December 2006

The Cohesion-Based Requirements Set Model for Improved Information System Maintainability

Joanne Hale
The University of Alabama

David Hale
The University of Alabama

Randy Smith
The University of Alabama

The Cohesion-Based Requirements Set Model for Improved Information System Maintainability

Nelson E. Barnes, Jr.
The University of Alabama
nbarnes@cba.ua.edu

Joanne E. Hale
The University of Alabama
jhale@cba.ua.edu

David P. Hale
The University of Alabama
dhale@cba.ua.edu

Randy K. Smith
The University of Alabama
rsmith@cs.ua.edu

ABSTRACT

Complexity and system structure are two factors that have been proven to significantly affect the software maintenance costs incurred over the life of the system. The concept of cohesion, which is normally associated with software design, is commonly used to measure the degree to which elements of a module are related based on the structure of the design. Currently, the responsibility of designing a cohesive system falls solely upon the shoulders of the developer who typically possesses limited knowledge about the system and the environment in which it operates. It is proposed in this research that it may be more advantageous to apply the principle of cohesion at an earlier phase of the software development life cycle, thus placing more responsibility on the analyst who has a better understanding of the business. This paper proposes that by using the CBRS technique, the overall maintainability of the resulting system will be positively affected.

Keywords

Cohesion, requirements structuring, requirements validation, requirements engineering, software maintenance

INTRODUCTION

The amount of maintenance that can occur over the total life of a large system can be enormous. The vast majority of these activities can be described as corrective, adaptive, perfective or preventive. More specifically, software maintenance includes tasks such as error corrections, changes (amendments or enhancements) and improvements to operational software (Bhatt et al, 2004). As a result of this, software maintenance is often viewed as an iceberg because only the very tip of the maintenance effort is visible (Canning, 1972). Maintenance activities roughly account for 60% of all effort that is expended by a development organization. In addition, software enhancements consume more than 50% of total maintenance costs (Pressman, 1992). The results of a factor analysis done by Lientz and Swanson (1981) identified user knowledge, programmer effectiveness, and product quality as the top three problem factors for software maintenance. Within the context of their experiment, user knowledge covered both lack of user understanding and inadequate user training, while programmer effectiveness included the skills of the programmers. Finally, product quality covered the adequacy of system design specs, the quality of original programming and documentation quality. It is within the scope of these factors that we drive towards utilizing the notion of cohesion.

Cohesion has been commonly defined as a measure of the degree to which elements of a module belong together (Mallens, 1997). The concept of cohesion is normally considered during software design and development, during which developers strive for components which are highly cohesive (Beiman and Kang, 1995). The cohesiveness of the system is frequently used to measure characteristics such as the separation of responsibilities of the components, the independence of the components, the quality of the software design, the fault proneness of the system, the modularization of the components, and the amount of reuse that a component provides (Etzkorn et al, 1997; Haley et al, 2004; Lee et al, 2001; Mallens, 1997).

This article proposes applying these same general principles of cohesion at an earlier phase of the software development life cycle. By doing this, the decision of determining the level of cohesion in the system modules becomes the responsibility of

the system analyst, who has more knowledge of the work system and thus has the ability to predict what requirements are more likely to change, instead of the software designer who has less contextual knowledge of the domain. Using this technique, the analyst may be able to positively affect the overall maintainability of the system by applying a synthesis or expansion approach when gathering, structuring, and communicating requirements rather than using an approach based on analysis or reduction.

BACKGROUND

Software Maintenance

Martin and McClure (1983) state that software maintenance consists of the work done on a software system after it is deployed and becomes operational, while the IEEE defines software maintenance as the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment (IEEE, 1993). This work generally includes tasks such as error corrections, changes (amendments or enhancements) and improvements to operational software. Gibson and Senn (1989) emphasize that the issue of software complexity pervades the issue of maintainability. They also mention that complexity causes two general problems in maintenance. First, as the complexity of a system increases, so does the difficulty to understand and maintain it. Secondly, complex systems require more corrective maintenance throughout the life of the system (Gibson and Senn, 1989). So, as systems get larger and continue to age, it will become progressively difficult to avoid the maintenance that they will require. However, we can alleviate this issue by improving the initial quality of the system design. From a maintenance perspective, this notion of improving design quality most commonly involves improving the overall structure of the system and the complexity of the system as they relate to the volatility of the requirements describing the system.

Cohesion

Yourdon and Constantine (1979) state that cohesion is a qualitative measure which falls into one of the following categories in order of lowest to highest:

Coincidental –	Parts of a module are grouped arbitrarily.
Logical –	Parts of a module are a set of related functions.
Temporal –	Elements of a module are processed within a certain time period.
Procedural –	Elements of a module follow a certain sequence of execution.
Communication –	Elements of a module produce the same output or operate on the same input data.
Sequential –	The output from one module element is the input to another module element.
Functional –	The operations of a module participate in accomplishing a single well-defined task.

Comprehension is often cited as the most time consuming component of the maintenance activity. High cohesion leads to code that is easier to comprehend, increases the likelihood of reuse, and creates components whose complexity is kept manageable. The level of cohesion decreases if the responsibilities of a class are unrelated -- in other words, they perform a wide range of disparate actions or operate on diverse types of data.

Although significant progress has been made in the area of developing more extensive cohesion metrics, no one cohesion method currently exists that is accepted as a standard. Under the category of structural cohesion metrics, Chidamber and Kemerer (1994) originally proposed LCOM (Lack of Cohesion of Methods) which has become one of the most widely known cohesion metrics for object-oriented systems. Subsequent research has produced similar metrics such as LCOM2, LCOM3, LCOM4, and LCOM5 (Beiman and Kang, 1995; Chidamber and Kemerer, 1994; Hitz and Montazeri, 1995; Lee et al, 2001), as well as TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion) (Beiman and Kang, 1995). Other proposed cohesion metrics include specialized metrics such as ontology cohesion metrics (Yao et al, 2005), knowledge-based metrics (Kramer and Kaindl, 2004), and metrics for distributed systems (Cho et al, 1998). While these metrics have proven

effective for measuring modules during the design phase, no research was found that took the notion of cohesion and applied it to an earlier phase in the life cycle.

Requirements Structuring

The significance of representing requirements so that the system behaves and evolves as intended has increased as systems have become more integrated and complex. The task of effectively organizing and categorizing requirements can be surprisingly complex in a fluid and dynamic environment (Haley et al, 2004). The difficulty involved in developing a powerful yet flexible classification scheme is partly due to the heterogeneity of factors usually considered during the requirements elicitation phase.

Requirements structuring mechanisms include techniques such as use cases, creating a system of subsystems, decision trees, and Entity Relationship Diagrams (ERD). In addition, techniques such as Rapid Application Development (RAD) and Object-Oriented Analysis (OOA) have requirements structuring and analysis techniques embedded within the frameworks. Although each of these techniques are effective within the frameworks in which they are used, no common as well as flexible architecture or tools exist for structuring requirements. Additionally, all of the aforementioned techniques are typically directed towards a platform and the development phase instead of a maintenance goal. For example, while use cases are very useful in amalgamating requirements into modules, the selection mechanisms for doing so are interpreted rather loosely, which may result in modules that are not comprehensive. To supplement the effectiveness of these techniques, a method is needed to drive the manner in which modules are structured using these techniques.

MAPPING THE COHESION LEVELS TO REQUIREMENTS

The goal of this research stream is to determine if applying the concept of cohesion to requirements gathering, structuring, and communication results in improved system maintainability. In order to do this, the original cohesion definitions as stated by Yourdon and Constantine (1979) were first modified to fit within the context of requirements rather than design. The refined definitions of cohesion are shown below:

Coincidental –	Requirements are grouped together although limited constructive relationships exist between them.
Logical –	Requirements are grouped together because they represent specifications for alternative flows or exceptions.
Temporal –	Requirements are grouped into a module because they provide specifications that are processed within the same limited time period.
Procedural –	Requirements are grouped because they relate to a specific procedure or algorithm.
Communication –	Requirements are grouped because they relate to a task or a set of tasks that produce the same output data or operate on the same input data.
Sequential –	Requirements are grouped because their output is the input to another requirements set.
Functional –	Requirements are grouped because they focus on a single well-defined task.

By describing cohesion in this manner, we can associate a set of requirements with the appropriate cohesion level that it represents. During the requirements elicitation phase, it is imperative for the analyst to normalize the requirements that are obtained to ensure that the requirements are correctly classified. Although this is an important step in the requirements phase, it is assumed to have taken place prior to the application of the CBRS model and is therefore out of the scope of this paper.

THE COHESION-BASED REQUIREMENTS SET MODEL

After successfully refining the cohesion definitions to fit within the context of requirements, the next step is to develop a scheme for structuring the requirements based on these definitions. Currently, neither an overall architectural rationale nor a business architecture driven approach for structuring requirements exists for the analyst. This research begins to address this gap with the proposed Cohesion-Based Requirements Set (CBRS) model. The most general CBRS model is depicted in Figure 1 as a non-directed graph along with two possible paths through the model. In the CBRS model, the cohesion levels are represented as nodes and the arcs represent potential binary precedence. Underlying the proposed method is the proposition that there exists a logical structuring order of requirements derived from the fully enumerated *general CBRS model* that minimizes maintenance costs of the resulting system.

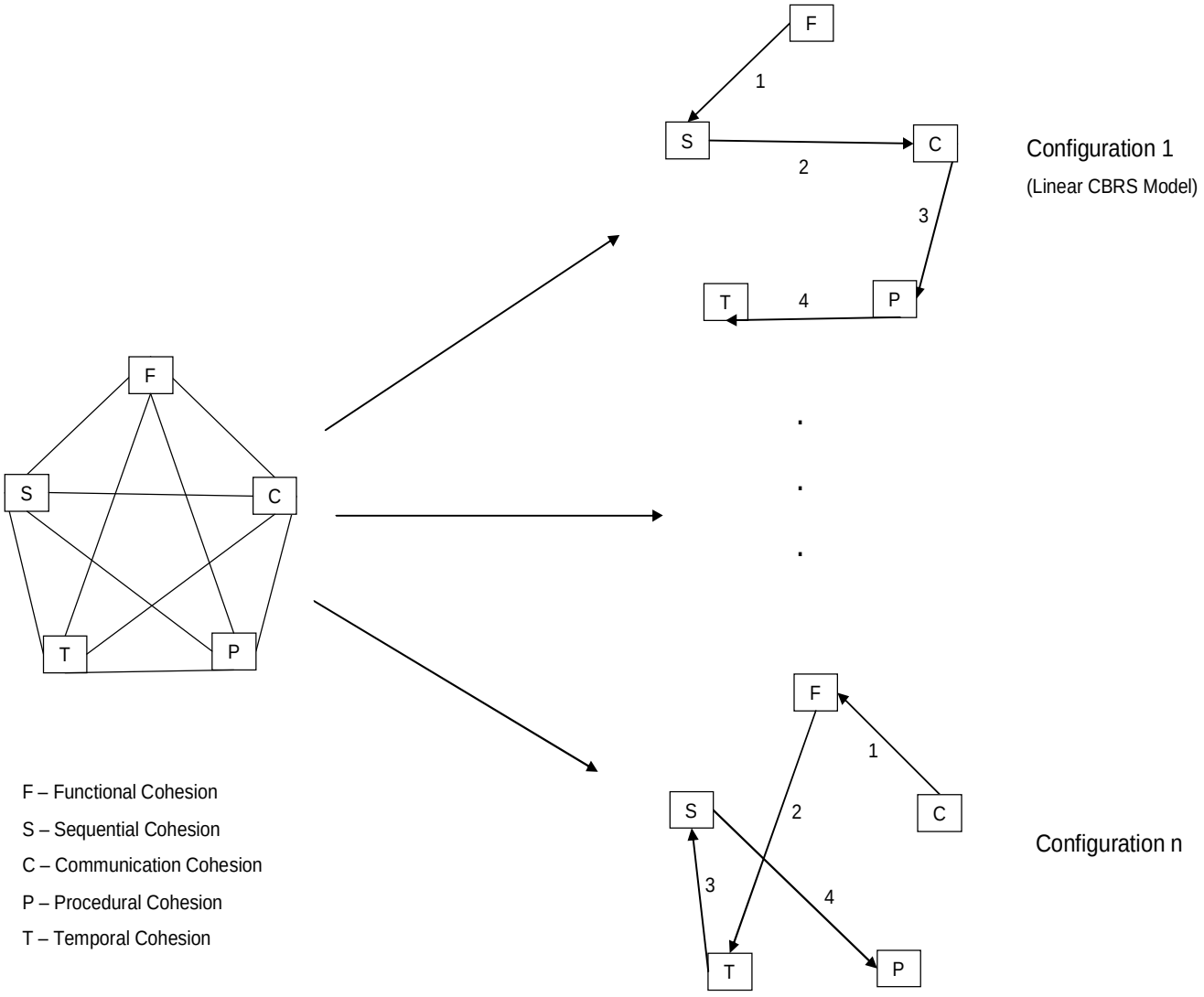

Figure 1: General CBRS Model for Requirements Structuring with Sample Configurations

Using the concept of modularization, the objective is to construct comprehensive requirement sets that provide an integrated means for improved maintainability. The nodes in the CBRS Model represent a cohesion-based view of the requirements after the applicable rules for that node have been applied. Each of the requirement sets within a node is simply a collection of individual requirements that, when viewed as a single module, exhibit the cohesion characteristics of the node. Due to the fact that each cohesion-based view represents the overall system from a different perspective, a different method was needed to create requirement sets based on the view used. These view-specific rules are listed in Table 1. Where one may question the omission of logical and coincidental cohesion from both the model and the views listed in Table 1, they have been excluded because they typically should be avoided due to their weak association nature.

Cohesion- Based View	**Method for Creating the View**
Functional	1) Select a task. 2) Group all the functional requirements related to the task. 3) If a requirement shows up in more than one view, consider separating the requirement into two or more additional requirements that are more specific.
Sequential	1) Identify the data to be passed. 2) Select the requirement or requirement set that generates the output. 3) Select the requirement or requirement set that receives the input. 4) If the sequence is too complex or ambiguous, decompose the requirements into smaller sets.
Communication	1) Identify the data to be operated on. 2) Select the requirement or requirements that operate on the data. 3) If the requirement set is too complex or ambiguous, decompose the requirements into smaller sets.
Procedural	1) Select a procedure or algorithm. 2) Group all the requirements related to the procedure or algorithm. 3) If a requirement shows up in more than one view, consider separating the requirement into two or more additional requirements that are more specific. 4) If several related requirements show up in multiple procedures, consider isolating them as a single requirement set.
Temporal	1) Identify the time period. 2) Select the requirements that are applied during this time period. 3) If several requirements show up in multiple temporal sets, decompose them into smaller functional, sequential, communication or procedural sets.

Table 1: Methods for Generating the Cohesion-Based Views

The number of different paths and the total number of iterations through the cohesion nodes in the general model increases exponentially as the number of requirements increase. However, regardless of the number of requirements that you have, the following fundamental steps should be utilized when applying the CBRS Model to structure your requirements:

- Step 1: Select a cohesion-based view
- Step 2: Apply the structuring method defined for the selected view
- Step 3: Validate the requirement sets created for the selected view and note any ambiguities or incompleteness
- Step 4: Repeat steps 1-3 as needed

One configuration using these steps may consist of first creating the temporal views, then the sequential views, followed by the communication views, the functional views, the logical views, and finally the procedural views. Ideally, the chosen configuration for a specified target system would be determined by the strategic goals of the business which are determined by various factors such as the application size and type, the amount of code reuse that is desired, the expected volatility of the requirements, and the application domain.

Experience and intuition may suggest that requirements are structured in a linear fashion based on the levels of cohesion that are desired between the attributes, methods, and objects during the design phase. This is the logic behind the path representing the linear CBRS model in Figure 1. However, a more detailed representation of this linear sequential CBRS model is shown in Figure 2. In this illustration of the model, the requirements are structured in a sequential fashion based on the levels of cohesion. If the design goal is to produce a highly cohesive system, then the rules dictate that we should initially begin to combine requirements at the highest level of cohesion and subsequently proceed down the cohesion levels. This model is built on the hypothesis that the relative desirability of design-phase cohesion levels maps directly to the requirements phase. In this case, requirements are grouped into functional views and validated first. Next, the sequential views are added, followed by the communication views, then the procedural views, and finally the temporal views. It is important to note that this process is iterative and continues until the analyst is satisfied with the resulting structures and how they affect the maintainability of the overall system.

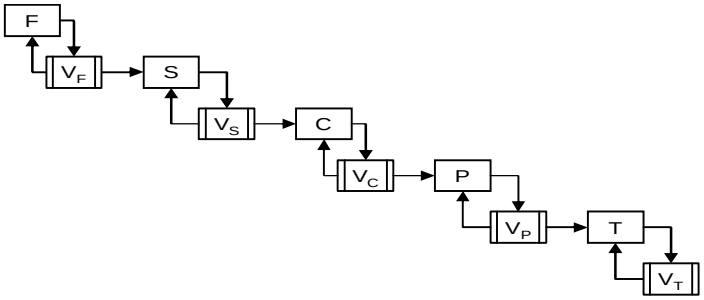

Figure 2: Linear CBRS Model for Requirements Structuring

We then generalized the usage of the linear CBRS model into two distinct techniques which we have designated as Type I and Type II usages. One scenario illustrating how the Type I technique could be applied to the linear CBRS model is shown in Figure 3. In this Type I usage, the first step is to create the functional views. From Table 1, it follows that this step may focus heavily on identifying behavioral tasks that the system must perform. The second step is to create the sequential views. Based on the methods from Table 1, this would include identifying the data involved, followed by the sender and the receiver. Some of the requirement sets created in this step may also be added to the existing functional sets that were created in step 1. Similar steps are continued for the next three cohesion levels. Again, logical and coincidental cohesion are explicitly omitted because they typically should be avoided due to their weak association nature.

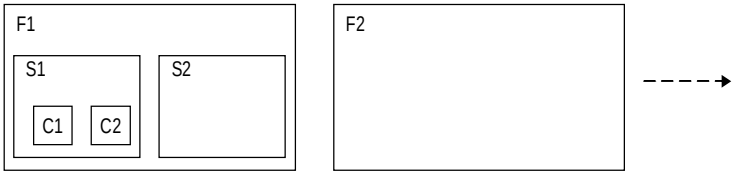

Figure 3: Type I Usage of the Linear CBRS Model

In some situations, the analyst may be unable or unwilling to combine requirements sets. In either case, the analyst would employ the Type II technique which results in groups of tightly related requirement sets that are divided based on cohesion levels. For instance, these modules may be used to design isolated components that are constructed solely for the purpose of

modeling a set of volatile requirements for maintenance or testing purposes. With the knowledge that 80% of the total cost of ownership of a system is in maintenance, the analyst may opt to extract certain related requirements and place them into segregated modules. This added insight as to the context in which the system will ultimately function allows the analyst the ability to specify how these volatile requirements are functionally related to the overall system. An example of this case is shown in Figure 4. One final variation on the linear model is a hybrid in which the analyst still follows the order of the linear model, but some requirements types may be skipped or relaxed based on the unique needs of the business.

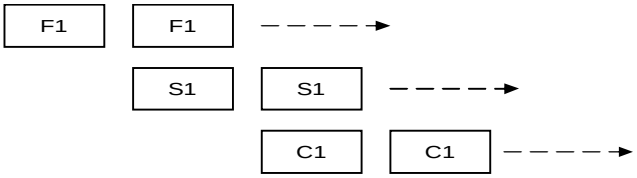

Figure 4: Type II Usage of the Linear CBRS Model

Applying the CBRS Model to an Example

Program understanding means having the knowledge of what a system does, how a system operates within its environment, identifying where in the system changes are to be effected and having in-depth knowledge of how the parts to be corrected or modified work (Takang and Grubb, 1996). Take the subset of requirements for the security component of an HR system shown in Table 2 for instance.

Req #	Description
186	The system must prevent workers or supervisors from updating cases to which they are not assigned.
187	The system must provide an audit trail, printable on demand, for data on the system. The audit trail must include: - the old value - the new value - the time/date of change - the user ID of the person who made the change - the computer from which the data was modified
188	The system must automatically time out and require the user to re-authenticate via user ID and password when there has been a period of inactivity of 30 minutes. Any data entered prior to the time out must be saved and the user must be returned to the point where he/she left off when logging back in.
189	The system must encrypt any e-mail, reports, or other data sent to addresses outside the State network or residing outside the State firewall using 128 bit SSL/VPN encryption.
190	The system must allow an administrator to restrict access to a case to authorized users.
191	The system must employ a profile based security methodology that allows user access to functional areas based on user security level.
192	The system must provide for application security by requiring users to enter a user ID and password / PIN to logon to the system.
193	The system must ensure that any external access has a method of authenticating the

	user (i.e. user ID and password or PIN).
194	The vendor's security solution must comply with National Institute of Standards and Technology (NIST) security guidelines.
195	The system must prevent users from making changes to system-entered data that is determined to be critical data.
196	The system must support all necessary levels and classes of security to protect employee information, such as SSN.
197	The system must allow users to generate a Medicaid state/federal adoption assistance document.
567	The system must provide the ability either manually, based upon user's appropriate security level, or automatically with specific criteria, to expunge records from the Child Abuse and Neglect Central Registry.

Table 2: A Subset of the Requirements for the Security Component of an HR System

Using this subset, we applied the linear CBRS model by making use of the fundamental steps for structuring requirements. As required by the model, the initial step was to create the functional view which is illustrated in Table 3. Using the method specified in Table 1 for creating a functional view, the tasks were first identified and then the requirements associated with each task were added to the set. After creating all possible sets for the functional view, we then applied the applicable methods for constructing the subsequent views in order to create the requirement sets for the sequential view in Table 4, the communication view in Table 5, and the procedural view in Table 6.

#	Task	Requirement Set
1	Validate user access	o The system must provide for application security by requiring users to enter a user ID and password/PIN to logon to the system. (192) o The system must ensure that any external access has a method of authenticating the user (i.e. user ID and password or PIN). (193) o The vendor's security solution must comply with National Institute of Standards and Technology (NIST) security guidelines. (194)
2	Restrict user access	o The system must prevent workers or supervisors from updating cases to which they are not assigned. (186) o The system must allow an administrator to restrict access to a case to authorized users. (190) o The system must employ a profile based security methodology that allows user access to functional areas based on user security level. (191)

Table 3: Functional View

#	Data	Output Requirement Set	Input Requirement Set
1	User access credentials	o Validate user access (Functional Task #1)	o Restrict user access (Functional Task #2)

Table 4: Sequential View

#	Data	Requirement Set
1	System-entered data *(Analyst Note: More detail is needed about the data required here)*	o The system must provide an audit trail, printable on demand, for data on the system. The audit trail must include: • The old value • The new value • The time/date of change • The user ID of the person who made the change • The computer from which the data was modified o The system must encrypt any e-mail, reports, or other data sent to addresses outside the State network or residing outside the State firewall using 128 bit SSL/VPN encryption. (189) o The system must prevent users from making changes to system-entered data that is determined to be critical data. (195) o The system must support all necessary levels and classes of security to protect employee information, such as SSN. (196)

Table 5: Communication View

#	Algorithm	Requirement Set
1	Backup data	*(Analyst Note: No specific requirements provided for this algorithm; More detail is needed – See Requirement #187)*
2	Encrypt outgoing data	*(Analyst Note: No requirements provided for this algorithm; More detail is needed – See Requirement #189)*
3	Expunge records	*(Analyst Note: No requirements provided for this algorithm; More detail is needed – See Requirement #567)*

Table 6: Procedural View

After the different views have been created by the analyst, they are passed on to the developer or programmer. These views are then used to create workflows, sequence diagrams, use cases, ERDs, data flow diagrams, etc. In this case, the primary purpose of the particular view used is to provide a structured manner for utilizing the aforementioned techniques. In addition to providing the analyst the ability to influence the system structure, we gain from the insight he or she may have regarding certain requirements which are likely to change. This allows the analyst to structure the requirements in an effort to reduce the amount of maintenance will be needed later.

There are several additional benefits associated with applying the CBRS technique. First of all, each view provides information that supplements an existing requirement structuring method or a design framework. For instance, the major flaw with the use case approach is that it is not comprehensive enough to include all associated requirements for a particular task. However, the functional view allows the analyst to associate all requirement types to an individual use case during the analysis phase as opposed to strictly the functional or behavioral requirements. Secondly, by using the requirements to structure the design, the model provides the ability to trace any particular requirement through design, code, and even testing. This level of traceability is vital from a maintenance perspective because as requirements change, it is imperative to know where the changes will need to be made and how the system will be affected. Finally, an added bonus of the CBRS Model is the ability to identify gaps in the requirements. While creating the procedural view for the example, we discovered that although a requirement specifying the need for each of the algorithms exists, the details for implementing the algorithms were not included. By recognizing this early in the process, analyst is able to reduce the amount of ambiguity in the requirements and thus reduce the amount of corrective maintenance needed in the future.

CONCLUSIONS AND FUTURE WORK

This paper presented a conceptual model for structuring requirements using the notion of cohesion levels. The model was derived from the pairings of the requirements to the cohesion levels. It is proposed that this general model, deemed the Cohesion-Based Requirements Set (CBRS) model, will improve the cohesion at the design stage as well as improve the overall quality of the system in the area of reuse and maintainability.

The next step in this research is to conduct empirical studies to gain insight into which paths through the general model provide the optimal structure in terms of maximizing component reuse and decreasing maintenance costs when analysts can predict areas in which requirements are more likely to occur. There are numerous sequences that can be constructed through the CBRS model and these planned empirical studies will serve as a tool to determine the most effective sequences. Future phases will empirically identify sequence effectiveness based on project types as well as application domain.

REFERENCES

1. Bhatt, P., Shroff, G., and Misra, A. (2004) Dynamics of Software Maintenance, *ACM SIGSOFT Software Engineering Notes*, 29(5).
2. Bieman, J. and Kang, B. K. (1995) Cohesion and reuse in an object-oriented system, *Proceedings of ACM Symposium on Software Reusability (SSR'95)*, 259-262.
3. Canning, R. (1972) The Maintenance Iceberg, *EDP Analyzer*, October 1972.
4. Chidamber, S. R. and Kemerer, C. F. (1994) A Metrics Suite for Object Oriented Design, *IEEE Transactions on Software Engineering*, 20(6), 476-493.
5. Cho, E. S., Kim, C. J., Kim, D. D., and Rhew, S. Y. (1998), Static and dynamic metrics for effective object clustering, *Proceedings of Asia Pacific International Conference on Software Engineering 78-85.*
6. Etzkorn, L. H., Gholston, S. E., Fortune, J. L., Stein, C. E., Utley, D., Farrington, P. A., and Cox, G. W. (2004), A comparison of cohesion metrics for object-oriented systems, *Information and Software Technology*, 46(10), 677-687.
7. Gibson, V. R. and Senn, J. A. (1989), System Structure and Software Maintenance, *Communications of the ACM*, 32(3), 347 – 358.
8. Haley, D. T., Nuseibeh, B., Sharp, H. C., and Taylor, J. (2004), The Conundrum of Categorizing Requirements: Managing Requirements for Learning on the Move, *Proceedings of the 12th IEEE International Requirements Engineering Conference.*
9. Hitz, M. and Montazeri, B. (1995), Measuring Coupling and Cohesion in Object-Oriented Systems, *Proceedings of International Symposium on Applied Corporate Computing*, Monterrey, Mexico.

10. IEEE (1993), IEEE Std. 1219: Standard for Software Maintenance, IEEE Computer Society Press.
11. Kramer, S. and Kaindl, H. (2004), Coupling and cohesion metrics for knowledge-based systems using frames and rules, *ACM Transactions on Software Engineering and Methodology*, 13(3), 332-358.
12. Lee, J. K., Jung, S. J., Kim, S. D., Jang, W. H., and Ham, D. H. (2001), Component identification method with coupling and cohesion, *Proceedings of Eighth Asia-Pacific Software Engineering Conference*, 79-86.
13. Lientz, B., P. and Swanson, E. B. (1981), Problems in Application Software Maintenance, *Communications of the ACM*, 24(11).
14. Mallens, P. (1997), Business Rules-Based Application Development, Database Newsletter, 25(1).
15. Martin, J. and McClure, C. (1983), Software Maintenance: The Problem and Its Solutions, Prentice Hall, New Jersey.
16. Pressman, R. S. (1992), Software Engineering: A Practitioner's Approach, McGraw-Hill, New York.
17. Takang, A. A., and Grubb, P. A. (1996), Software Maintenance Concepts and Practice, Thompson Computer Press, London, UK.
18. Yao, H., Orme, A. M., Etzkorn, L. H. (2005), Cohesion Metrics for Ontology Design and Application, *Journal of Computer Science*, 1(1), 107-113.
19. Yourdon, E. and Constantine, L L. (1979), Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice-Hall, Englewood Cliffs, NJ.