quantified in this research.

Binding time accounts for the nature and time when symbolic data references are converted into physical machine addresses. If this conversion occurs at compile time then it is known as compile time binding and if it is done at execution time then it is known as execution time binding. With compile time binding, the inter-modular reference is fixed at compile time which leads to stronger reference coupling than execution or run time coupling where the association are made at run time. Table 1 summarizes the coupling categories and weights used in this research.

Table 1 Coupling Categories and Weights

Acronym	Coupling Category	Weight
MC	Minimally connected	0
NC	Normally connected	1
CR	Call-by-reference	2
CV	Call-by-value	1
FC	Frequent calls	2
NFC	Non-frequent calls	1
DC	Data coupling	1
CC	Control Coupling	2
CmC	Common Coupling	3
CTB	Compile Time Binding	2
RTB	Run Time Binding	1

3.3 Research Technologies

The partitioning of software development modules into distinct groups was implemented using Linear Programming and Cluster Analysis. This section describes how these traditional methods were modeled to suit the research requirements.

3.3.1 Optimization and Linear Programming

Optimization helps in making the most efficient use of available resources, under defined set of operating environments. The philosophy of optimization rests on the principle of choosing the best component from a set of available alternatives. Mathematical programming is an optimization technique that helps in finding the optimal solution (Anderson, 1994) with limited availability of resources in order to realize the objectives of an individual or a business (T.Ragsdale, 2007).

The approach to any optimization problem involves considering the following issues:

- The objective that determines the goal that is considered while determining the best alternative,
- The decisions that need to be made to realize the goal, and
- The constraints that limit the use of resources that are available to realize the goal

3.3.2 Linear Programming in this research

Linear Programming (LP), one of the mathematical programming techniques, solves optimization problems using linear objective functions and linear constraints. Formulating a Linear Programming model involves expressing the optimization function algebraically, by specifying the objective function, decision variables and the constraints. The steps in formulating the Linear Programming problem as adopted from (T.Ragsdale, 2007) for this research include the following:

- understanding the problem,
- identifying the decision variables,
- stating the objective function as a linear combination of decision variables,
- stating the constraints as a linear combination of decision variables, and
- identifying any upper and lower bounds on the decision variables

The problem in this research is to calculate the best possible way in which the software modules can be grouped. The decision variables in this research are found to indicate the assignment of the modules in same or different groups. Binary decision variables are used in this research, a value of 1 indicating module assignments in the same group and a value of 0 indicating module assignments in different groups. The objective function for this research is formulated using the interdependence index (interdependence value) for each module pair. The objective is to maximize the product of decision variable X_{ij} and the interdependence index C_{ij} for all possible module pair combinations. The constraints for this research are designed to obey the software design rules of modular coupling. Consider a scenario with three modules, module i, module j and module k. Let X_{ij}, X_{jk} and X_{ik} be the decision variables denoting the group assignments among modules i, j and k, then the maximum combinations of group assignments are as shown below

Table 2 Example of Module Assignments into groups

X_{ij}	X_{jk}	X_{ik}	Module i and j	Module j and k	Module i and k
1	1	1	Same Group	Same Group	Same Group
0	0	0	Different Group	Different Group	Different Group
1	**1**	**0**	**Same Group**	**Same Group**	**Different Group**
1	**0**	**1**	**Same Group**	**Different Group**	**Same Group**
0	**1**	**1**	**Different Group**	**Same Group**	**Same Group**
1	0	0	Same Group	Different Group	Different Group
0	1	0	Different Group	Same Group	Different Group
0	0	1	Different Group	Different Group	Same Group

The rows indicate in bold denote assignments which are illegal and cannot become a part of module assignment logic. This is because they do not obey the law of transitive relationship. Consider Module A, Module B and Module C. If Module A and Module B are in one group, Module B and Module C are in one group then Module A and Module C should also be in the same group. If this law of transitivity is not obeyed then such module assignments are illegal and cannot be implemented in the model. These illegal module

assignments are the constraints for this research. In order to limit the number of module assignments per group, upper and lower bounds for this research are identified. Lower bound specifies the minimum number of modules that can exist in a group and upper specifies the maximum number of modules per group.

The Linear Programming model that is formulated for this research is described below:

Consider

Let $V = \{1, 2, \ldots, n\}$ be the finite set of modules to be grouped, c_{ij} $(j > i)$ be the interdependence index of modules i and j, x_{ij} $(j > i)$ be the decision variable indicating the grouping assignment (1 if i and j are in the same group, 0 otherwise), and l and u be the lower and upper bounds, respectively, of the number of modules to be included in each group. The linear integer programming formulation can be written as:

$$\text{maximize} \sum_{i \in V \setminus \{n\}} \sum_{j>i} c_{ij} x_{ij} \tag{1}$$

subject to:

$$x_{ij} \in \{0,1\} \quad \forall i \in V \setminus \{n\},\ j > i; \tag{2}$$

$$\sum_{i<k} x_{ik} + \sum_{j>i} x_{ij} + 1 \geq l \quad \forall i \in V; \tag{3}$$

$$\sum_{i<k} x_{ik} + \sum_{j>i} x_{ij} + 1 \leq u \quad \forall i \in V; \tag{4}$$

$$\left.\begin{aligned} x_{ij} + x_{ik} - x_{jk} &\leq 1 \\ x_{ij} - x_{ik} + x_{jk} &\leq 1 \\ -x_{ij} + x_{ik} + x_{jk} &\leq 1 \end{aligned}\right\} \quad \forall i \in V \setminus \{n, n-1\},\ j > i, k > j; \tag{5}$$

The number of decision variables and the number of constraints are given by (6) and (7) below (note: integer (binary) constraints are not included in (7), which is given by (6)):

$$\frac{n^2 - n}{2}; \tag{6}$$

$$2n + \frac{3n!}{3!(n-3)!}; \tag{7}$$

The number of constraints (3) and (4) each is given by n, and the number of constraints (5) is given by $C_{n,3}$, where $C_{n,3} \equiv$ combination of n objects taken 3 at a time.

3.3.3 Premium Solver: Linear Programming tool

The research made use of the in-built Excel Solver add-in to model computations for smaller module counts. But the computational limitations of the in-built Excel Solver add-in prompted the use of a more powerful tool that would solve problems with large number of constraints and computations in a lesser time frame. For this purpose, the research made use of Premium Solver provided by Frontline System, Inc ("Solver.com," 2009). The Excel Solver PSP 7.0 Education version is used as Excel add-in to solve the Linear Programming model of this research.

3.3.4 Cluster Analysis

Cluster analysis is defined as the procedure of partitioning data objects (C.Wunsch, 2009) into required number of clusters. Cluster analysis does not refer to a single method of partitioning data but refers to a wide range of algorithms. A cluster is a group or subset consisting of the data objects under observation. The goals of cluster analysis can be summarized (Blashfield, 1984) as follows:

- developing a classification,
- investigating schemes for grouping entities,
- generating hypothesis through data expression, and
- determining if the types defined through other procedures are present in a data set

The cluster analysis is performed in four steps. The first step in cluster analysis is *feature selection* or *feature extraction*. While both the terms are used interchangeably across the clustering literature, there exists a subtle difference between the two. Feature selection uses the distinguishing features from a set of candidates (observations) to perform cluster analysis (Jain, Duin, & Jianchang, 2000; Jain, Murty, & Flynn, 1999) while feature extraction generates features that uniquely identify the observations. Feature selection is very important for realizing the effectiveness of the clustering algorithm that is implemented. Any defect or error in choosing the features has a negative impact on the clustering results. The selected features should provide a clear understanding of the data, since the feature that is selected serves as the basis for cluster formation. The features of the data observations are used for classifying the observations into distinct sets.

The second step in clustering analysis is the selection of the *clustering algorithm*. This step involves formulating the proximity measure and criterion function. The proximity measure is used to define the method of determining the closeness or belongingness of observations. The criterion function helps in generating the clusters by making use of the proximity measures.

The selection of clustering algorithm is followed by the *cluster validation* procedure. Cluster analysis always leads to a definite set of clusters by partitioning the data but there is a need to examine the significance of the clusters that are formed as a result of the clustering algorithm. If the clustering algorithm leads to cluster formations which are not meaningful and/or do not offer easy interpretation, then the clustering algorithm needs to be altered. Thus, cluster validation is an important step which aids the comparison of results from multiple clustering algorithms to find out the one that best reveals the characteristics of objects.

The final step in the cluster analysis process is the result interpretation. This process helps in drawing meaningful insights from the original data. A cluster does not convey

results in itself. It is a mere group of the original set of data observations. Hence, post cluster formation, a judicious interpretation of the cluster results is required. The set of cluster is not considered as "a finished result but only a possible outline" (Anderberg.M, 1973).

Clustering analysis requires repeated trials and use of different algorithms to obtain the best resultant clusters. The primary reason for this is the lack of ability of a single clustering algorithm to generate optimal results with different sets of data. In order to find the most efficient solution, different clustering algorithms are implemented in this research.

3.3.5 Cluster Analysis in this Research

The clustering techniques that are examined in this research are partitional clustering and agglomerative hierarchical clustering. While partitional clustering divides the data into a pre-specified number of clusters, agglomerative hierarchical clustering results in clusters with a sequence of nested partitions. Of all the types of agglomerative hierarchical clustering, average link method is used to form the clusters, since it is used in scenarios where the objects are similar in their interactions. The average link method uses the average distances between all possible pairings of objects and results in compact clusters. The results from both the techniques are compared to suggest the most favorable technique.

The proximity matrix, for both the clustering techniques, is developed using the coupling coefficients. Coupling coefficients provide a way of denoting the strength of interdependence between modules. The coupling coefficient matrix that is formulated in this research for each module pair is referred to as the 'interdependence matrix'. Both partitional clustering and agglomerative clustering techniques, calculate the distance between the data points (observations) and group the observations based on the distances between the individual observations and the clusters. The formation of pair of cluster(s) is defined by the distance function between the clusters or the individual observations. The interdependence matrix contains coupling coefficient data. The interdependence matrix is not used directly for

cluster analysis since it does not provide the correct measure of distance between module pairs. A sample interdependence matrix for a software system consisting of 5 modules is shown in Table 3.

Table 3 Sample interdependence matrix for a system containing 5 modules

Module	One	Two	Three	Four	Five
1	0	1	2	0	5
2	1	0	4	5	3
3	2	4	0	4	2
4	0	5	4	0	1
5	5	3	2	1	0

Consider two modules from Table 3; module 1 and module 5. The coupling coefficient for this pair of module is 5, which implies that module 1 is highly coupled with module 5 and hence these modules should be present in one cluster. If the interdependence matrix is directly used in cluster analysis, then a value of 5 will indicate greater distance between module 1 and module 5 leading to the placement of both the modules into separate clusters. In order to avoid this, the research proposes the conversion of interdependence matrix into a matrix suitable for use by the clustering methods, referred to as 'distance matrix'. The distance matrix consists of values that can be used directly as input to the clustering method.

The conversion of interdependence matrix into distance matrix is done by using transformation functions. The two transformation techniques that are implemented to evaluate the clustering performance are linear translation and direct inverse.

Linear translation technique uses the highest weight assigned to the coupling coefficient in the interdependence matrix to form the linear translation equation. The highest weight is used as the reference for this transformation. The linear translation is done using the following equation,

(-1) * x + (highest weight) = (-1) * x + 5

The variable x denotes the coupling coefficient for a module pair obtained from interdependence matrix. Table 4 shows the conversion of the sample interdependence matrix, shown in Table 3, into distance matrix using linear translation.

Table 4 Sample distance matrix using linear translation for a system containing 5 modules

Module	One	Two	Three	Four	Five
1	0	4	3	5	0
2	4	0	1	0	2
3	3	1	0	1	3
4	5	0	1	0	4
5	0	2	3	4	0

This research provides the flexibility for change in the weight scale. The highest weight scale value can be changed depending on the highest coupling coefficient of the module pair in the interdependence matrix.

Inverse Transformation is formulated using the equation,

(1/x)

The variable x denotes the coupling coefficient for a module pair obtained from interdependence matrix. In situations where the coupling coefficient value is zero, the resultant of inverse transformation is considered to be a large number (example 9999), since 1/0 is not defined. Table 5 shows the conversion of the sample interdependence matrix, shown in Table 3, into distance matrix using direct inverse transformation.

Table 5 Sample distance matrix using direct inverse for a system containing 5 modules

Module	One	Two	Three	Four	Five
1	0	1	0.5	99999	0.2
2	1	0	0.25	0.2	0.33
3	0.5	0.25	0	0.25	0.5
4	99999	0.2	0.25	0	1
5	0.2	0.33	0.5	1	0

The transformation function is applied to each and every module pair value of the interdependence matrix.

3.3.6 SAS: Clustering Tool

SAS version 9.2 is used for implementing the clustering procedures. The SAS 9.2 package offers clustering procedures to implement both hierarchical and disjoint clusters. The VARCLUS procedure ("Overview: Clustering Procedures," 2010) is used to create disjoint clusters and the CLUSTER procedure is used to generate agglomerative hierarchical clusters.

SAS generates the output of hierarchical clustering in the form of a tree structure. This research analyzed and divided (cut) the tree structure at suitable levels to form the required number of clusters. A sample tree structure for a system consisting of 5 modules is shown in Figure 1.

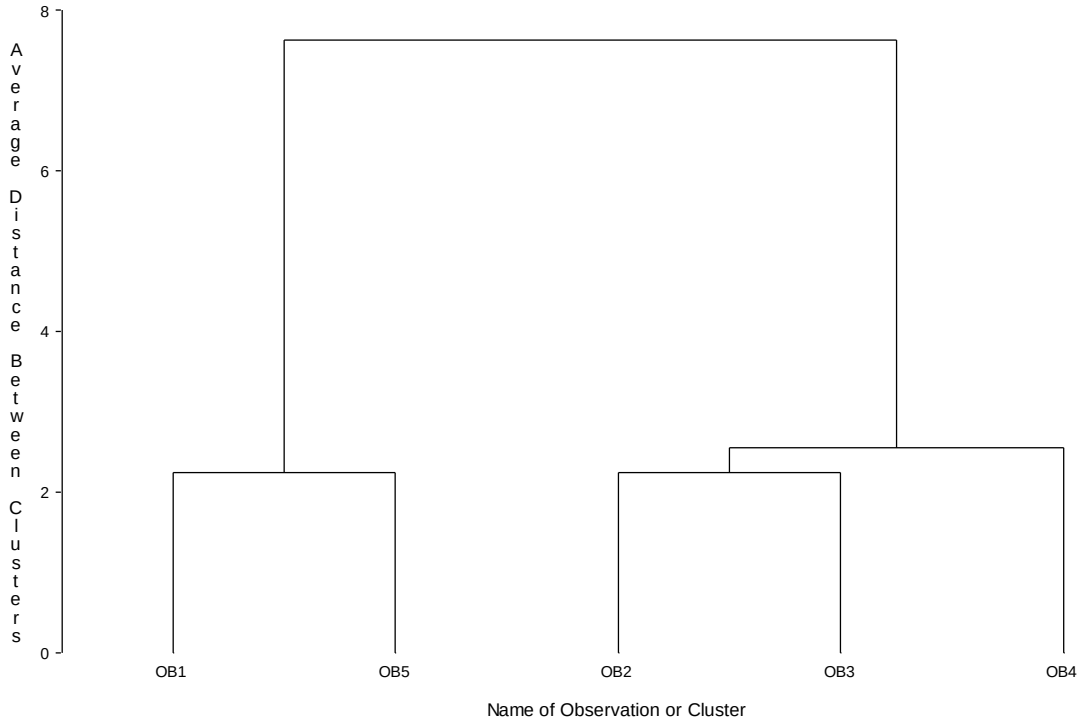

Figure 1 SAS agglomerative clustering result for a system containing 5 modules

SAS produces the output of the partitional clustering procedure in the form a table displaying the number of clusters generated and the members present in each cluster. The table generated for sample system consisting of 5 modules using SAS partitional clustering is shown in Figure 2.

3 Clusters		R-squared with			
Cluster	Variable	Own Cluster	Next Closest	1-R**2 Ratio	Variable Label
Cluster 1	Two	0.9778	0.7445	0.0869	Two
	Four	0.9778	0.9000	0.2222	Four
Cluster 2	Three	1.0000	0.7473	0.0000	Three
Cluster 3	One	0.9674	0.8921	0.3019	One
	Five	0.9674	0.7656	0.1390	Five

Figure 2 SAS partitional clustering result for a system containing 5 modules

The R-square value of a variable in Figure 2 indicates how well separated it is from the nearest cluster. The column labeled 1-R**2 Ratio in Figure 2 displays the ratio of (1-R-squared with Own Cluster)/ (1-R-squared with Next Closest). Smaller values of 1-R**2 ratio indicate good clustering.

SAS procedures are implemented for module count of 5, 10 and 15. The distance matrix for cluster analysis is generated using linear translation and direct inverse transformation. These matrices are then used as inputs for hierarchical and partitional clustering algorithms. Since the SAS clustering procedures do not offer a direct method to limit the number of members within a cluster, '*maxclusters*' feature that limits the number of clusters being formed is used. The results from the clustering techniques are observed and the lowest and highest number of members in each group is calculated. This count serves as the upper bound and lower bound of the cluster. The upper bound refers to the maximum number of members in a cluster and the lower bound refers to the minimum number of members in a cluster. The upper bound and lower bound count obtained from clustering method is used as

input to the Linear Programming model. The module assignments that are generated as a part of the Linear Programming model; using this upper bound and lower bound values, are compared with that of the module assignments suggested by clustering technique.

CHAPTER 4 RESULT ANALYSIS

The quantitative data that is used for conducting this research is derived from the coupling coefficients between each pair of modules. Coupling coefficients measure the degree of coupling that exists between the pair of modules. Firstly, the coupling categories are identified and each identified category is assigned a weight depending on the intensity of the coupling (obtained from Table 1). Secondly, the cumulative weight is calculated by adding all the coupling weights for the module pair. This aggregated weight is used as the coupling coefficient between that module pair. This process is repeated for all the module pair combinations. The coupling coefficients for module pairs are represented in a matrix form (Refer Table 3). This serves as the sample data for the Linear Programming model. The interdependence matrix consists of coupling coefficients for every module pair. The matrix shown in Table 3 represents an example of interdependence matrix for a system consisting of 5 modules. The diagonal divides the interdependence matrix into two symmetrical halves. Thus only the upper-half of the interdependence matrix (Refer Table 6) is used for formulating the Linear Programming model.

Table 6 Sample interdependence matrix for a system containing 5 modules (upper half matrix)

Module	One	Two	Three	Four	Five
1		1	2	0	5
2			4	5	3
3				4	4
4					1
5					

The conversion of interdependence matrix into distance matrix, referred to as transformation technique in this research, is essential for the purpose of clustering. Coupling coefficients for module counts of 5, 10 and 15, is processed using Linear Programming and clustering technique. Because of the long computational times required by the Linear

Programming model, the maximum module count processed in this research was limited to 15. The number of binary constraints increases significantly with the number of modules and this constrains the computational feasibility of the Linear Programming model to compute the results. (Refer Table 7)

Table 7 Increase in binary constraints with module count

Module Size	Number of Constraints
5	10
10	120
15	455
18	816
20	1140
25	2300

The module assignments are obtained using Linear Programming and Cluster Analysis. These assignments obtained from both the techniques are compared on the basis of the objective function values. The objective function values are compared based on the type of clustering algorithm (partitional and agglomerative clustering) and transformation function (linear translation and direct inverse). Figure 3 shows the results obtained from Linear Programming and cluster analysis for a sample software system consisting of 5 modules. The percentage closeness column is the objective function value obtained from SAS as a percentage of objective function value obtained from Linear Programming (LP). For example, 44.44% of the objective function value obtained from LP, which is 18, is equal to 8, the objective function value obtained from SAS. The percentage closeness value is a measure of how close the SAS result is to the result obtained from LP.

Transformation	Lower Bound	Upper Bound	Clusters		Objective Function Value		% Closeness
			SAS	LP	SAS	LP	
Partitional Linear Translation	2	3	1,2,4 3,5	2,3,4 1,5	8	18	44.44
	1	2	1,5 2,4 3	1,5 2,4 3	10	10	100
Partitional Direct Inverse	2	3	1,4,5 2,3	2,3,4 1,5	10	18	55.55
	1	3	1,2,4 3 5	1,5 2,4 3	6	10	60
Agglomerative Linear Translation	1	3	1,5 2,3,4	1,5 2,3,4	18	18	100
Agglomerative Direct Inverse	1	3	1,4 2,3,5	1,5 2,3,4	9	18	50

Figure 3 Module assignment results for system containing 5 modules

Figure 7 and 8 in Appendix summarize the results obtained from Linear Programming and cluster analysis for module counts of 10 and 15 respectively. The following section summarizes the findings of this research.

4.1 Inferences

This section describes the inferences obtained by conducting this research.

4.1.1 Inference #1

The percentage closeness value measures how close the clustering results are to the ones obtained from Linear Programming. It is observed from Table 8 that, on an average, the percentage closeness of the agglomerative clustering technique is more than the partitional clustering. Thus agglomerative clustering is more favorable over partitional clustering technique for obtaining the optimal module assignments.

Table 8 Percentage closeness in objective function value: Clustering Vs LP

Module Size	Percentage Closeness in Objective Function Value	
	Partitional Clustering	Agglomerative Clustering
5	44.00	100.00
	100.00	50.00
	55.55	-
	60.00	-
Average	64.89	75.00
10	81.01	95.77
	52.85	100.00
	63.04	44.89
	69.23	72.85
Average	66.53	78.38
15	72.28	100.00
	75.94	73.45
	53.90	90.96
	54.43	-
Average	64.14	88.14
Overall Average	65.19	80.50

Figure 4 shows a plot of percentage closeness in objective function value versus the module count. As the module count value increases the percentage closeness of agglomerative clustering technique increases. This implies that, as module count increases agglomerative clustering partitions modules into groups which are more closely aligned to Linear Programming module partitioning results.

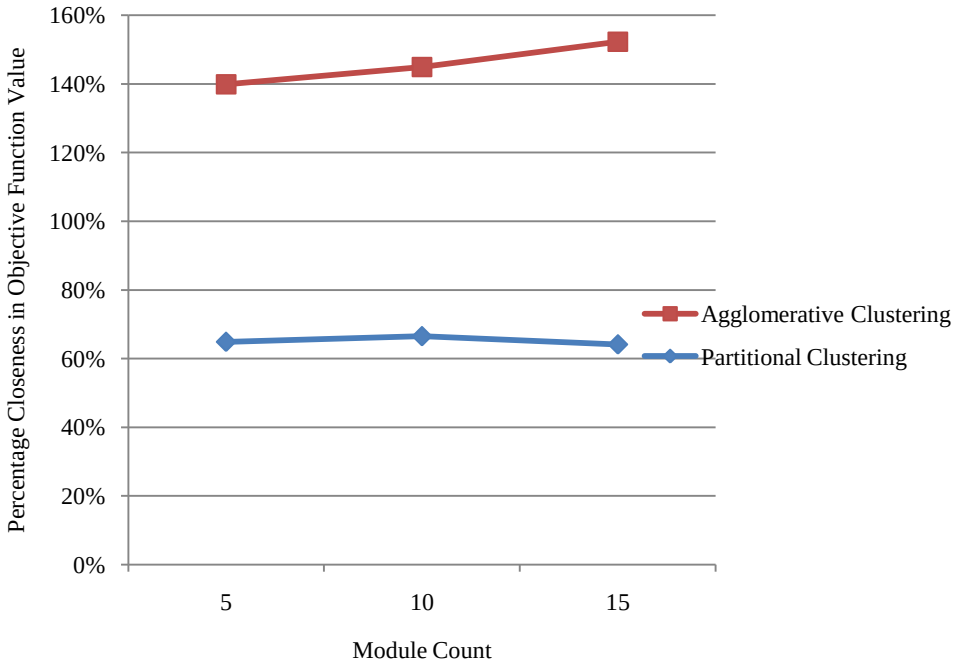

Figure 4 Percentage closeness in objective function value: Clustering Vs LP

4.2.2 Inference #2

Table 9 Percentage difference in objective function value: Clustering Vs LP

Module Size	Percentage Difference in Objective Function Value	
	Partitional Clustering	Agglomerative Clustering
5	55.00	0.00
	0.00	50.00
	44.44	-
	40.00	-
Average	34.86	25.00
10	18.00	4.00
	47.00	0.00
	37.00	55.00
	31.00	27.00
Average	33.25	21.50
15	28.00	0.00
	24.00	27.00
	46.00	9.00
	46.00	-
Average	36.00	12.00
Overall Average	34.70	19.50

The average value of the percentage difference in objective function decreases with the increase in the number of modules for partitional clustering and agglomerative clustering. The percentage difference measures the amount of change in objective function value of clustering techniques with that of Linear Programming. Figure 4 shows a plot of percentage difference in objective function value versus the module count. As the module count increases the percentage difference of agglomerative clustering technique decreases more rapidly as compared to partitional clustering. This implies that, as module count increases agglomerative clustering generates module assignments which are more closely aligned to Linear Programming results.

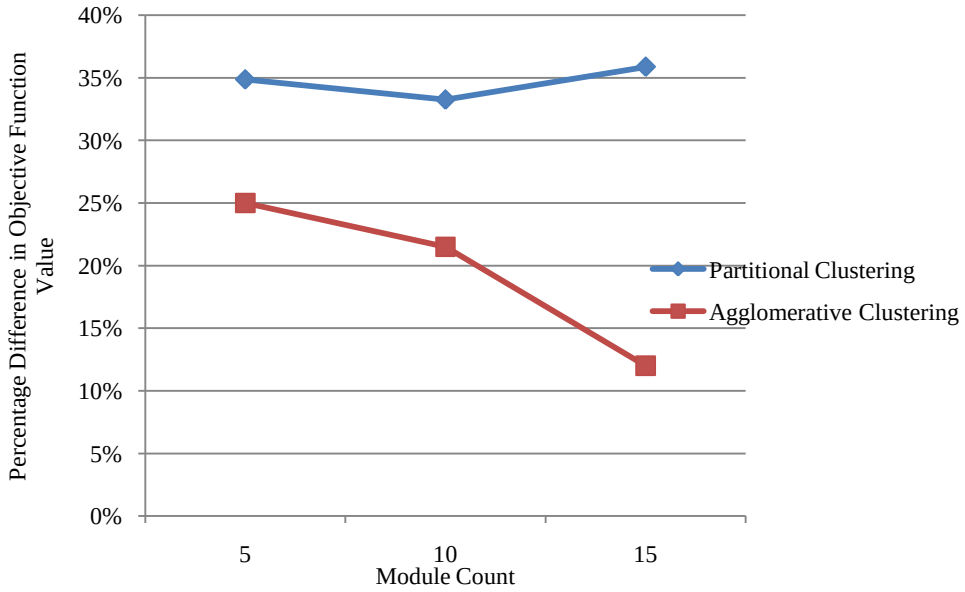

Figure 5 Percentage difference in objective function value: Clustering Vs LP

4.4.3 Inference #3

Table 10 shows that the change in the transformation function does not drastically change the percentage difference in performance of the clustering techniques. Among the two techniques, linear translation provides the lesser percentage difference in objective function value; see Figure 5, when compared to direct inverse technique.

Table 10 Percentage difference in objective function value: Transformation techniques Vs LP

Module Size	Percentage Difference in Objective Function Value	
	Linear Translation	Direct Inverse
5	55.00	44.00
	0.00	40.00
	0.00	50.00
Average	18.33	44.67
10	19.00	37.00
	47.00	31.00
	55.00	27.00
	0.00	-
Average	30.25	31.67
15	28.00	46.00
	24.00	46.00
	0.00	9.00
	27.00	-
Average	19.75	33.67
Overall Average	22.78	36.67

Thus linear translation is more suitable when an optimal solution is required.

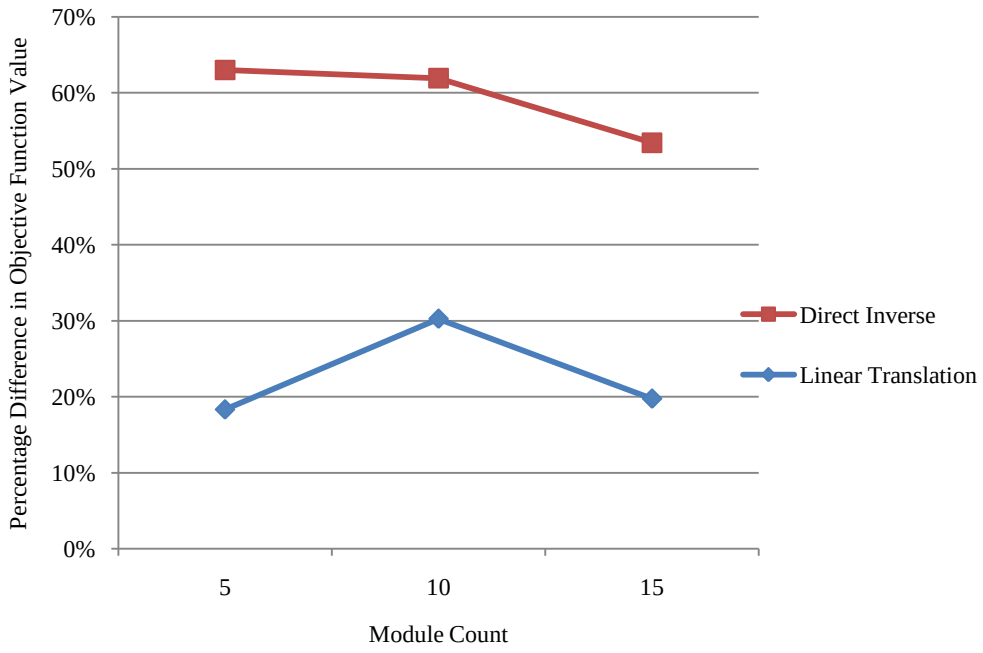

Figure 6 Percentage difference in objective function value: Transformation techniques Vs LP

Linear Programming model offers a method for obtaining the most favorable (optimal) module assignments. Hence, the results obtained from clustering technique are compared with that of the Linear Programming model. The objective function value of the Linear Programming model acts a reference for measuring the performance of the clustering technique. The percentage closeness in objective function value, shown in Table 8, and percentage difference in objective function value, shown in Table 9, serve as the metrics for comparing the closeness of results obtained from cluster analysis with that of Linear Programming. The percentage closeness in objective function value is the cluster analysis objective function value expressed as a percentage of the Linear Programming objective function. The higher the percentage closeness value, the better is that clustering technique. Table 10 displays the percentage difference values calculated from linear translation and direct inverse transformation methods.

Cluster Analysis offers a heuristic approach to cluster generation while Linear Programming model is a mathematical programming technique offering best solutions under pre-defined set of operating conditions. Partitional clustering and agglomerative clustering are the two cluster analysis techniques that were implemented in this research. The overall average percentage closeness in objective function value of partitional clustering technique is found to be approximately 67% and that of agglomerative clustering is found to be approximately 80%. The overall average percentage difference in objective function value of partitional clustering technique is found to be approximately 33% and that of agglomerative clustering is found to be approximately 20%. The higher the value of percentage closeness and the lower the value of percentage difference, the more favorable is the technique. Thus it can be concluded that the results of agglomerative clustering match 80% of the group assignments obtained from Linear Programming. This indicates that in situations where Linear Programming model cannot be implemented for modeling group assignments,

agglomerative clustering technique can act as a decent substitute to offer results that are 80% optimal.

Agglomerative Clustering method performed better than partitional clustering method to provide the optimal solution. This can be attributed to the difference in the proximity measures that are used by each of the two techniques. The agglomerative clustering type known as average link method is used in this research, which makes use of average distances between observations, as the proximity measure while partitional clustering makes use of actual distances. Since average distances between observations provide a better measure of distances than actual distances, agglomerative clustering was found to provide better results for module assignments than partitional clustering.

4.4.4 Inference #4

Table 7 shows the effect of increase in module count on the number of constraints. Increasing constraints cause considerable increase in the computational time required for providing the module assignment solution. Since, agglomerative clustering offers, a reasonable solution to the problem at hand, and does not require higher computational time, it can be used to implement the group assignments of software modules.

CHAPTER 5 CONCLUSION

5.1 Contribution of this research

There exists a high degree of complexity in designing a model that would result in optimal group assignments of software modules due to the large number of modules in real world software applications and the constraints that limit the way these group assignments can be realized. Practitioners have always acknowledged the importance of communication and coordination in distributed software development team environments. Thus, in such development environments, it is required that the modules assigned for development are such that the demand for communication is minimal among the modules that are assigned in geographically distributed teams for development. This research proposes a method that could be used by the practitioners to implement the challenging task of dividing the set of modules into groups for assignment to teams operating remotely.

Though this research suggests a method for optimal module assignments, the Linear Programming model cannot be used for solving large problems hence the results obtained from cluster analysis needs to be used. Clustering does not offer the best solution but, offers a close to best solution for module assignments. As the module count increases, the Linear Programming model cannot be used to solve the module assignment problem. Hence, this research could not compare the Linear Programming model results with that of clustering results for larger module counts.

5.2 Practical Significance

This research offers an important insight to the software practitioners on the way the module assignments should be made. The benefits of using the methodologies offered as a part of this research are:

- optimal division of modules among software teams,
- reduced time to development,
- increased group dynamics and efficiency, and
- reduced inter-team dependency

This research results in opportunities that can be exploited to enhance the decision making capabilities of software practitioners. The study focused on the module assignments to development teams but did not take into consideration the following factors:

- the number of members within the team,
- the capabilities and skills of team members; which are considered to be equal in this research, and
- the number of available teams and the influence of team hierarchical structure on module assignments

The considerations mentioned above will significantly change the module assignments in real world scenarios. The future work in this area will quantify each of the considerations mentioned above and incorporate the same for modeling optimal module partitioning logic in software environment.

APPENDIX

Transformation	Lower Bound	Upper Bound	Clusters		Objective Function Value		% Closeness
			SAS	LP	SAS	LP	
Partitional Linear Translation	3	7	3,4,6,7,8,9,10 1,2,5	2,4,5,6,8,9,10 1,3,7	64	79	81.01
	2	5	3,6,8,9,10 1,2,5 4,7	1,3,5,6,7 2,4,8,9,10	37	70	52.85
Partitional Direct Inverse	3	4	1,3,6,10 4,5,7 2,8,9	1,4,5 2,3,6,7 8,9,10	29	46	63.04
	1	3	1,3,6 4,5,7 2,8,9 10	1,4,5 2,3,6 8,9,10 7	27	39	69.23
Agglomerative Linear Translation	4	6	1,3,4,5,6,7 2,8,9,10	1,2,4,8,9,10 3,5,6,7	68	71	95.77
	3	4	1,5,4 3,6,7 2,8,9,10	1,4,5 2,8,9,10 3,6,7	47	47	100
	1	4	1,5 4 7 3,6 2 8,9,10	1,4,5,6 2,8,9,10 3,7	22	49	44.89
Agglomerative Direct Inverse	1	6	3,4,6,7,8,9 5 2 1 10	1,3,5,7 2,4,6,8,9,10	51	70	72.85

Figure 7 Module assignment results for system containing 10 modules

Transformation	Lower Bound	Upper Bound	Clusters		Objective Function Value		% Closeness
			SAS	LP	SAS	LP	
Partitional Linear Translation	5	10	1,2,3,7,8,9,10,12,13,15 4,5,6,11,14	1,2,4,5,6,9,11,12,13,14 3,7,8,10,15	120	166	72.28
	3	4	3,8,9,13 4,5,11,14 2,7,10,15 1,6,12	1,6,12,13 2,5,9,11 3,4,14 7,8,10,15	60	79	75.94
Partitional Direct Inverse	2	6	1,2,6,8,14,15 4,7,11 9,12 3,5,10,13	1,4,6,12,13,14 2,5,9 3,7,8,10,11,15	61	113	53.9
	3	4	2,8,14,15 4,7,11 9,12 1,5,6 3,10,13	1,6,12,13 2,5,9,11 3,4,14 7,8,10,15	43	79	54.43
Agglomerative Linear Translation	7	8	1,2,5,6,9,11,12,13 3,4,7,8,10,14,15	1,2,5,6,9,11,12,13 3,4,7,8,10,14,15	152	152	100
	2	6	1,12,13,6,5,11 2,9 3,7,15,8,10 4,14	1,4,6,12,13,14 2,5,9 3,7,8,10,11,15	83	113	73.45
Agglomerative Direct Inverse	1	10	1,2,6,9 3,4,5,7,8,11,12,13,14,15 10	1,2,4,5,6,9,11,12,13,14 3,7,8,10,15	151	166	90.96

Figure 8 Module assignment results for system containing 15 modules

BIBLIOGRAPHY

Alan Dennis, B. H. W., David Tegarden. (2005). *System Analysis and Design with UML Version 2.0 An Object Oriented Approach*

(Second ed.). New Jersey: John Wiley & Sons, Inc

Alan, S. (1986). *Encapsulation and inheritance in object-oriented programming languages*. Paper presented at the Conference proceedings on Object-oriented programming systems, languages and applications.

Anderberg.M. (1973). *Cluster Analysis for Applications*. New York: Academic Press.

Anderson, S., Williams. (1994). An Introduction to Management Science: Quantitaive Approaches to Decision Making. In (pp. 17). St.Paul,MN: West Publishing Company.

Aruna, M., Devi, M. P. S., & Deepa, M. (2008). *Measuring the Quality of Software Modularization Using Coupling-Based Structural Metrics for an OOS System.* Paper presented at the Emerging Trends in Engineering and Technology, 2008. ICETET '08. First International Conference on.

Bin, X., Hua, H., Yun, L., Xiaohu, Y., Zhijun, H., & Ma, A. (2007). *Efficient Collaborative Task Arrangement in Global Software Design via Micro-Estimation and PERT Technique.* Paper presented at the Computer Supported Cooperative Work in Design, 2007. CSCWD 2007. 11th International Conference on.

Blashfield, A. a. (1984). *Cluster Analysis*. Newbury Park, California: Sage Publications.

Briand, L., & Wust, J. (2001). Modeling development effort in object-oriented systems using design properties. *IEEE Transactions on Software Engineering, 27*(11), 963.

Brito e Abreu, F., & Goulao, M. (2001). *Coupling and cohesion as modularization drivers: are we being over-persuaded?* Paper presented at the Software Maintenance and Reengineering, 2001. Fifth European Conference on.

Brito e Abreu, F., Pereira, G., & Sousa, P. (2000). *A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems.* Paper presented at the Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European.

C.Wunsch, R. X. a. D. (2009). *Clustering*. New Jersey: A John Wiley & Sons,Inc.Publication.

Cain, J. W., & McCrindle, R. J. (2002). *An investigation into the effects of code coupling on team dynamics and productivity.* Paper presented at the Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International.

Damian, D., & Moitra, D. (2006). Global Software Development: How Far Have We Come? *IEEE Software, 23*(5), 17-19.

Darcy, D. P., Kemerer, C. F., Slaughter, S. A., & Tomayko, J. E. (2005). The structural complexity of software an experimental test. *Software Engineering, IEEE Transactions on, 31*(11), 982-995.

David, N. C., Gerald, T. P., & Frank, E. M. (1985). *Criteria for software modularization.* Paper presented at the Proceedings of the 8th international conference on Software engineering.

Edward Yourdon, L. L. C. (1975). *Structured Design Fundamentals of a Discipline of Computer Program and Systems Design* New York: Yourdon Press.

Garlan, D., Allen, R., & Ockerbloom, J. (2009). Architectural Mismatch: Why Reuse Is Still So Hard. *Software, IEEE, 26*(4), 66-69.

Hla Myat, K., Nan Si, K., & Ni Lar, T. (2005). *To Visualize the Coupling among Modules.* Paper presented at the Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings. 6th Asia-Pacific Symposium on.

Hung-Fu, C., & Lu, S. C. Y. (2009). *Decomposition and Traceability in Software Design.* Paper presented at the Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International.

Jain, A. K., Duin, R. P. W., & Jianchang, M. (2000). Statistical pattern recognition: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22*(1), 4-37.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Comput. Surv., 31*(3), 264-323.

Jones, P. (1988). *The Practical Guide to Structured Systems Design* (2 ed.). Englewood Cliffs, New Jersey: Yardon Press.

Kiczales, G. (1996). Beyond the black box: open implementation. *Software, IEEE, 13*(1), 8, 10-11.

Lieberherr, K. J., & Holland, I. M. (1989). Assuring good style for object-oriented programs. *Software, IEEE, 6*(5), 38-48.

Lionel, C. B., Jurgen, W., John, W. D., & Porter, D. V. (2000). Exploring the relationship between design measures and software quality in object-oriented systems. *The Journal of Systems and Software, 51*(3), 245.

Michael, V., & David, N. (1996). *Decoupling change from design*. Paper presented at the Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering.

Mockus, A., & Weiss, D. M. (2001). Globalization by Chunking: A Quantitative Approach. *IEEE Software, 18*(2), 30.

Myers, G. (1978). *Composite/Structured Design*. New York: Van Nostrand Reinhold.

Offutt, J., Abdurazik, A., & Schach, S. (2008). Quantitatively measuring object-oriented couplings. *Software Quality Journal, 16*(4), 489.

Overview: Clustering Procedures. (2010). *SAS/STAT(R) 9.2 User's Guide* Second. Retrieved 11 Feb, 2010, from

http://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/statug_introclus_sect001.htm

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM, 15*(12), 1053-1058.

Paul, J. C., & Jack, B. (2000). Utilizing cluster analysis to structure concurrent engineering teams. *IEEE Transactions on Engineering Management, 47*(2), 269.

Sarkar, S., Kak, A. C., & Rama, G. M. (2008). Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software. *Software Engineering, IEEE Transactions on, 34*(5), 700-720.

Sarkar, S., Rama, G. M., & Kak, A. C. (2007). API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization. *Software Engineering, IEEE Transactions on, 33*(1), 14-32.

Sarkar, S., Ramachandran, S., Kumar, G. S., Iyengar, M. K., Rangarajan, K., & Sivagnanam, S. (2009). Modularization of a Large-Scale Business Application: A Case Study. *Software, IEEE, 26*(2), 28-35.

Solver.com. (2009). Retrieved 11 Feb 2010, from http://www.solver.com/xlsplatform.htm

T.Ragsdale, C. (2007). Spreadsheet Modeling and Decision Analysis:A Practical Introduction to Management Science. In A. v. Rosenberg (Ed.), (pp. 17). Mason,OH: Thomson South-Western.

Yourdon, P. C. a. E. (1991). In *Object-Oriented Design* (pp. 128): Englewood Cliffs NJ:Yourdon Press.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to those who helped me with this research work. First, Dr. Zhengrui Jiang for his guidance, patience, support for this research and my graduate journey. His insights into the research topic and words of encouragement inspired me and strengthened my hopes of completing graduate education as planned. Second, I would like to thank Dr.Suzuki who helped me in formulation of this thesis and third, Dr.Crum for his support and insightful comments. Third, I would like to thank my roommates for their constructive criticism and insightful feedback while I was writing this work and my best friend for always questioning my position on this thesis and helping me unwind after long intervals of research and typing.