2006

Analysis Of Complexity And Coupling Metrics Of Subsystems In Large Scale Software Systems

Harish Ramakrishnan
University of Central Florida

ANALYSIS OF COMPLEXITY AND COUPLING METRICS OF SUBSYSTEMS IN LARGE SCALE SOFTWARE SYSTEMS

by

HARISH RAMAKRISHNAN
B.E. Bharathidasan University, 2003

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2006

ABSTRACT

Dealing with the complexity of large-scale systems can be a challenge for even the most experienced software architects and developers. Large-scale software systems can contain millions of elements, which interact to achieve the system functionality. Managing and representing the complexity involved in the interaction of these elements is a difficult task. We propose an approach for analyzing the reusability, maintainability and complexity of such a complex large-scale software system. Reducing the dependencies between the subsystems increase the reusability and decrease the efforts needed to maintain the system thus reducing the complexity of the system.

Coupling is an attribute that summarizes the degree of interdependence or connectivity among subsystems and within subsystems. When used in conjunction with measures of other attributes, coupling can contribute to an assessment or prediction of software quality. We developed a set of metrics for measuring the coupling at the subsystems level in a large-scale software system as a part of this thesis work. These metrics do not take into account the complexity internal to a subsystem and considers a subsystem as a single entity. Such a dependency metric gives an opportunity to predict the cost and effort needed to maintain the system and also to predict the reusability of the system parts. It also predicts the complexity of the system. More the dependency, higher is the cost to maintain and reuse the software. Also the complexity and cost of the system will be high if the coupling is high. We built a large-scale system and implemented these research ideas and analyzed how these measures help in minimizing the complexity and system cost. We also proved that these coupling measures help in re-factoring of the system design.

This thesis is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake.
It is also dedicated to my mother, who taught me that even the largest task can be accomplished if it is done one step at a time.
It is also dedicated to my brother and sister-in-law who motivated me to do a good work.
It is also dedicated to all my family members and my friends, who extended me their support, love and enthusiasm.

I specially dedicate to my shining, energetic and lovable daughter, Sharadha, who filled my heart with her lovely smiles.

Without these things, this thesis could not have been possible.

ACKNOWLEDGMENTS

I would like to extend my gratitude to Dr. Ronald Eaglin for his enthusiastic supervision and support of my thesis work. I also extend my thanks to Dr. Joseph S. Berrios for helping me in starting my thesis and for supporting me through out my work. I am grateful to Dr. Essam Radwan for his extra-ordinary support through out my masters program. I also thank Dr. Chris Bauer for being in my committee amidst his busy schedule.

I would like to thank my friend Pradeep Aleuri for helping and supporting me through out the project. I would like to extend my thanks to Karla Alvarado who supported my thesis with enthusiasm. I also thank all of my team mates for their support in the project.

I thank Viji and Karthik for giving their endless love, encouragement and support. I thank all my roommates for their encouragement and understanding. I thank all my friends for giving me good time and support.

Finally, I am forever indebted to my parents and my family for their understanding, endless patience and encouragement when it was most required.

Last but not the least, this work would have never been completed without the blessings of GOD.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ACRONYMS/ABBREVIATIONS

OLC	Object Level Coupling
CLC	Class Level Coupling
CBO	Coupling Between Objects
RFC	Response For Class
PIM	Polymorphistically Invoked Method
NAS	Number of Associations
MPC	Message Passing Coupling
NSI	Number of Static Invocations
DAC	Data Abstraction Coupling
ADT	Abstract Data Type
COF	Coupling Factor
CA	Class – Attribute interaction
CM	Class – Method interaction
MM	Method – Method interaction
CDM	Coupling Dependency Metric
CSL	Coupling at Subsystem Level
SC	Subsystem Coupling
CBS	Coupling Between Subsystems

CCMS	Coupling Complexity Measure of Large Scale System
NPI	Number of Polymorphistically Invoked methods
Ns	Number of Subsystems
SCINET	Seminole County Integrated Network
DBM	Database Manager
UMS	User Management System
CCM	Category Code Management
HFS	Help and Feedback System
CEB	Code Enumerations Builder
WFM	Work Flow Management
DM	Document Management
LM	Letter Management
BRM	Business Rules Manager

CHAPTER ONE: INTRODUCTION

1.1. Qualities of Good Software in Different Perspectives:

The expectation of software varies based on the views of users, developers and the managers. According to the end-users software is good based on the following.

- Usefulness — does it do what the user wants?
- Performance — speed, size.
- Correctness — does it meet its specification?
- Reliability — does it do what the user wants most of the time?
- Robustness — does it respond well to error/failures of other systems?
- Usability — is it easy to use?
- Interoperability — does it conform to relevant standards/formats etc.?
- User documentation.

The current and the future developers care about

- Correctness — internal documentation consistent with itself and the code.
- Manageability — is the product easily maintainable at low cost?
- Changeability — can the changes be made reasonably?
- Understandability — is code & documentation understandable?
- Reusability — Can components be reused in other projects?

The managers care about

- Cost — production and maintenance.

- Timeliness — When will it be ready?
- Traceability — Can the progress of the production be monitored?

1.2. Large Scale System and Complexity:

A large scale software system is a system that contains a huge number of elements such as classes, interfaces, enumerations, web services etc. To perform the expected function, these elements interact with each other. Since the number of elements is humongous, the interaction is also high. These interactions make the system, hard to understand, hard to maintain, hard to change and thus making the system highly error prone and costly. Such a hard system is called a complex system. Since the large scale system is always hard, it is always complex.

More over, there are no adequate criteria and guidelines for making a good design for a large scale system. Design patterns are very much helpful in small systems but for a large scale system, a single design pattern never fits in. A large number of design patterns need to be combined and used. Such a usage of design patterns itself add to the system cost. There are no well proven and adequate metrics to measure the complexity of the large scale system. If the complexity of the system is measured, it would be easy to find the portion of the system that increased the complexity and modify it to reduce the complexity. Due to varying requirements of the software, change is unavoidable. These make the large scale systems always complex.

1.3. Coupling between Sub-systems:

When considering a large scale system, as already discussed, we will have a huge number of classes and methods. Considering the class level coupling criteria and class level coupling measures, it will be very difficult to determine the cost and complexity of the entire system. In most of the cases, cost to calculate the coupling measure will be higher. In physics, there is a first law of thermodynamics which states "The total energy of the system plus the surroundings is constant." Similarly, the law of complexity can be stated as "*The underlying complexity of software is constant*." That is, no matter how good is the design, the complexity of a system never goes away. But the complexity of the system can be hidden by abstraction. Hidden complexities do not affect the system cost. We propose an approach to reduce the cost of measuring the coupling and complexity of the system using this law. The better design would be to combine the tightly coupled cohesive classes and to abstract them as subsystems. Doing so will not only improve the reusability of the subsystem but also eases the estimation of the maintenance cost of the system. A subsystem can be defined as a chunk of cohesive classes that can be compiled, linked and tested separately. For the efficient working of the whole system, the subsystems should be glued together, i.e. there should be some level of dependency between the subsystems. But this dependency between different subsystems should be minimal to improve the reuse of the subsystem. According to Don Batory and Sean O'Malley [1] a hierarchical design of a system will help reducing the level of coupling between the subsystems.

The subsystems can be viewed as a hierarchy of systems, where in the subsystems assembled in different levels can be coupled from higher level to lower level and the subsystems assembled in the same level should not be coupled and there can be no coupling from lower level

to higher level. For e.g. in Figure 1 there can be coupling between the subsystem - 1 and subsystem - 2 but the subsystem - 2 and subsystem - 3 and subsystem - 4 should not be coupled.

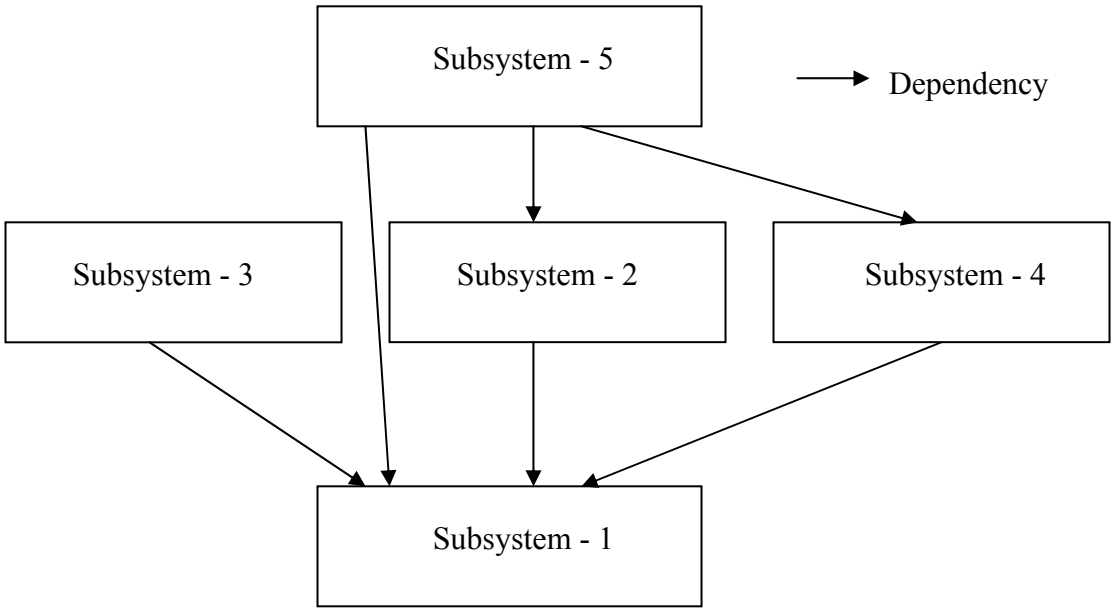

Figure 1: Sub-Systems viewed as a hierarchy

The hierarchical design not only reduces the coupling and complexity of the system but also reduces the cost needed to measure the coupling as there is no need to measure the coupling between the subsystems at the same level. We only need to consider measuring the coupling of a subsystem from a higher hierarchical level to the lower hierarchical level thus reducing the cost of measuring the coupling and complexity.

1.4. Coupling and Reusability:

In all the three perspectives discussed in the previous section, software maintenance, performance and reusability are considered to be important for good software. Re-use based

software development is one of the most promising technologies in software development. The software development that utilizes the software reuse gets the benefits of reduction in the development as well as testing cost and time. It is possible to create customized systems economically by building only the parts that are application-specific. Unnecessary reinvention of technology is thereby avoided. A reusable software component can be anything such as program code, design specifications, plans, documentation, expertise and experience, and any information used to create software and software documentation. A classical, but largely unrealized, goal of software engineering is software component technologies. Such technologies are envisioned to exploit large-scale reuse, and elevate the granularity of programming to the subsystem level. The major steps in the Software Reuse Procedure includes,

1. *Component extraction* - Extracting the component with high reuse potential and high quality from existing software systems.
2. *Component standardization* - Packaging a reusable software component in a standard format.
3. *Component classification* – Classifying the different reusable components into libraries based on their attributes.
4. *Component retrieval* – Reusing the suitable component out of the library based on the attributes needed.
5. *Component adaptation* – Adapting the reuse component to the new application.

The key step is to extract the software components with high reuse potential and high quality i.e. Component Extraction, as this step may affect the steps that follow directly. The most serious problem with the Component Extraction is an overly high degree of dependency

throughout the whole software product. It is therefore difficult to reuse selected packages or subsystems in another software system without making many changes. If we pick up one package for reusing, we get almost the whole system because of these direct or indirect dependencies. The recompilation of large parts of the system (eventually the whole system itself) for only minor modifications in any of the subsystems indicates the effect of dependency between the different subsystems. Therefore, it is hard to extract a single subsystem as a reusable component out of the whole system. This dependency is called Coupling. Measuring the coupling will give a chance to predict the reusability of a module or to predict the change needed to make it reusable.

1.5. Coupling and Software Maintenance:

Coupling is considered by many to be an important concept in measuring design quality. Coupling was introduced by Constantine et al. [2] in the 1960s as a heuristic for designing modules. They observed that programs that were easier to implement and change were those composed of simple independent modules, and coupling was a way to determine the dependency of a module with the others in the system. Constantine and other researchers' main concern were in reducing the cost of debugging, which is a dominant cost of software. The issue here is that making a change to fix a fault, may in fact introduce a new fault somewhere else. In more recent times, studies reported in software engineering texts suggest that software maintenance cost is the dominant cost, with varying figures reported but typically around 70% of the lifetime cost of the product as shown in Figure 2. Maintenance has been characterized in a number of ways, but generally the part associated with removing faults from code tends to be the smaller part,

whereas costs associated with adding new functionality or adapting to a new environment dominates. It is important to have the ability to predict which modules in a software product are likely to be fault-prone so that corrective action can be taken. Measuring the coupling will help in predicting the occurrence of runtime failures. One consequence of reducing the coupling between modules is the belief that it will reduce the likelihood of the impact of the change in one module on another module.

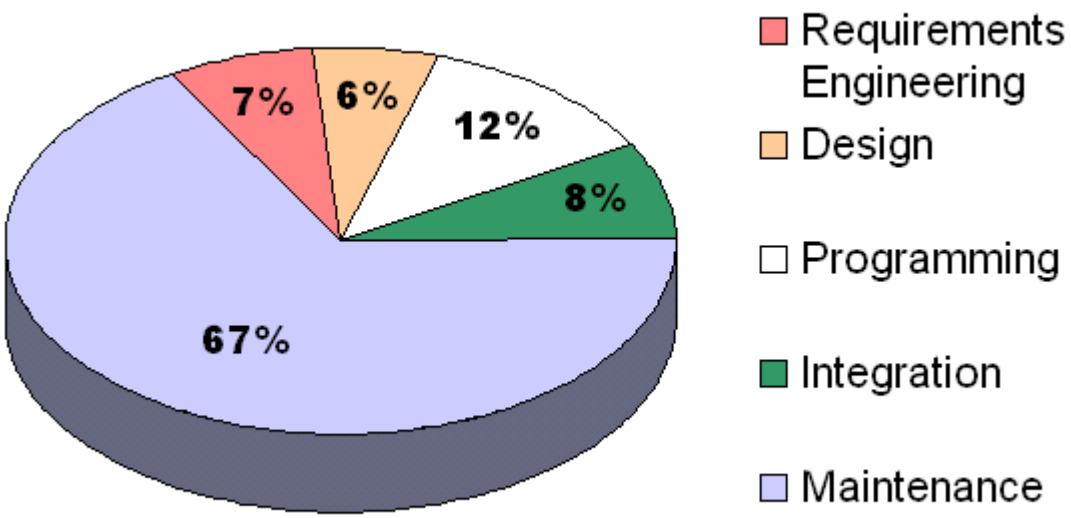

Figure 2: Approximate Relative Costs of the Phases of the Software Life Cycle [3]

From the above discussion, we conclude that for effective reuse and maintenance of a subsystem, it should be less coupled with other subsystems. In other words, a highly coupled subsystem is hard to maintain and reuse in other systems. Hence the goal of almost every software development will be to remove the coupling between the subsystems. But removing the coupling altogether is almost impossible, because for the effective functioning of the system,

cohesion, the different subsystems should interact with each other. This interaction causes some level of coupling between them. Hence a certain level of coupling is unavoidable between the subsystems for the efficient functioning of the system.

From the above discussion, we conclude that an important quality attribute of software that the concept of coupling can help us with is the management of change, or modifiability and reusability. We evaluated different ways of measuring the coupling and found that there is not much work to measure coupling in a subsystem level in a large scale system. We present a methodology to cost effectively measure coupling between different subsystems in a large scale system based on the foundation laid by the past research works.

1.6. Thesis Overview:

The general overview of coupling reported in the literature of Software Engineering is presented in chapter 2. It mainly involves the different methods of measuring coupling and the effect of the coupling on software reuse and maintenance and different methodologies to reduce the coupling.

Chapter 3 deals with the development of a methodology to measure the coupling between the subsystems in a large-scale systems and presents a metric to measure the coupling at the subsystem level in a large scale software system.

Chapter 4 deals with the analytical evaluation of the new metric presented in chapter 3 to evaluate the effect of coupling in maintenance cost and the reuse capability.

Chapter 5 gives the experimental analysis and results of the coupling measures by applying them to SCINET, a large scale software system. This chapter also shows how these

coupling measures help in designing the large scale system and there by reducing the maintenance effort needed and shows how these coupling measures are helpful in increasing the subsystem reusability.

Chapter 6 concludes this thesis work and gives some of the future works that can be done to improvise the metrics given in this work and to help reduce the coupling and improve the reuse capability of subsystems and reduce the maintenance cost.

A list of references which helped us in understanding the literature of the coupling and that helped us in defining new metrics for coupling measures is presented at the end of this work.

CHAPTER TWO: LITERATURE REVIEW

2.1. Coupling – An Overview

High quality software design, among many other principles, should obey the principle of low coupling. Stevens et al., who first introduced coupling in the context of structured development techniques, define coupling as "the measure of the strength of association established by a connection from one module to another." Therefore, the stronger the coupling between modules, i.e., the more integrated they are, the more difficult these modules are to understand, change, and maintain and thus the resulting software system will be more complex. While software reuse demands for separation of the subsystems, the software development process demands for integration of subsystems. Consequently, achieving both effective software reuse and an efficient software development process seem hard to unite. Hence, higher the coupling between two packages, the reuse capability of those packages is lower as it is hard to separate two highly coupled packages. Similarly, higher the coupling, more complex is the system is and it is hard to maintain such system.

2.2. Coupling in Procedure-Oriented Systems:

2.2.1. Levels of Coupling:

Myers, Glenford J [4] defined six distinct degrees of coupling to measure the interdependence among the modules.

1. Data Coupling - Two modules are data coupled if they pass data through scalar or array parameters.
2. Stamp Coupling -Two modules are stamp coupled if they pass data through a parameter that is a record. Stamp coupling is perceived as worse than data coupling because any change to the record will affect all of the modules that refer to that record, even those modules that do not refer to the fields that are changed.
3. Control Coupling -Two modules are control coupled if one passes a flag value that is used to control the internal logic of the other.
4. External Coupling - Two modules are external coupled if they communicate through an external medium such as a file.
5. Common Coupling - Two modules are common coupled if they refer to the same global data.
6. Content Coupling - Two modules are content coupled if they access and change each other's internal data state or procedural state.

Offutt et al. [5] include 'no coupling' as the zeroth level. If two modules are coupled in more than one way, they are considered to be coupled at the highest level. Page-Jones [6] also introduced the notion of tramp coupling, where data flows through many intermediate modules i.e. from the module where the data are defined to where they are used. This differs from the other coupling levels in that it measures the coupling among many modules instead of just two modules. Tramp coupling is a mostly form of data coupling, but can also be stamp or control coupling. Though the use of structured design is outdated by the object-oriented designs, these

coupling levels are not limited to the structured design. They are modified by the researchers to accommodate the object oriented design.

2.3. Coupling in Object-Oriented Systems:

Object-oriented design and analysis incorporate the data and its functionality into objects thereby reducing the coupling between objects. But still object-oriented mechanisms does not guarantee minimum coupling. Coad and Yourdon, in their books, introduced the concept of coupling in the Object Oriented design. As opposed to procedure-oriented systems where the module is the only one subject of interest, in object-oriented systems there are several subjects of interest, such as methods and object classes. Coupling properties of the various subjects of interest are not independent from each other i.e. the coupling properties between methods of different object classes highly influence the coupling properties between these object classes. Budd [7] gives the classical concepts of coupling in terms of object- oriented languages. The terms internal data coupling and Global data coupling seem to fit into the object-oriented paradigm. But in the literature, it is believed data coupling is not of much importance in object-oriented paradigm as the main communication mechanism is message passing rather than data sharing.

2.4. Coupling Frameworks:

Eder et al. [8] identify three different types of relationships, interaction coupling, component coupling, and inheritance coupling. According to them, methods are coupled by

invocation of each other and/or by sharing data. Thus it is possible to have an interaction relationship among the methods. Next to methods, also object classes have to be analyzed in terms of relationships with each other, and thus in terms of coupling properties. In addition to the interaction relationships, the object classes may also have component relationships and inheritance relationships with each other. From these three relationships, the three different coupling types mentioned above are identified. Further, they divide them into different degrees of coupling.

2.4.1. Interaction Coupling:

It is almost similar to the original definition of coupling and uses the same degrees proposed by Myers, Glenford J [4]

2.4.2. Component Coupling:

Two components c and c' are component coupled if c' is the type of either

a. an attribute of c, or
b. an input or output parameter of a method of c, or
c. a local variable of a method of c, or
d. an input or output parameter of a method invoked within a method of c

They defined four degrees of component coupling namely,

1. Hidden: If a class contains a cascaded invocation of methods, the type of object returned by last invoked function need not be explicit in the interface or body of the invoked class.
2. Scattered: case c. and d. specified above

3. Specified: case a. and b. specified above
4. Nil: No component coupling

2.4.3. Inheritance Coupling:

Two classes are inheritance coupled if one class is an ancestor of another class. Four degrees of Inheritance coupling are,

1. Modification: Information inherited from the parent class is modified or deleted (either signature modification or implementation modification) in a manner which violates the inheritance rules
2. Refinement: Information inherited from the parent class is modified (either signature refinement or implementation refinement) according to predefined rules but still affects the polymorphism.
3. Extension: New methods and attributes added to the inheriting class
4. Nil: No inheritance relation between two classes.

Hitz and Montazeri [9] derived two different types of coupling: object level coupling (OLC) and class level coupling(CLC) which are determined by the state of an object and the state of an object's implementation, respectively. The object level coupling refers to the dynamic dependencies between the objects at the runtime and the class level coupling refers to the static dependencies between the classes at the design time. The class level coupling is more important when compared to the object level coupling when considering maintenance and reusability as the class level coupling affects the reuse capability and the object level coupling affects the testing and debugging. The strength of CLC depends on stability of server class,

access type, and access scope. The strength of OLC depends on access type, access scope, and interface complexity. Briand et al. [10] constitute coupling as interactions between classes. The strength of the coupling is determined by the type of the interaction namely, class-attribute, class-method, method-method interactions, and the relationship between the classes. Their framework focuses solely on the coupling relationships available during the high level design phase. Briand et al. [11] compares these three frameworks of coupling. In the table below, each row represents one mechanism that constitute coupling and an "X" indicated that the mechanism is covered by the framework in the respective column.

Table 1: Comparison of Mechanisms that constitute Coupling

#	Mechanism	Eder et al.	Hitz, Montazeri	Briand et al.
1	Methods share data	X		
2	Methods references attribute		X	
3	Method invokes method	X	X	X
4	Method receives pointer to method			X
5	Class is type of a class' attribute	X	X	X
6	Class is type of a method's parameter or return type	X	X	X
7	Class is type of a method's local variable	X	X	
8	Class is type of a parameter of a method invoked from within another method	X		
9	Class is ancestor of another class	X	X	

Table shows the comparisons of different mechanisms constituting to coupling in three different coupling frameworks. 'X' indicates that the corresponding mechanism is dealt in the corresponding framework

Briand et al. [11] also gives a unified framework to design a coupling metric. They consider 6 criteria to design a coupling metric. The six criteria of their framework are

- Type of connection – what constitutes coupling
- Locus of impact – import or export coupling

- Granularity of the measure – domain of the measure and how to count coupling connections
- Stability of the server
- Direct or indirect coupling
- Inheritance – inheritance or non-inheritance based coupling
- Polymorphism

However these are not the sufficient conditions to be accounted for when designing a coupling metric. The results of the empirical validation of the measure should be taken into account. But the authors did not address about the empirical validation.

2.5. Design Patterns:

The use of design patterns while designing a system will help in reducing the coupling. A design pattern is nothing but a well proven method of designing software systems. Using a pattern, same results can be expected in different programming languages and different programming contexts. There are so many design patterns available. Most of the design patterns do not exactly fit the needs. Choosing the correct pattern for our design is a costly process, since it is a big decision to make. There are several patterns like Subsystem Façade pattern, Abstract Interface pattern, Inverted Association pattern, Association pattern, Director Pattern etc., which helps in decoupling, i.e., reducing the coupling between objects. Structural Facade pattern can be used effectively in systems that consist of a variety of subsystems. Although the creation of subsystems or sets of functionally related classes serves to organize the development effort, the interactions between subsystems can be complex and introduce a web of dependencies that aren't

managed easily. This pattern provides a higher-level interface that makes working with subsystems simpler. The Facade pattern can be applied productively to multi-tier architectures where a layer of controller or business process classes are introduced between the user interface classes and the data-access classes. This allows developers to design generic data-access classes, while simplifying the interactions with them from the user interface using the controller classes. This design also promotes loose coupling between the user interface and data-access classes, thereby enhancing reusability and extensibility. In a large scale system, a single design pattern will not fit the design. Hence different design patterns need to be combined to fit the system.

2.6. Existing Coupling Measures:

The strength of coupling between two classes is determined by the frequency of connections between the classes and the type of connections between the classes. Counting the frequency of connections between classes has to be resolved when defining the measures. It is important that definition of measures should be based on the intended application of the measure. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be. Chidamber and Kemerer presented a metric suite for object oriented design which is considered the base for all the other coupling metrics. We analyzed some of the inter-class coupling metrics which are relevant to our area of research.

2.6.1. Coupling Between Objects(CBO)

Chidamber and Kemerer [12] defines the measure coupling between objects (CBO) as "CBO for a class is a count of the number of other classes to which it is coupled." According to CBO, an object is coupled to another object if one of them acts on the other, i.e., methods of one use methods or instance variables of another. As stated earlier, since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. CBO can be stated as follows [11].

$$CBO(c) = \mid \{d \in C - \{c\} \mid uses(c,d) \vee uses(d,c)\} \mid \forall c,d \in C$$

$$CBO'(c) = \mid \{d \in C - (\{c\} \cup Ancestors(C)) \mid uses(c,d) \vee uses(d,c)\} \mid \forall c,d \in C \quad \textbf{(1)}$$

Class c is coupled to class d if c uses d or c is used by d where class c and d belongs to the set of classes C. A high CBO measure is regarded as a disadvantage, as it is suggested that it indicates the extent of:

- lack of reuse potential
- effort needed during maintenance
- effort required to test the class

According to the authors, the CBO metric can be used by designers and project managers as a way to track if the class hierarchy is losing its integrity, and whether different parts of a large system are having unnecessary interconnections in inappropriate places. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system.

2.6.2. Response For Class (RFC)

Chidamber and Kemerer [12] also gave another metric for measuring coupling between classes. Response for Class (RFC) is defined as the count of the set of methods that can potentially be executed in response to a message received by an object of a class. It can be formally defined as

$$RFC = | RS |$$
$$RS = \{M\} \cup_{alli} \{R_i\}$$

RS is the response set for the class and $\{R_i\}$ is the set of methods called by i. It includes all the methods directly invoked by i and all the methods invoked by these methods and so on. {M} is the set if all methods in the class. Since this could go on to a multiple level of nesting, for practical considerations, Churcher and Shepperd [13] modified this metric to take the levels of nested method invocations in to consideration. Their measure can be given as

$$R_0(c) = M(c)$$
$$R_{i+1}(c) = \bigcup_{m \in R_i(c)} PIM(M)$$

For α = 1, 2, 3… **(2)**

$$RFC_\alpha(c) = | \bigcup_{i=0}^{\alpha} R_i(c) |$$

where PIM (M) is the set of polymorphically invoked methods. RFC not only includes the direct coupling but it also accounts the indirect coupling. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system.

2.6.3. Number of Associations (NAS)

Harrison et al. [14] presented a coupling metric, Number of Associations (NAS) metric, which is defined as the number of associations of each class, counted by the number of association lines emanating from a class on an Object Model diagram. This measure therefore also includes inheritance. The NAS metric is directly collectible from design documents. NAS is similar to CBO except that in CBO, if a class uses a method of another class more than once, then the CBO metric will count each usage as a separate occurrence of coupling. NAS, on the other hand counts repeated invocations as a single occurrence of coupling. Hence, we expect values of CBO for a class to be greater than the NAS for the same class. According to the authors, NAS can be measured in the early design stage and hence helps the managers to obtain early coupling estimates. For a large scale system, measuring the coupling from object diagram is very difficult and costly as there will be a huge number of objects in a large scale system.

2.6.4. Message Passing Coupling(MPC)

Li and Henry proposed a coupling metric, Message Passing Coupling (MPC), [11, 15] which is defined as the number of method calls to other classes from within a given class. This is not to be confused with Response for Class (RFC) which measures all of the method calls, include those to the local methods. Only the method calls in local methods are counted. The authors also say "The local methods of a class constitute the interface increment." Therefore method calls in inherited methods are not counted.

$$MPC(c) = \sum_{m \in M_i(c)} \sum_{m' \in SIM(m) - M_i(c)} NSI(m, m') \quad \textbf{(3)}$$

NSI is the number of static invocations of methods not implemented in c by methods implemented in c. m is a local method in class c which invokes a method m'. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system.

2.6.5. Data Abstraction Coupling (DAC)

Data Abstraction Coupling (DAC), given by Li and Henry, [11, 15] is defined as the number of abstract data types (ADTs) (say another class in the system) defined in a class. The number of variables having an ADT type may indicate the number of data structures dependent on the definitions of other classes. 'T (a)' is the type of attribute and 'a' is the attribute and C is the set of all classes in the system. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system. DAC can be given as follows

$$DAC(c) = |\{T(a) \mid a \in A_i(c) \wedge T(a) \in C\}| \quad \textbf{(4)}$$

2.6.6. Coupling Factor (COF)

The coupling factor (COF) metric, given by Abreu and Esteeves, [11, 16] represents the actual number of client-server relationships between classes that are not related via inheritance to the maximum possible number of such client-server relationships. It is normalized between 0 and 1 to allow comparisons between systems of different sizes. To obtain the actual number of couplings, the metric goes through each class in the system (considering all its methods and attributes) and finds its relationships to all other classes in the system as described by the

numerator of the model. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system.

$$COF(c) = \frac{\sum_{c \in C} |\{d \mid d \in C - (\{c\} \cup Ancestors(c)) \wedge uses(c,d)\}|}{|C|^2 - |C| - (2\sum_{c \in C} |Descendents(c)|)} \quad \textbf{(5)}$$

2.6.7. Metric Suite of Briand et al

Briand et al. [11] implemented a unified framework for measuring coupling in object oriented systems. For each class, they counted

- the number of class-attribute/class-method/method-method interactions
- originating from / directed at
- ancestor / friend / other classes

The number of Class- Attribute interaction from class c to class d is given by

$$CA(c,d) = |\{a \mid a \in A_i(c') \wedge T(a) = d\}| \quad \textbf{(6)}$$

The number of Class- Method interaction from class c to class d is given by

$$CM(c,d) = \sum_{m \in M_{NEW}(c)} |\{a \mid a \in Par(m) \wedge T(a) = d\}| \quad \textbf{(7)}$$

The number of Method-Method interaction from class c to class d is given by

$$MM(c,d) = \sum_{m \in M_i(c)} \sum_{m' \in M_{NEW}(d) \cup M_{OVR}(d)} (NSI(m,m') + PP(m,m')) \quad \textbf{(8)}$$

Briand et al. extends this set of metrics to provide additional metrics. They define import coupling of a class as the interaction originated from the class and export coupling of a class as the interaction directed at the class. So based on the interaction type, interaction direction and the interacting objects, they define a set of metrics. Class-Attribute Interaction measures are expressed as follows.

$$IFCAIC(c) = \sum_{d \in Friends^{-1}(c)} CA(c,d) \quad \textbf{(9)}$$

counts all CA-interactions from class c to inverse friends of c,

$$ACAIC(c) = \sum_{d \in Ancestors(c)} CA(c,d) \quad \textbf{(10)}$$

counts all CA-interactions from class c to ancestors of c,

$$OCAIC(c) = \sum_{d \in Others(c) \cup Friends(c)} CA(c,d) \quad \textbf{(11)}$$

$$Others(c) = C - (Ancestors(c) \cup Friends(c) \cup Friends^{-1}(c) \cup Descendents(c))$$

counts all CA-interactions from class c to classes that are not ancestors or inverse friends of c,

$$FCAEC(c) = \sum_{d \in Friends(c)} CA(d,c) \quad \textbf{(12)}$$

counts all CA-interactions from friends of class c to class c,

$$DCAEC(c) = \sum_{d \in Descendants(c)} CA(d,c) \quad \textbf{(13)}$$

counts all CA-interactions from descendents of class c to class c,

$$OCAEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} CA(d,c) \quad \textbf{(14)}$$

counts all CA-interactions from classes that are not friends or descendents of class c to class c,

The definitions of class-method interaction measures namely, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC and the method-method interaction measures namely, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, and OMMEC follow the same template as class-attribute interaction measures except that CA interactions are replaced by CM interactions and MM interactions respectively. Thus the authors provide a set of 18 metrics for measuring different coupling in object oriented systems. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system. For further details on these coupling measures, please read [10, 11].

2.6.8. Coupling Dependency Metric (CDM)

The coupling dependency metric (CDM), given by Schach, [17] is the sum of three components: a measure of the extent to which a program relies on its declarations remaining unchanged (referential dependency); a measure of the extent to which a program relies on its internal organization remaining unchanged (structural dependency); and a measure of the vulnerability of data elements in one module to change by other modules (data integrity

dependency). This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system.

$$CDM = R_d + S_d + DI_d \quad \textbf{(15)}$$

Table 2 given at the end of this chapter gives the overview of all the coupling metrics discussed in this chapter.

All of these measures are either confined to the class level or are costly to implement. There is no cost-effective measure of coupling at the subsystem level in a large-scale system in the literature to our knowledge. So after analyzing different criteria we propose a coupling measure at the subsystem level which will help determine the reusability of a subsystem and estimate the maintenance cost of the system and complexity of the system and in effect, the cost of the whole system.

Table 2: Various Coupling Measures

#	Name	Definition	Source
1	CBO (Coupling Between Objects)	$CBO(c) = \mid \{d \in C - \{c\} \mid uses(c,d) \vee uses(d,c)\} \mid \forall c,d \in C$ $CBO'(c) = \mid \{d \in C - (\{c\} \cup Ancestors(C)) \mid uses(c,d) \vee uses(d,c)\} \mid \forall c,d \in C$	[12]
2	RFC (Response For Class)	$R_0(c) = M(c)$ $R_{i+1}(c) = \bigcup_{m \in R_i(c)} PIM(M)$ For α = 1, 2, 3… $RFC_\alpha(c) = \mid \bigcup_{i=0}^{\alpha} R_i(c) \mid$	[13]
3	NAS (Number of Associations)	Similar to CBO except that it counts repeated invocations as a single occurrence of coupling	[14]
4	MPC (Message Passing Coupling)	$MPC(c) = \sum_{m \in M_i(c)} \sum_{m' \in SIM(m) - M_i(c)} NSI(m,m')$	[15]

#	Name	Definition	Source
5	DAC (Data Abstraction Coupling)	$DAC(c) = \mid \{T(a) \mid a \in A_i(c) \wedge T(a) \in C\} \mid$	[15]
6	COF (Coupling Factor)	$COF(c) = \frac{\sum_{c \in C} \mid \{d \mid d \in C - (\{c\} \cup Ancestors(c)) \wedge uses(c,d)\} \mid}{\mid C \mid^2 - \mid C \mid - (2 \sum_{c \in C} \mid Descendents(c) \mid)}$	[16]
7	IFCAIC	$IFCAIC(c) = \sum_{d \in Friends^{-1}(c)} CA(c,d)$ where, $CA(c,d) = \mid \{a \mid a \in A_i(c') \wedge T(a) = d\} \mid$	[10, 11]
8	ACAIC	$ACAIC(c) = \sum_{d \in Ancestors(c)} CA(c,d)$ where,, $CA(c,d) = \mid \{a \mid a \in A_i(c') \wedge T(a) = d\} \mid$	[10, 11]
9	OCAIC	$OCAIC(c) = \sum_{d \in Others(c) \cup Friends(c)} CA(c,d)$ $Others(c) = C - (Ancestors(c) \cup Friends(c) \cup Friends^{-1}(c) \cup Descendents(c))$ where, $CA(c,d) = \mid \{a \mid a \in A_i(c') \wedge T(a) = d\} \mid$	[10, 11]
10	FCAEC	$FCAEC(c) = \sum_{d \in Friends(c)} CA(d,c)$ where, $CA(d,c) = \mid \{a \mid a \in A_i(d') \wedge T(a) = c\} \mid$	[10, 11]
11	DCAEC	$DCAEC(c) = \sum_{d \in Descendants(c)} CA(d,c)$ where, $CA(d,c) = \mid \{a \mid a \in A_i(d') \wedge T(a) = c\} \mid$	[10, 11]
12	OCAEC	$OCAEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} CA(d,c)$ where, $CA(d,c) = \mid \{a \mid a \in A_i(d') \wedge T(a) = c\} \mid$	[10, 11]
13	IFCMIC	$IFCMIC(c) = \sum_{d \in Friends^{-1}(c)} CM(c,d)$ where, $CM(c,d) = \sum_{m \in M_{NEW}(c)} \mid \{a \mid a \in Par(m) \wedge T(a) = d\} \mid$	[10, 11]
14	ACMIC	$ACMIC(c) = \sum_{d \in Ancestors(c)} CM(c,d)$ where, $CM(c,d) = \sum_{m \in M_{NEW}(c)} \mid \{a \mid a \in Par(m) \wedge T(a) = d\} \mid$	[10, 11]
15	OCMIC	$OCMIC(c) = \sum_{d \in Others(c) \cup Friends(c)} CM(c,d)$ $Others(c) = C - (Ancestors(c) \cup Friends(c) \cup Friends^{-1}(c) \cup Descendents(c))$ where, $CM(c,d) = \sum_{m \in M_{NEW}(c)} \mid \{a \mid a \in Par(m) \wedge T(a) = d\} \mid$	[10, 11]
16	FCMEC	$FCMEC(c) = \sum_{d \in Friends(c)} CM(d,c)$ where, $CM(d,c) = \sum_{m \in M_{NEW}(d)} \mid \{a \mid a \in Par(m) \wedge T(a) = c\} \mid$	[10, 11]
17	DCMEC	$DMMEC(c) = \sum_{d \in Descendants(c)} CM(d,c)$	[10, 11]

#	Name	Definition	Source
		where, $CM(d,c) = \sum_{m \in M_{NEW}(d)} \lvert \{a \mid a \in Par(m) \wedge T(a) = c\} \rvert$	
18	OCMEC	$OCMEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} CM(d,c)$ where, $CM(d,c) = \sum_{m \in M_{NEW}(d)} \lvert \{a \mid a \in Par(m) \wedge T(a) = c\} \rvert$	[10, 11]
19	IFMMIC	$IFMMIC(c) = \sum_{d \in Friends^{-1}(c)} MM(c,d)$ where, $MM(c,d) = \sum_{m \in M_i(c)} \sum_{m' \in M_{NEW}(d) \cup M_{OVR}(d)} (NSI(m,m') + PP(m,m'))$	[10, 11]
20	AMMIC	$AMMIC(c) = \sum_{d \in Ancestors(c)} MM(c,d)$ where, $MM(c,d) = \sum_{m \in M_i(c)} \sum_{m' \in M_{NEW}(d) \cup M_{OVR}(d)} (NSI(m,m') + PP(m,m'))$	[10, 11]
21	OMMIC	$OMMIC(c) = \sum_{d \in Others(c) \cup Friends(c)} MM(c,d)$ $Others(c) = C - (Ancestors(c) \cup Friends(c) \cup Friends^{-1}(c) \cup Descendents(c))$ where, $MM(c,d) = \sum_{m \in M_i(c)} \sum_{m' \in M_{NEW}(d) \cup M_{OVR}(d)} (NSI(m,m') + PP(m,m'))$	[10, 11]
22	FMMEC	$FMMEC(c) = \sum_{d \in Friends(c)} MM(d,c)$ where, $MM(d,c) = \sum_{m \in M_i(d)} \sum_{m' \in M_{NEW}(c) \cup M_{OVR}(c)} (NSI(m,m') + PP(m,m'))$	[10, 11]
23	DMMEC	$DMMEC(c) = \sum_{d \in Descendants(c)} MM(d,c)$ where, $MM(d,c) = \sum_{m \in M_i(d)} \sum_{m' \in M_{NEW}(c) \cup M_{OVR}(c)} (NSI(m,m') + PP(m,m'))$	[10, 11]
24	OMMEC	$OMMEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} MM(d,c)$ where, $MM(d,c) = \sum_{m \in M_i(d)} \sum_{m' \in M_{NEW}(c) \cup M_{OVR}(c)} (NSI(m,m') + PP(m,m'))$	[10, 11]
25	CDM	$CDM = R_d + S_d + DI_d$	[17]

CHAPTER THREE: A METRIC FOR COUPLING AT SUBSYSTEM LEVEL

In this chapter, a new metric for measuring coupling at the sub-system level in object- oriented systems is proposed. The metric is defined on the basis of different issues identified by comparing many of the relevant coupling measures as in chapter two. The main objective of this new measure is to estimate the complexity, maintenance cost and reusability of subsystems by measuring the coupling at the sub-system level. The intention of this measure is

- To measure the coupling count of a subsystem with other subsystems in a large-scale system.
- To measure the complexity of the system.
- To ease the maintenance cost estimation.
- To ease the estimation of the subsystem reusability.
- For easy re-factoring of the system design.

3.1. Criteria Considered

When considering a large scale software system, as discussed in chapter two, we have many numbers of classes and objects and thus measuring the coupling between those classes will be hard and need more effort and cost. Hence it is better to group closely related classes and abstract them as a subsystem and measure the coupling at the subsystem level. We have a lot of coupling metrics to measure the coupling between two classes or objects as discussed in chapter two, but we could not find a coupling metric to measure the dependency between the subsystems

of a large scale system. Since the criteria that need to be considered for designing a coupling metric is different based on the level of measurement and the purpose of the metric, we need a coupling metric for the subsystem level coupling measurement. Hence our main goal is to define a metric to measure the coupling at the subsystem level. We carefully designed this metric based on the following criteria derived from [11]

3.1.1. Measurement Goal

Maintenance cost which is around 70% of the total cost of the software, as discussed in chapter 1, is the higher cost in a software lifecycle. Hence goal of almost every system is to minimize the maintenance cost. Our measurement goal is to analyze the source code of different subsystems of a large-scale system to measure coupling and complexity to predict maintenance effort and reusability of a subsystem. Maintenance effort may be defined as the number of man-hours needed for fixing the faults in the system or to implement the changes to the system due to requirement changes. Higher the dependencies between the subsystems, we need more resources and efforts to maintain the system. Hence the maintenance cost will be high. A coupling metric will thus help in predicting the effort and resources needed to maintain the system. Reusability can be defined as the degree to which a subsystem or module can be used in more than one software system without needing much changes (almost no changes). If a subsystem is tightly coupled to another system, then it means that the subsystem is highly dependent on another subsystem and reusing this subsystem might need more changes or the other subsystem should also be used along with this subsystem. Designing and developing a subsystem from scratch might be better when compared to the efforts and cost associated with reusing a highly coupled subsystem. A coupling metric that can give the dependencies between

two subsystems can be a good metric in predicting the reusability of the subsystems rather the metric will be helpful in making a decision of reusing a subsystem against developing from scratch. This metric is defined to achieve these goals with less efforts and cost.

3.1.2. Type of Connection

It is important to choose the type of connections that we consider as coupling interaction, in defining the coupling metric, since type of the connection implies the mechanism that constitutes the coupling. For e.g. method level, which is the invocation of a method of a subsystem from another subsystem, or subsystem level, which is the high level relationship between subsystems. Different types of coupling connections have different strength. For e.g. invoking a method of another subsystem is stronger than having a reference to another subsystem without invoking any method.

In our metric, we consider method invocations and attribute references as the connection type between two subsystems. The reason behind this selection is that at the subsystem level, method invocations and attribute references are the most relevant type of connection between two subsystems. There will be almost no higher level relationships between subsystems like "consists-of" which is more common among class level coupling.

3.1.3. Locus of Impact

It has to be decided whether to count the import coupling or export coupling or both when defining a metric. High import coupling means that the subsystem depends strongly on other subsystem and its methods. Hence import coupling is more relevant in conjunction with understandability, error-proneness, maintainability and reusability, since to understand a

subsystem, we must know about all the services that subsystem uses and if our understanding of a service is wrong it might lead to more faults which will result in increased maintenance cost. Similarly if a subsystem depends on a large number of subsystems, it is hard to reuse since the other subsystems will have to be made available in the reusing system. Whereas high export coupling of a subsystem means that the subsystem is being used by a lot of other subsystems. Hence export coupling count will be more useful when our focus is mainly on the testability.

Since our measurement goal is maintainability and reusability, we focus only on counting the import coupling. If a method in a subsystem invokes many other methods of different subsystems, it is more likely to be affected by the changes in the other subsystems i.e., highly import coupled subsystems are hard to maintain and reuse. Moreover, if we consider both import and export coupling between two subsystems, it counts the same connection twice thus duplicating the coupling count and hence we focus only on import coupling.

3.1.4. Granularity

Granularity of the measure can be defined as the level of detail at which the coupling count is measured. It is determined by the domain of measure and the method of counting the coupling connections.

3.1.4.1. Domain of Measure

As discussed earlier in this work, we consider measuring the coupling at the subsystem level to help predicting the maintainability of each and every subsystem as well as the system as a whole and the reusability of each and every subsystem. Here we take into account all the connections starting from a subsystem and ending in another subsystem.

3.1.4.2. Counting Connections

Another important thing is to decide how to count the connections between subsystems. Some of the options for counting connections for a subsystem are

- Count individual connections to each subsystem
- Count individual connections from the each class/method of a subsystem to every other subsystem
- Count the number of other subsystems to which there is at least one connection from the subsystem

We consider counting the distinct individual connections from a subsystem to every other subsystem, i.e., if a method of a subsystem is invoked or an attribute is referenced more often from another subsystem; the same effort is needed to modify the invoking method/class when the invoked method or referenced attribute changes have no effect on the maintenance cost. Since we count only the method invocations or attribute references as type of connection, we count the number of methods of a subsystem invoked or number of attributes referenced from another subsystem. For e.g., if a subsystem invokes 4 methods of another subsystem and if each method is invoked 2 times, then the coupling count will still be 4.

3.1.5. Direct or Indirect Coupling

Like most of the other coupling measures, we consider only direct coupling. Direct coupling describes a relation on a set of elements like relation "invokes" on the set of all methods of other subsystems. Indirect coupling is a transitive closure of that relation i.e., if a subsystem invokes a method of another subsystem which in turn invokes another method of another subsystem and so on. If we consider indirect coupling, it will duplicate the coupling

count and hence it is not relevant in our case. Hence we consider only the direct invocation of methods of another subsystem as a coupling count.

3.1.6. Inheritance and Polymorphism

There is no inheritance relationship at the subsystem level. Inheritance relationships within the subsystem do not count for coupling at the subsystem level. Hence we do not consider the inheritance relation at all in defining our metric.

Any distinct method that is invoked by a subsystem might give rise to a modification in the invoking subsystem. Therefore, we include all methods that can be possibly invoked through polymorphism i.e., overloaded method invocations are considered as a single invocation.

3.2. Coupling at Subsystem Level (CSL)

Based on the criteria discussed in the previous sections, we design a coupling metrics to measure the dependency subsystems at the subsystem level. We call it "Coupling at Subsystem Level" (CSL). CSL of a system can be defined as the sum of all subsystem coupling (SC) of all the subsystems present in the system.

SC of a subsystem can be defined as the number of invocations of methods in other subsystems from the subsystem.

Let us assume that we have a set S of all the subsystems s_i. M(s) is a set of methods in subsystem s.

$$S = \{s_1, s_2, s_3, ...\}$$
$$M(s) = \{m1, m2, m3 ...\}$$
$$m1, m2, m3 .... \in s$$

$$CSL = \sum_{\forall s \in S} SC(s) \quad \textbf{(16)}$$

Where,

$$SC(s) = \sum_{\forall m \in M(s)} \sum_{\forall m' \in R_s} NPI(m, m') \quad \textbf{(17)}$$

Where m' belongs to a set of all the methods in all other subsystems invoked (accounts for polymorphism) by the subsystem s, which can be formally defined as

$$R_s = \sum_{\forall s_i \in S - s} PI(s_i)$$

Here, NPI (m, m') is the number of method invocations or attribute references from method m to method or attribute m' and

PI (s_i) is the non-invoked polymorphic method or attribute in subsystem s_i from a method in subsystems.

Here, SC is the coupling measure of a subsystem with all other subsystems in a large scale system. If the value of SC(s) is high, that means that the subsystem s is highly coupled to the system and it is hard to reuse or maintain the subsystem s thus increasing the complexity. CSL gives the coupling count of all the subsystems in the whole system. If the value of CSL is high, that means the system is highly nested inside. But SC gives more information about the reusability of a subsystem since it gives a measure of how a subsystem is related with other subsystems.

Sometimes we might need to know the coupling measure between two subsystems, like while designing the subsystems, if we know the coupling between two subsystems, combining two tightly coupled subsystems into a single subsystem might reduce the actual coupling measure of the system if those subsystems are not highly coupled to the other subsystems. Thus in order to measure the coupling between two subsystems, we define another measure called Coupling Between Subsystems which is similar to SC but takes into account only coupling with one subsystem instead of considering coupling with all the other subsystems.

CBS can be defined as the coupling count of a subsystem with another subsystem. It can be formally defined as

$$CBS(s,s') = \sum_{\forall m \in M(s)} \sum_{\forall m' \in PI(s')} NPI(m,m') \quad \textbf{(18)}$$

Where S={s, s'…}. CBS measure can never be negative or infinite. When there is no coupling between two subsystems, then CBS (s, s') = 0, if there is no coupling between s and s'. Similarly, a subsystem can never be coupled to itself. Hence,
CBS (s, s) = 0. From this definition the subsystem coupling SC defined in (17) can also be defined as

$$SC(s) = \sum_{\forall s_i \in S-s} CBS(s,s_i) \quad \textbf{(19)}$$

3.3. Coupling Complexity Measure of Large Scale System (CCMS)

Complexity measure of a large scale system (CCMS) is the complexity of the system in terms of coupling. Presence of a subsystem to a large scale system adds to the complexity as it needs to be maintained. The coupling of the system at the subsystem level also adds to the complexity. Thus Complexity measure of a large scale system can be defined as the sum of the number of subsystems and the coupling at the subsystem level. CCMS can be formally defined as

$$CCMS = Ns + CSL \quad \textbf{(20)}$$

Where Ns is the number of subsystems of a system S

CHAPTER FOUR: ANALYSIS AND FINDINGS

4.1. Properties of Metrics

Several researchers have recommended properties that software metrics should possess to increase their usefulness. For example, Basili and Reiter[18] suggest that metrics should be sensitive to externally observable differences in the development environment, and must also correspond to intuitive notions about the characteristic differences between the software artifacts being measured. Weyuker [19] has developed a formal list of properties for software metrics and has evaluated a number of existing software metrics using these properties. These properties include notions of monotonicity, interaction, non-coarseness, non-uniqueness and permutation. He developed nine properties. They are

a. *Non-coarseness:* Given an entity, A, and a metric m another entity B can always be found such that: m (A) ≠ m (B). This implies that not every entity can have the same value for a metric; otherwise it has lost its value as a measurement.
b. *Granularity:* The number of cases having the same metric value is always finite
c. *Non-uniqueness (Notion of Equivalence):* There can exist distinct entities A and B and a metric m such that m (A) = m (B). This implies that two entities can have the same metric value, i.e., the two entities are equally complex.
d. *Design Details are Important:* Given a metric m and two entities, A and B, which has the same functionality, does not imply that m (A) = m (B). The intuition

behind Property 3 is that even though two entities perform the same function, the details of the design matter in determining the metric for the entity.

e. *Monotonicity:* For all entities A and B, given a metric m, the following must hold: m (A) ≤ m (A+B) and m (B) ≤ m (A+B) where A+B implies combination of A and B. This implies that the metric for the combination of two entities can never be less than the metric for either of the entity.

f. *Non-Equivalence of Interaction:* For all entities A, B and C, give a metric m, m (A) = m (B) does not imply that m (A+C) = m (B+C). This means that interaction between A and C can be different than interaction between B and C resulting in different complexity values for A+C and B+C.

g. *Permutation:* If an entity B is formed by changing the order of statements of entity A, then given a metric m, m (A) ≠ m (B). The intent is to ensure that metric values change due to permutation of program statements.

h. *Renaming:* When the name of the measured entity changes, the metric should remain unchanged.

i. *Interaction Increases Complexity:* There exists a metric m and entities A and B such that m (A) + m (B) < m (A+B). The principle behind this property is that when two entities are combined, the interaction between entities can increase the complexity metric value.

4.2. Analytical Evaluation of SC and CBS

In this section, our two new metrics SC and CBS are analyzed and evaluated against the properties of metrics defined by Weyuker as discussed in the previous section. This analytical evaluation explains how our metric satisfies or not satisfies those properties.

a. *Non-coarseness:* Not all subsystems can have the same SC or CBS since the number of subsystems that it interacts with varies for different subsystems. Hence this property is satisfied.

b. *Granularity:* Since the number of subsystems of any large scale system is always finite, the number of subsystems having the same SC or CBS is also finite. Hence this property is satisfied.

c. *Non-uniqueness (Notion of Equivalence):* A subsystem can have the same number of interactions with the rest of the subsystems as another subsystem thus having the equal value of SC. CBS (s1, s2) can be equal to CBS (s3, s2), since both s1 and s3 can have the same level of interactions with s2. Therefore this property is satisfied.

d. *Design Details are Important:* Inter-subsystem coupling occurs when methods of one subsystem use methods of another subsystem, i.e., coupling depends on the manner in which methods are designed and not on the functionality provided by the subsystem. Therefore this property is satisfied.

e. *Monotonicity:* Let A and B be two subsystems with SC (A) = p and SC (B) = q. If A, and B are combined, the resulting subsystem will have p + q – r couples, where r is the number of couples reduced due to the combination i.e., SC (A+B) =

SC (A) + SC (B) – (CBS (A, B) + CBS (B, A)). If A and B are highly coupled i.e., r is very high, SC (A+B) may be less than SC (A) or SC (B). Hence this property is not satisfied for CSL. Where as for any three subsystems, A, B and C, if A and B are combined, then CBS (A+B, C) ≥ CBS (A, C) and CBS (A+B, C) ≥ CBS (B, C). Thus this property holds good for CBS.

f. *Non-Equivalence of Interaction:* For all subsystems A, B and C, let SC (A) = SC (B). SC (A + C) = SC (A) + SC (C) – CBS (A, C) and SC (B+C) = SC (B) + SC (C) – CBS (B, C). Since CBS (A, C) and CBS (B, C) may not be equal, SC (A+C) is not necessarily equal to SC (B+C). This means that interaction between A and C can be different than interaction between B and C resulting in different complexity values for A+C and B+C. Thus this property is satisfied.

g. *Permutation:* This property holds good only for structured programming, since order of methods need not be the same as the order of execution in object-oriented systems.

h. *Renaming:* SC and CBS do not depend on the name of the subsystem and depends only on the details of the implementation. Hence renaming a subsystem does not affect the CSL and CBS. Hence this property holds good.

i. *Interaction Increases Complexity:* Let A, B and C be any three subsystems with SC (A) = p and SC (B) = q. If A, and B are combined, the resulting subsystem will have p + q – r couples, where r is the number of couples reduced due to the combination. That is SC (A+B) = SC (A) + SC (B) – (CBS (A, B) + CBS (B, A)). Since CBS (A, B) and CBS (B, A) are non-negative, SC (A) + SC (B) > SC

(A+B). But, CBS (A, C) + CBS (B, C) ≤ CBS (AB, C) is possible for some cases.

Hence this property is not satisfied for SC and satisfied only for CBS

Our metrics satisfies almost all of the Weyuker's properties. Though it does not satisfy two of those properties, these two properties clearly shows that not being satisfied actually decreases the complexity. Hence our set of metrics is proven to be useful for the given set of goals.

4.3. Coupling as a Predictor of Maintenance Effort

Coupling is a well proven predictor of maintenance effort. Many researchers have done validation on coupling as a good predictor. According to A.B.Binkley and S.R.Schach [17], a significant impediment to maintenance is the level of interaction (or coupling) between modules. This was supported by a series of case studies which show that modules that display a low level of coupling undergo less maintenance and have fewer maintenance faults. That is, coupling metrics are excellent predictors of maintenance measures. Coupling metrics also are also excellent predictors of run-time failures. Since predicting the run-time failures will also aid in predicting the maintenance effort. Consider a subsystem S. Let C denote the value of a coupling metric, for instance SC, applied to S. This coupling-based metric incorporates the coupling between S and the rest of the system. Then, if a change is made outside S, P is a measure of the probability that the change outside S will require a corresponding change within S. In most of the cases, the need for this change within S will be revealed by the compiler or linker while compiling. However, other types of changes may be overlooked, especially when a large-scale system is developed. Unless the required change is made, we might eventually get a run-time

failure. This is why coupling based metrics in general are good predictors of run-time failures. To fix a run-time failure requires corrective maintenance. Thus, a metric that can predict where a run-time failure is likely to occur will also be a good predictor of subsystem-level corrective maintenance measures like the number of faults or time to repair faults. However, coupling also is a good predictor of all other forms of maintenance, including perfective and adaptive maintenance. The reason is that during maintenance the code is changed, and coupling is a measure of the likelihood that a change outside S will necessitate a change within S, irrespective of the reason for that change. In short, the value of a coupling metric is a measure of the probability that S must be changed as a consequence of any change to the rest of the system.

4.4. Coupling as a Predictor of Reusability

Coupling is also very much helpful in predicting the reusability of a subsystem. Many researchers have done validation on coupling as a good predictor of reusability. Consider a subsystem S. Let C denote the value of a coupling metric, say CBS, applied to S and another subsystem S'. This coupling-based metric incorporates the coupling between S and S'. If code inside S calls a method M inside S' then it means S depends on S' for performing some operation. The coupling-metric will measure such kind of dependencies. If S needs to be reused in another system that does not contain S', then either S' should be added along with S or S should be modified in such a way that S does not depend on S'. Since the coupling metrics measures the dependency of a subsystem with the rest of the subsystems, coupling metrics will help in knowing the number of subsystems (since S' might depend on another subsystem s'') that must be added along with S. Also the coupling metric reveals the number of changes needed

inside S for S to be reused in another system. Unless the required change is made or the required subsystem is added along, we might eventually get a run-time failure in the new system. The coupling is a measure of the likelihood that a reuse of S will necessitate the addition of S' along with S, or that a reuse of S will necessitate the change inside S irrespective of the place of reuse. In short, the value of a coupling metric is a measure of the probability that S must be changed or S' must be added to S, as a consequence of the reuse of S in another system.

4.5. Empirical Validation using Regression

To do an empirical validation, a univariate logistic regression [10, 20, 21] can be performed, for each coupling measure (independent variable), against the dependent variable (reusability or maintainability) to determine if the measure is statistically related, in the expected direction, to reusability and maintenance cost. Logistic regression is a standard technique based on maximum likelihood estimation. A univariate logistic regression model is based on the following equation:

$$p(X) = \frac{e^{c_0 + c_1 X}}{1 + e^{c_0 + c_1 X}} \quad \textbf{(21)}$$

where p is the probability that a subsystem can be reused or the probability that the maintenance cost is increased, and the X is the coupling measure included as independent variable in the model (covariates of the logistic regression equation). The dependent variable, p, is a conditional probability: the probability that a subsystem can be reused or the probability that the maintenance cost is increased. Unlike with other regression techniques (e.g., linear regression,

Poisson regression), the dependent variable is not measured directly. To further illustrate this concept, we consider a simple example. We could have a prediction model for the reusability of a subsystem with a coupling with other subsystems CSL. A result such as p (CSL=3) = 0.3 could be interpreted as "there is a 30% probability that a subsystem with CSL=3 can be reused". The curve between p and X takes a flexible S shape which ranges between two extreme cases:

1. When a variable is not significant, then the curve approximates a horizontal line, i.e., p does not depend on X.
2. When a variable entirely differentiates fault-prone software parts, then the curve approximates a step function.

Such an S shape is perfectly suitable as long as the relationship between X and p is monotonic. The coefficients C_0 and C_1 are estimated through the maximization of a likelihood function, i.e., as the product of the probabilities of the single observations, which are functions of the covariates (whose values are known in the observations) and the coefficients (which are the unknowns). For mathematical convenience, $l = \ln [L]$, the log likelihood, is usually the function to be maximized. In our case, observation could be the increase in the maintenance cost in-terms of person-hours or the ease of reusability in terms of the number of changes made to the reusing subsystem. In contrast, we could also perform the validation using the linear regression model, if we want measure the maintenance cost or reusability (number of changes needed to reuse) directly instead of measuring the probability. We present an experimental analysis of these metrics using a large scale system in the next chapter. We have not performed the regression analysis of our metrics yet since we do not have enough data to perform a regression analysis.

CHAPTER FIVE: EXPERIMENTAL ANALYSIS AND RESULTS

5.1. Coupling Measure and Design Complexity

In the previous chapter, we discussed how the coupling measure helps in predicting the reusability and maintainability in a large scale software system. In this chapter, we will discuss how the coupling measure determines the complexity of the large scale system and how this is helpful in increasing the reusability and decreasing maintainability by applying them to a large scale system.

Reusability and maintainability go hand in hand to give out a best performance. The maintenance effort needed for a large scale software system will be reduced if we use a solid well maintained subsystem. Hence using reusable systems is one of the significant ways of reducing maintenance effort. Complexity of a system does a major role in designing the subsystem with reusing capability. Complexity of a system directly shows the effort needed to maintain the system i.e., higher the complexity of the system, higher is the maintenance cost. Coupling measure helps in designing the system in such a way that the complexity of the system is minimal, thus helping to reduce the maintenance cost of the software system. We analyzed the design of a large scale system and measured the coupling measure of its different subsystems and thus measured the complexity of the system. Based on the coupling measures, we redesigned the system to achieve minimal complexity thus decreasing its maintenance effort needed.

5.2. Experimental Large Scale Software System

We used a large scale system built in a .NET environment for our experimental study. SCINET is an enterprise level application consisting of more than12 subsystems. The subsystems, we considered for the experiment are

1. Database Manager (DBM)
2. User Management System (UMS)
3. Category Code Management System (CCM)
4. Help and Feedback System (HFS)
5. Code Enumerations Builder (CEB)
6. Work Flow Management System (WFM)
7. Document Management System (DM)
8. Letter Management System (LM)
9. Business Rules Manager (BRM)
10. Agenda
11. Building
12. Permit Code Handler

5.2.1. Design Complexity of SCINET System

The initial design of the SCINET application is shown in Figure 3. As we see in the figure, all the subsystems are coupled to the Database Manager through which all the subsystems access the database. This is a perfect example of a hyper-spaghetti system.

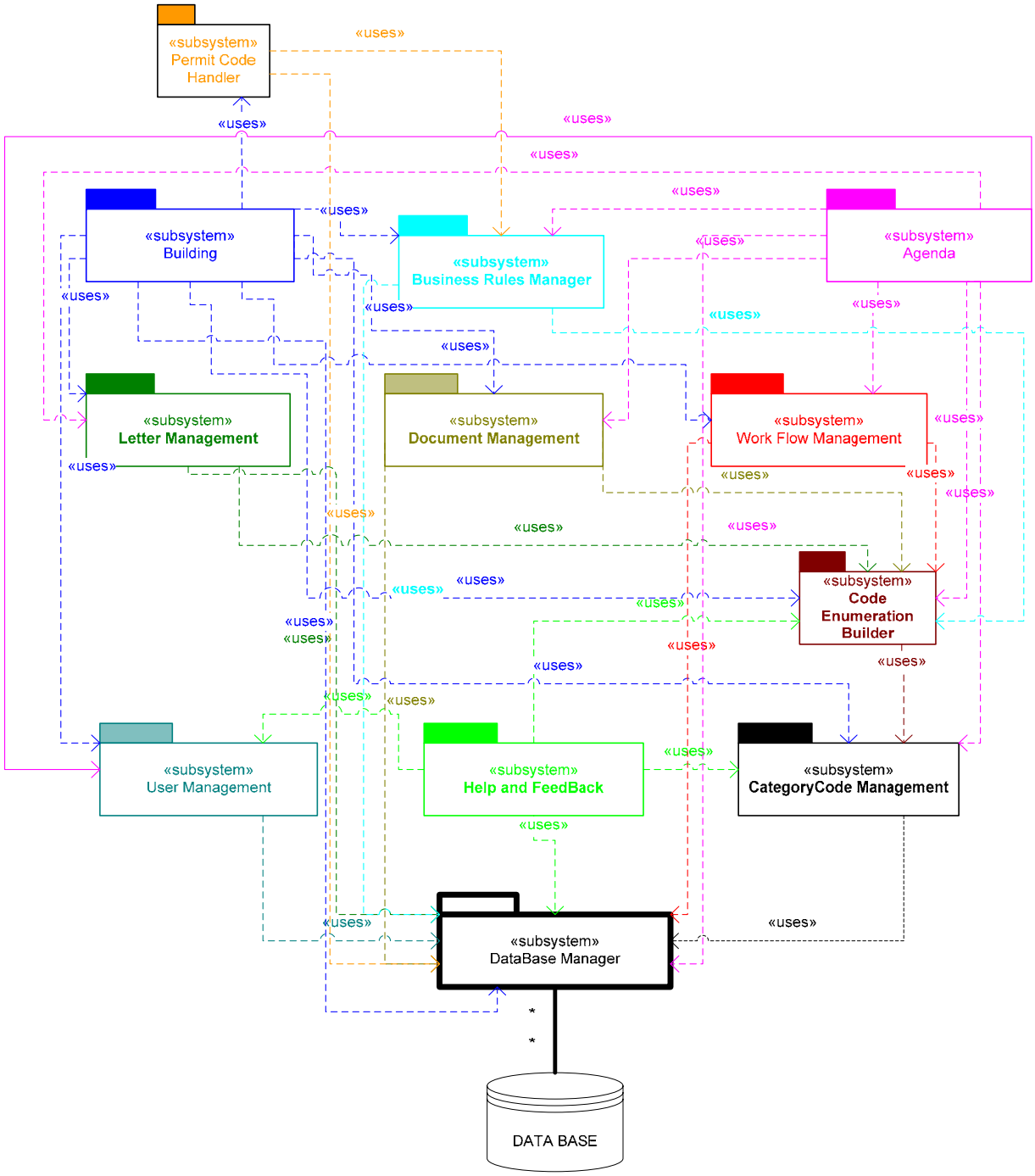

Figure 3: Initial SCINET System Dependencies

In the Figure 3, lines from one subsystem to another subsystem that indicates “uses”, implies that the subsystem uses the other subsystem (one with the arrow head pointing towards it) i.e., the subsystem is coupled to the other subsystem. Here we consider only the import coupling with respect to a subsystem, since the export coupling is counted with respect to the other subsystem.

We viewed the system as a hierarchy as subsystems to reduce the cost involved in measuring the coupling and complexity. As discussed before, two subsystems at the same hierarchical level is not coupled and hence the CBS of the subsystems at the same level is always 0 and need not be measured explicitly. Similarly, CBS between the subsystems in the lower hierarchical level and the subsystems in the upper hierarchical levels is always zero and need not be explicitly measured. The hierarchical arrangement of our system is shown in figure 4. As we see in figure 4, subsystems at the same level are not coupled and the subsystems are not coupled from lower level to upper level.

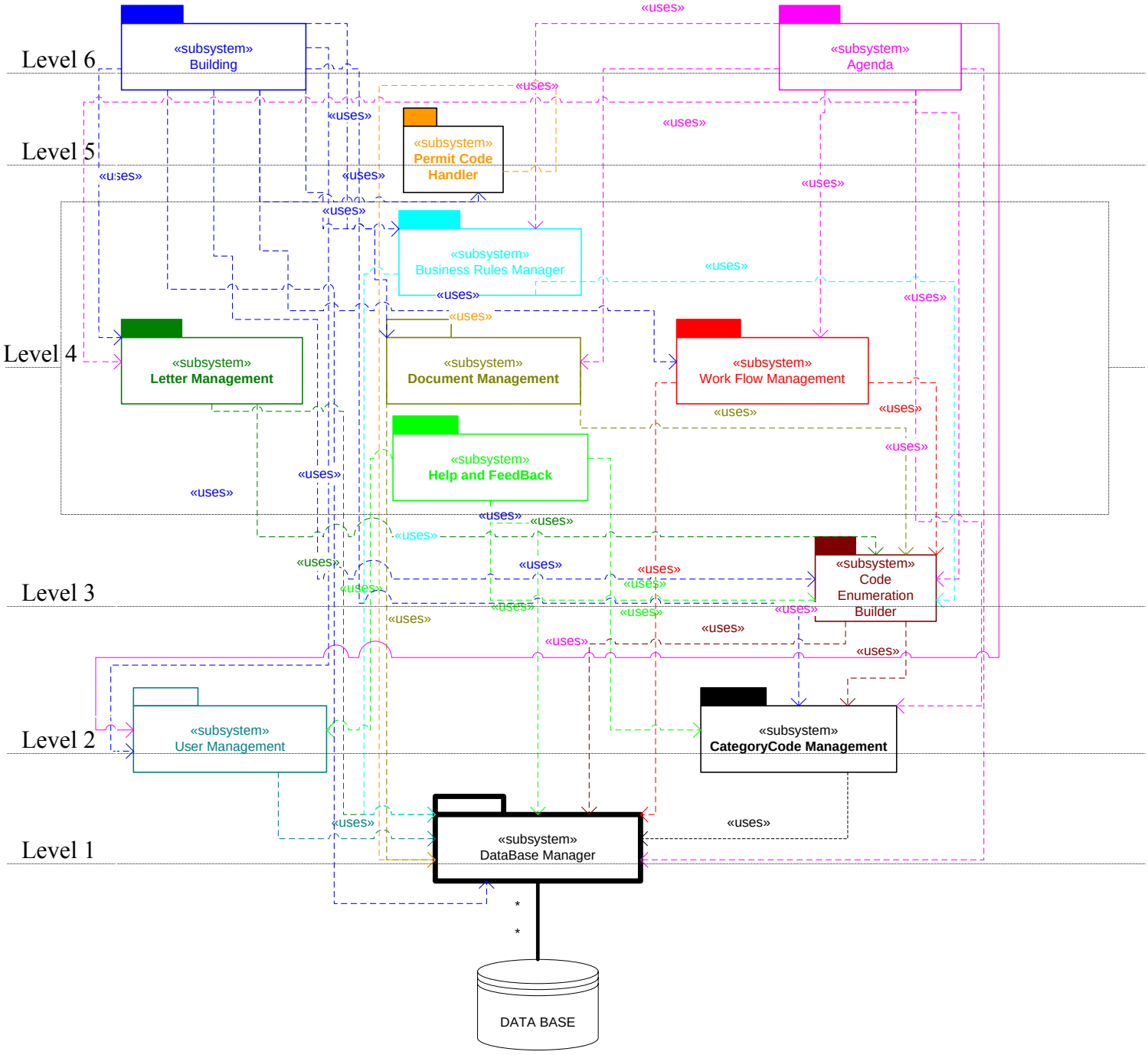

Figure 4: SCINET - Hierarchical View

We applied the coupling metrics we defined in chapter 3 to this large scale system to determine the complexity of the system. The CBS of a subsystem with itself is 0. Thus

CBS (DBM, DBM) = 0

CBS (UMS, UMS) = 0

CBS (HFS, HFS) = 0

CBS (CCM, CCM) = 0

CBS (CEB, CEB) = 0

CBS (WFM, WFM) = 0

CBS (LM, LM) = 0

CBS (DM, DM) = 0

CBS (BRM, BRM) = 0

CBS (Agenda, Agenda) = 0

CBS (Building, Building) = 0

CBS (Permit Code Handler, Permit Code Handler) = 0

The UMS is coupled to DBM with a weighted measure of 4 i.e., UMS invokes 4 polymorphic methods of DBM. Hence CBS (UMS, DBM) = 4. Since UMS is not coupled to any other subsystems, CBS of UMS with every other system except DBM is 0. Thus UMS adds a total weight of 4 to the system complexity.

Similarly, HFS is coupled to DBM, UMS, CCM and CEB with weighted measure of 4, 3, 1, and 1 respectively. Thus CBS (HFS, DBM) = 4, CBS (HFS, UMS) = 3, CBS (HFS, CCM) = 1 and CBS (HFS, CEB) = 1. Since HFS is not dependent on the rest of the subsystems, CBS

of HFS with the rest of the subsystems is 0. Thus HFS adds a total weight of 4+3+1+1 = 9 to the system complexity.

Similarly CCM is coupled only to DBM with a weighted measure of 2 and hence CBS (CCM, DBM) = 2 and CBS of CCM with the rest of the subsystems is 0. Thus CCM adds a total weight of 2 to the system complexity.

CEB is dependent only on the CCM subsystem invoking 3 methods of CCM. Hence CBS (CEB, CCM) = 3 and CBS of CEB with the rest of the subsystems is 0. Thus CEB adds a total weight of 3 to the system complexity.

WFM is dependent only on DBM and CEB with a weight of 1 each. Hence CBS (WFM, DBM) = 1 and CBS (WFM, CEB) = 1 and CBS of WFM with rest of the subsystems is 0. Thus WFM adds a total weight of 1+1 = 2 to the system complexity.

DM is dependent only on DBM and CEB with a weight of 1 each. Hence CBS (DM, DBM) = 1 and CBS (DM, CEB) = 1 and CBS of DM with rest of the subsystems is 0. Thus DM adds a total weight of 1+1=2 to the system complexity.

LM is dependent only on DBM and CEB with a weight of 1 each. Hence CBS (LM, DBM) = 1 and CBS (LM, CEB) = 1 and CBS of LM with rest of the subsystems is 0. Thus LM adds a total weight of 1+1=2 to the system complexity.

BRM is dependent only on DBM and CEB with a weight of 1 each. Hence CBS (BRM, DBM) = 1 and CBS (BRM, CEB) = 1 and CBS of BRM with rest of the subsystems is 0. Thus BRM adds a total weight of 1+1=2 to the system complexity.

Agenda being an application specific subsystem is dependent on almost all the other subsystems. Agenda subsystem invokes 4 methods of DBM, 8 invocations of UMS, 4 methods

of CCM, 13 invocations of CEB, 2 methods of WFM, 2 methods of DM, 2 methods of LM, 1 method of BRM and is not coupled with Building, Permit Code Handler and HFS subsystems. Thus CBS (Agenda, DBM) = 4, CBS (Agenda, UMS) = 8, CBS (Agenda, CCM) = 4, CBS (Agenda, CEB) = 13, CBS (Agenda, WFM) = 2, CBS (Agenda, DM) = 2, CBS (Agenda, LM) = 2, CBS (Agenda, BRM) = 1 and CBS of Agenda with HFS, Building and Permit Code Handler subsystems is 0. Thus Agenda adds a total weight of 4+8+4+13+2+2+2+1 = 36 to the system complexity.

Building is also an application specific subsystem and hence is dependent on almost all the other subsystems. Building subsystem invokes 3 methods of DBM, 3 invocations of UMS, 2 methods of CCM, 18 invocations of CEB, 2 methods of WFM, 2 methods of DM, 2 methods of LM, 1 method of BRM, 30 invocations of Permit Code Handler and is not coupled with Agenda and HFS subsystems. Thus CBS (Building, DBM) = 3, CBS (Building, UMS) = 3, CBS (Building, CCM) = 2, CBS (Building, CEB) = 18, CBS (Agenda, WFM) = 2, CBS (Building, DM) = 2, CBS (Building, LM) = 2, CBS (Building, BRM) = 1, CBS (Building, Permit Code Handler) = 30 and CBS of Building with HFS, Agenda subsystems is 0. Thus Building adds a total weight of 3+3+2+18+2+2+2+1+30 = 63 to the system complexity.

Permit Code Handler is specific to building and hence is dependent only on DBM, BRM and CEB with a weight of 1 each. Hence CBS (Permit Code Handler, DBM) = 1 and CBS (Permit Code Handler, CEB) = 1, CBS (Permit Code Handler, BRM) = 1 and CBS of BRM with rest of the subsystems is 0. Thus Permit Code Handler adds a total weight of 1+1+1= 3 to the system complexity. These measures are shown in Table 3 and Table 4. Table 3 gives the Coupling Between Subsystems (CBS) measures of any two subsystems of the SCINET system

and Table 4 gives the Subsystem Coupling (SC) of each subsystem, which is the sum of the CBS measures of each subsystem with the other subsystems i.e., the total weight that each subsystem contributes to the complexity of the system. In Table 3, each cell represents the Coupling Between Subsystems measure (CBS) between the subsystems corresponding to its row and the column.

Table 3: Initial Coupling Between Subsystems (CBS) Measures of SCINET System

CBS	DBM	UMS	HFS	CCM	CEB	WFM	DM	LM	BRM	Agenda	Building	Permit Code Handler
DBM	0	0	0	0	0	0	0	0	0	0	0	0
UMS	4	0	0	0	0	0	0	0	0	0	0	0
HFS	4	3	0	1	1	0	0	0	0	0	0	0
CCM	2	0	0	0	0	0	0	0	0	0	0	0
CEB	0	0	0	3	0	0	0	0	0	0	0	0
WFM	1	0	0	0	1	0	0	0	0	0	0	0
DM	1	0	0	0	1	0	0	0	0	0	0	0
LM	1	0	0	0	1	0	0	0	0	0	0	0
BRM	1	0	0	0	1	0	0	0	0	0	0	0
Agenda	4	8	0	4	13	2	2	2	1	0	0	0
Building	3	3	0	2	18	2	2	2	1	0	0	30
Permit Code Handler	1	0	0	0	1	0	0	0	1	0	0	0

Table 3 shows that Agenda and Building subsystems which are the application specific subsystems, is strongly coupled to Code Enumerations Builder (CEB) than any other subsystem with a higher CBS measure. Similarly, Building is strongly coupled to Permit Code Handler System with the highest CBS measure.

Table 4 : Initial Subsystem Coupling (SC) Measures of SCINET System

Sub-System	SC
DBM	0
UMS	4
HFS	9
CCM	2
CEB	3
WFM	2
DM	2
LM	2
BRM	2
Agenda	36
Building	63
Permit Code Handler	3

Table 4 shows the SC measures of each subsystem. Building Subsystem having a stronger coupling with Permit Code Handler Subsystem contributes the most to the complexity of the system. The Coupling at the Subsystem Level (CSL) can be got by the summation of SC measures of the subsystems in the large scale system. Thus the CSL of SCINET is given by the summation of all the values in Table 4.

CSL (SCINET) = 0 + 4 + 9 + 2 + 3 + 2 + 2 + 2 + 2 + 36 + 63 + 3 = 128

The Actual Coupling Complexity Measure of the Large Scale System (CCMS) being the sum of the number of subsystems and the coupling at the subsystem level, CCMS of SCINET having 12 subsystems is given by

CCMS (SCINET) = 12 + 128 = 140

The complexity of SCINET system with the design shown in Figure 3 is 140. This complexity might be reduced by changing the design of the system. From Table 3, it is seen that the Code Enumerations Builder Subsystem contributes the most for the complexity of the

system since many of the subsystems are strongly coupled to the CEB Subsystem. Most of the subsystems that are coupled to CEB subsystem are also coupled to CCM subsystem and the CEB subsystem is coupled only to the CCM subsystem. From the reuse point of view also, since CEB cannot be reused without CCM, CEB can be embedded within CCM. Thus combining these two subsystems will help in reducing the coupling of the system at the subsystem level thus reducing the complexity of the system. In addition there is no point in considering combining the subsystems at the same level as they are not at all coupled.

5.2.2. Design Complexity of SCINET with CEB and CCM Merged

The SCINET is redesigned with the CCM subsystem and CEB subsystem merged into a single subsystem Category Code Management (CCM). The new design of the SCINET system is shown in Figure 4. In the bottom of the figure, it is shown how the CEB subsystem is embedded inside CCM subsystem thus reducing the number of subsystems of SCINET from 12 to 11. We measured the coupling measures of the subsystems after merging CEB subsystem and CCM subsystem. The results are shown in Table 5 and Table 6. Table 5 shows the Coupling Between Subsystems (CBS) measures of any two subsystems of the redesigned SCINET system and Table 4 gives the Subsystem Coupling (SC) of each subsystem of the redesigned SCINET system, which is the sum of the CBS measures of each subsystem with the other subsystems.

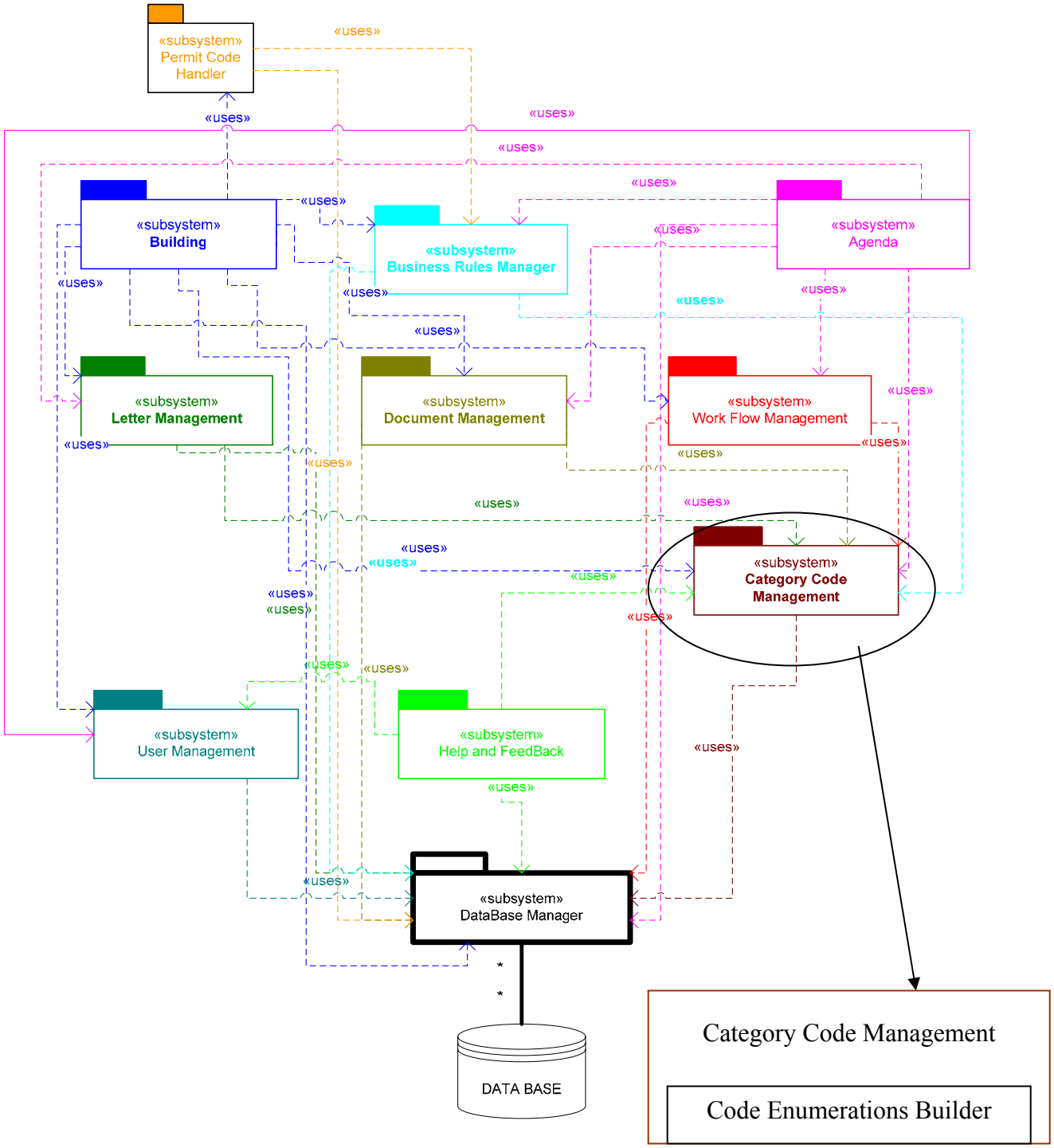

Figure 5: SCINET System Dependencies after Merging CCM and CEB

Table 5: CBS Measures after merging CCM and CEB

CBS	DBM	UMS	HFS	CCM	WFM	DM	LM	BRM	Agenda	Building	Permit Code Handler
DBM	0	0	0	0	0	0	0	0	0	0	0
UMS	4	0	0	0	0	0	0	0	0	0	0
HFS	4	3	0	2	0	0	0	0	0	0	0
CCM	2	0	0	0	0	0	0	0	0	0	0
WFM	1	0	0	1	0	0	0	0	0	0	0
DM	1	0	0	1	0	0	0	0	0	0	0
LM	1	0	0	1	0	0	0	0	0	0	0
BRM	1	0	0	1	0	0	0	0	0	0	0
Agenda	4	8	0	17	2	2	2	1	0	0	0
Building	3	3	0	20	2	2	2	1	0	0	30
Permit Code Handler	1	0	0	1	0	0	0	1	0	0	0

Table 6: SC Measures after merging CCM and CEB

Sub-System	**SC**
DBM	0
UMS	4
HFS	9
CCM	2
WFM	2
DM	2
LM	2
BRM	2
Agenda	36
Building	63
Permit Code Handler	3

The Coupling at the Subsystem Level (CSL) can be got by the summation of SC measures of the subsystems in the large scale system. Thus the CSL of SCINET is given by the summation of all the values in Table 4.

CSL (SCINET) = 0 + 4 + 9 + 2 + 2 + 2 + 2 + 2 + 36 + 63 + 3 = 125

Coupling Complexity Measure of the Large Scale System (CCMS) being the sum of the number of subsystems and the coupling at the subsystem level, CCMS of SCINET having 11 subsystems (reduced from 12 to 11) is given by

CCMS (SCINET) = 11 + 125 = 136

The complexity of SCINET system with the design shown in Figure 4 is 136. Since many of the subsystems depend on the CCM, merging of CCM and CEB did not reduce the complexity of the system to a great extent, except that the coupling between CEB and CCM is gone and reduction in the number of subsystems helps in reducing complexity of the system. Even though it did not reduce the complexity to a great extent, it is a significant reduction which will reduce the maintenance effort needed. More over, CCM system is still a black box which can be reused (along with Database Manager) in any other system thus reducing the cost of the other systems in which it is used.

Now when we look at Table 5, we see that Building subsystem is highly coupled to Permit Code Handler subsystem and it is the only subsystem that is coupled to the latter. We also see that, Permit Code Handler subsystem is coupled to three other subsystems namely DBM, BRM and CCM, to which Building subsystem is also coupled. From the reuse point of view, since both Building and Permit Code Handler subsystems are application specific, it is not necessary to give much importance to reuse capability to these subsystems. Hence if we combine these two subsystems to a single subsystem, the coupling measures and thus the complexity of the SCINET system will be greatly reduced.

5.2.3. Design Complexity of Redesigned SCINET

The SCINET is redesigned with the Building subsystem and Permit Code Handler subsystem merged into a single subsystem Building. The new design of the SCINET system is shown in Figure 5. In the top of the figure, it is shown how the Permit Code Handler subsystem is embedded inside Building subsystem thus reducing the number of subsystems of SCINET from 11 to 10.

We again measured the coupling measures of the subsystems after merging Permit Code Handler subsystem and Building subsystem. The results are shown in Table 7 and Table 8. Table 7 shows the Coupling Between Subsystems (CBS) measures of any two subsystems of the redesigned SCINET system and Table 8 gives the Subsystem Coupling (SC) of each subsystem of the redesigned SCINET system, which is the sum of the CBS measures of each subsystem with the other subsystems.

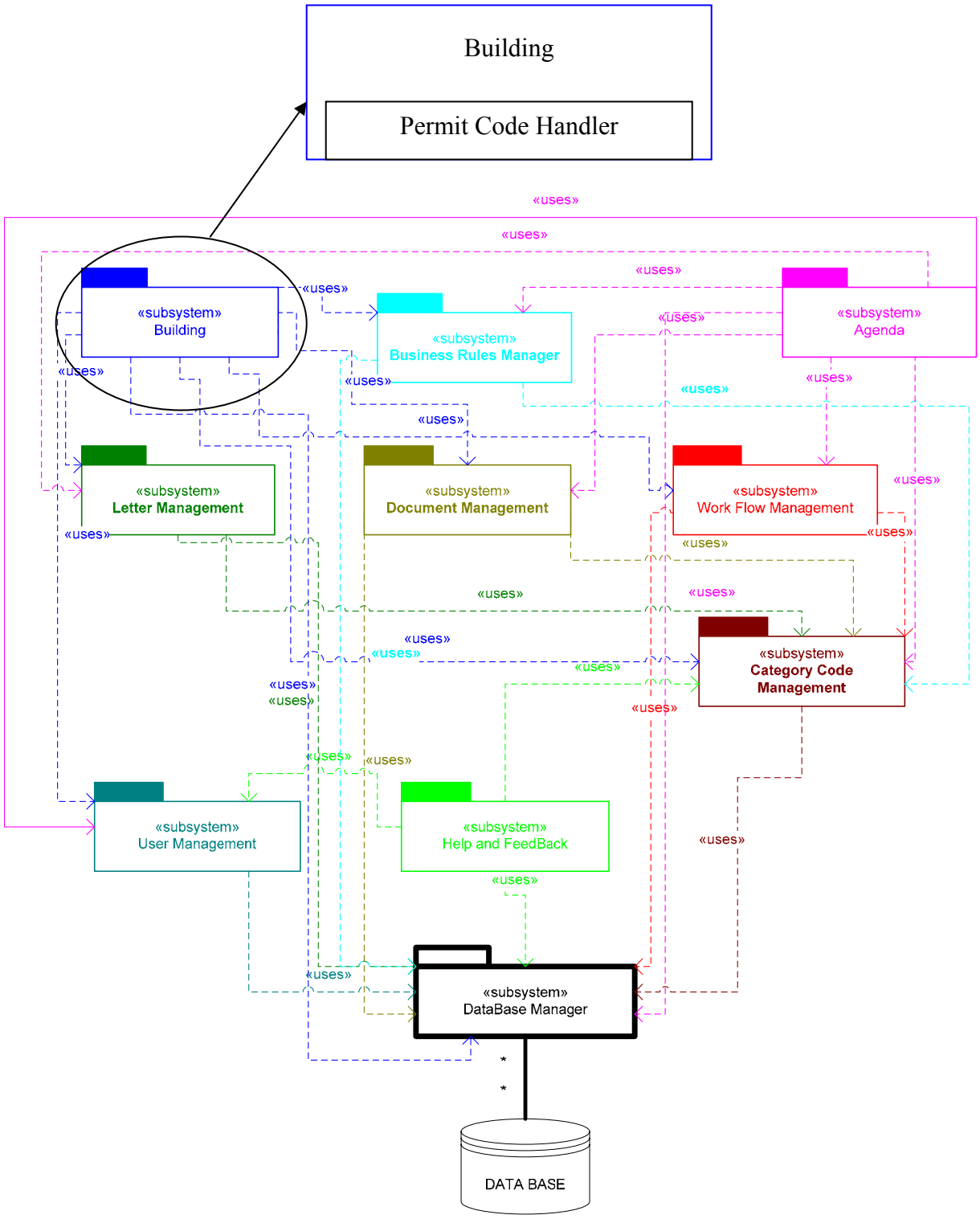

Figure 6: SCINET System Dependencies after merging Building and Permit Code Handler

Table 7: CBS Measures after merging Building and Permit Code Handler

CBS	DBM	UMS	HFS	CCM	WFM	DM	LM	BRM	Agenda	Building
DBM	0	0	0	0	0	0	0	0	0	0
UMS	4	0	0	0	0	0	0	0	0	0
HFS	4	3	0	2	0	0	0	0	0	0
CCM	2	0	0	0	0	0	0	0	0	0
WFM	1	0	0	1	0	0	0	0	0	0
DM	1	0	0	1	0	0	0	0	0	0
LM	1	0	0	1	0	0	0	0	0	0
BRM	1	0	0	1	0	0	0	0	0	0
Agenda	4	8	0	17	2	2	2	1	0	0
Building	3	3	0	21	2	2	2	1	0	0

Table 8: SC Measures after merging Building and Permit Code Handler

Sub-System	**SC**
DBM	0
UMS	4
HFS	9
CCM	2
WFM	2
DM	2
LM	2
BRM	2
Agenda	36
Building	34

The Coupling at the Subsystem Level (CSL) can be got by the summation of SC measures of the subsystems in the large scale system. Thus the CSL of SCINET is given by the summation of all the values in Table 4.

CSL (SCINET) = 0 + 4 + 9 + 2 + 2 + 2 + 2 + 2 + 36 + 34 = 93

Coupling Complexity Measure of the Large Scale System (CCMS) being the sum of the number of subsystems and the coupling at the subsystem level, CCMS of SCINET having 10

subsystems (reduced from 11 to 10 by merging Permit Code Handler Subsystem into Building Subsystem) is given by

CCMS (SCINET) = 10 + 93 = 103

The complexity of SCINET system with the design shown in Figure 5 is 103 which is very much lower than the previous two designs. This design with reduced number of subsystems requires a reduced maintenance effort.

If we take a look at the subsystems LM, DM, WFM and BRM, they all are coupled to CCM and DBM and only Agenda and Building are coupled to these subsystems. Thus merging all these subsystems might help in minimizing the maintenance cost of the system. But since these 4 subsystems do not talk to each other i.e., these subsystems belong to the same hierarchical level and hence are not coupled to each other and since these subsystems are generic, it is better to build them in a reusable way. Combining these systems will decrease the reusability of the subsystems there by increasing the development and maintenance cost of the other systems. So it is better to keep these subsystems separate. Table 9 shows the comparison of the CSL and CCMS of the three designs.

Table 9: Coupling and Complexity of SCINET

SCINET	Initial Design	CEB and CCM Merged	Building and Permit Code Handler Merged
Number of Sub-Systems	12	11	10
CSL	128	125	93
CCMS	140	136	103

5.2.4. Coupling Measures in Change Management

The coupling measures also helps in change management which is directly related to the prediction of maintenance effort. Change management is the method of getting to know all the changes that needs to be done to the system if there is some change in any of the subsystems. The coupling measures, mainly CBS, give the measure of the coupling of a subsystem with every other subsystem. Hence from CBS measure we will know which other subsystems are dependent on a subsystem. This knowledge will help us to identify the other subsystems that need change when there is a change in a subsystem.

For e.g. from Table 7, we see that HFS, Agenda and Building subsystems are the subsystems that are coupled to UMS subsystem. Hence if there is any modification in UMS subsystem we will know from Table 7, that we need to look into HFS, Agenda and Building to make the necessary changes and we don't have to care about changing the other systems since they are not coupled to UMS and if in turn some other subsystem depends on HFS or Agenda or Building subsystems we might need some changes in those subsystems as a result of modifications to HFS or Agenda or Building subsystems which we will know form Table 7. Once we know which subsystems need change, we can predict the effort needed to make these changes, since we now know the number of changes to be made. The prediction of effort also depends on certain other factors such as programmer experience, time available etc.

Thus the coupling measures directly help in predicting the maintenance cost of the system. Thus with the help of coupling measures, the system design can be changed in order to reduce the maintenance cost and in order to increase the reusability of the generic subsystems.

CHAPTER SIX: CONCLUSION AND FUTURE WORK

6.1. Summary

Object-oriented system is gaining wide attention both in research environments and in industry. A severe problem encountered is the quickly increasing complexity of such systems and lack of adequate criteria and guidelines for good designs. The success of a large scale software system lies in the standardization of well-understood technologies with reduced cost. By reusing the subsystems (modules) can create customized systems economically by building only the parts that are application-specific. Unnecessary reinvention of technology is avoided. Maintenance cost being the costliest step in software development, reducing the cost of maintenance, helps in reducing the total cost of the system to a higher degree. Coupling, which makes the system more complex, is one of the big hindrances to reduce cost and to provide capability of reuse.

Based on standardized terminologies and formalisms given in the literature, we have provided taxonomy of coupling in large scale object-oriented systems. In contrast to the classical notion of coupling being based on the class, we introduced coupling metrics to measure the coupling at the subsystem level. Our approach was based on grouping classes into reusable subsystems early in the design stages by utilizing reuse-specific characterizations. Subsystem boundaries are used to control dependencies throughout the system's development. This helped us in defining metrics for measuring coupling at the subsystem level. Measuring at the subsystem level indicates that we ignored the coupling between the classes inside a subsystem.

The measure is intended to predict the reusability of a subsystem i.e., the reuse of all the classes in a subsystem together, and to predict the maintenance cost of the whole system. These metrics not only helps in predicting the maintenance cost and reusability but also helps in re-factoring the subsystems in an effective way to achieve the lower cost and high reusability. We also provided the analytical evaluation of the metric against the standard metric properties. Also we provided an experimental analysis of the metrics by applying them to a large scale software system. But still, Empirical Validation of the metrics with more data and a few other systems needs to be performed to evaluate the usefulness of these metrics.

Table 10: List of Coupling Metrics defined in this work

#	Coupling Metric	Expansion	Description
1	$CBS(s,s') = \sum_{m \in M(s)} \sum_{m' \in PIM(s')} NPI(m,m')$ CBS (s, s) = 0	Coupling Between Subsystems	Number of invocations of methods in another subsystem from the given subsystem.
2	$CSL = \sum_{s \in S} SC(s)$ $SC(s) = \sum_{m \in M(s)} \sum_{m' \in R_s} NPI(m,m')$ $R_s = \sum_{i \in S-s} PIM(m_i)$ Or $CSL = \sum_{i \in S-s} CBS(s,s_i)$	Coupling at Subsystem Level	Sum of all subsystem coupling (SC) of all the subsystems present in the system. SC of a subsystem can be defined as the number of invocations of methods in other subsystems from the given subsystem.
3	CCMS = Ns + CSL	Coupling Complexity Measure of the Large Scale System	Sum of the number of subsystems and the total coupling at the subsystem level.

6.2. Future Work

There are a lot of works that can be done in future to improvise our work. The potential future works could include the following:

6.2.1. Empirical Validation

To be useful, these measures must be empirically validated using more data and different other systems with respect to external quality attribute namely, reusability and maintenance cost using regression analysis. Our future work will be in empirically validating our metrics against reusability and maintenance cost using different large scale software systems. The future work could include the collection of data to perform the empirical validation of the metrics defined in this work. As discussed in the previous chapter, empirical validation can be performed using the regression models to find the usefulness of the defined metrics.

6.2.2. Dynamic Coupling

In the coupling metrics we have defined in this work, only static coupling is considered (only the method invocations that can be determined at the compile time) i.e., coupling is measured through static code analysis. But in reality of a large scale object-oriented system, the dynamic method invocations also hinder the reusability of the subsystems and increases the maintenance cost of the system. Hence the dynamic coupling also needs to be considered while measuring the coupling between the subsystems by tracing the flow of messages between subsystems at runtime. That could be a good work in the future. Further, cost of measuring the dynamic coupling will be higher when compared to measuring the static coupling. Hence it is

equally important to evaluate if the benefits of the dynamic coupling outweighs the cost of collecting them.

6.2.3. Other Future Work

An automated system could be developed to measure the coupling measures of a large scale system which will further ease the use of these metrics. The future work could also include the research of other dependent variables like reusability and maintenance cost, which is dependent on the coupling. This could help in extending the metrics defined in this work to accommodate the prediction of those other dependent variables. These works could lead to find a better way to reduce the coupling between the subsystems, which in turn will increase the reusability and reduce the maintenance cost, thus helping to develop a successful large scale system with less cost and efforts.

LIST OF REFERENCES

1. Batory, D. and S. O'Malley, The Design and Implementation of Hierarchical Software Systems with Reusable Components. ACM TOSEM, 1992.

2. Yourdon, E. and L.L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice Hall, 1979.

3. Schach, S.R., Software Engineering. Forth ed., McGraw-Hill, Boston, MA, 1999.

4. Myers, G.J., Reliable Software Through Composite Design. Petrocelli / Charter, New York, 1975.

5. Offutt, A.J., M.J. Harrold, and P. Kolte, A Software Metric System for Module Coupling. Journal of Systems and Software, v.20: p. 295-308, March 1993.

6. Page-Jones, M., The Practical Guide to Structured Systems Design. 2nd ed., Yourdon Press, New Jersey, 1988.

7. Budd, T.A., An Introduction to Object-Oriented Programming. 2nd ed. Vol. 879. Addison Wesley Publishing Company, 1997.

8. Eder, J., G. Kappel, and M. Schrefl, Coupling and Cohesion in Object-Oriented Systems. Technical Report, 1994.

9. Hitz, M. and B. Montazeri, Measuring Coupling and Cohesion in Object-Oriented Systems. in Proc. Int'l Symp. Applied Corporate Computing, Monterrey, Mexico, p., v.1, n.4, 1995, http://www.sigs.com/publications/docs/oc

10. Briand, L.C., P. Devanbu, and W. Melo, An investigation into coupling measures for C++. in Proceedings of19th International Conference on Software Engineering, Boston, p. 412-421, May 1997,

11. Briand, L.C., J.W. Daly, and J.K. Wust, A Unified Framework for Coupling in Object-Oriented Systems. IEEE Transactions on Software Engineering, v.25(1): p. 91-121, 1999.

12. Chidamber, S.R. and C.F. Kemerer, A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, v.20(6): p. 476 - 493, 1994.

13. Churcher, N.I. and M.J. Shepperd, Towards a Conceptual Framework for Object-Oriented Design. ACM SIGSOFT, v.20(2): p. 69-76, 1995.

14. Harrison, R., S. Counsell, and R. Nithi, Coupling Metrics for Object-Oriented Design. in 5th International Software Metrics Symposium Metrics, p. 150-156, 1998,

15. Li, W. and S. Henry, Object-oriented metrics that predict maintainability. Journal of Systems and Software, v.23(2): p. 111-122, 1993.

16. Abreu, F., M. Goulao, and R. Esteves, Toward the Design Quality Evaluation of Object-Oriented Software Systems, in Proceedings from the Fifth International Conference on Software Quality. Oct 1995: Austin, Texas.

17. Binkley, A.B. and S.R. Schach, Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. in Proceedings of the 20th international conference on Software engineering: IEEE Computer Society Washington, DC, USA, p. 452 - 455, 1998,

18. Basili, V. and R. Reiter, Evaluating automatable measures of software Models. in IEEE Workshop on Quantitative Software Models, NY, p. 107-116, 1979,

19. Weyuker, E., Evaluating software complexity measures. IEEE Transactions on Software Engineering, v.14: p. 1357-1365, 1988.

20. Briand, L.C., et al., Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. The Journal of Systems and Software, 1998.

21. Hosmer., D.W. and S. Lemeshow, Applied Logistic Regression. John Wiley & Sons, 1989.