

# Defining Maintainable Components in the Design Phase

Orest Pilskalns and Daniel Williams  
ENCS, Washington State University  
Vancouver, WA  
orest.williada@vancouver.wsu.edu

Anneliese Andrews  
EECS, Washington State University  
Pullman, WA  
aandrews@eecs.wsu.edu

## Abstract

*During Component Based Software Engineering it is important for component developers to design components that show high cohesion within a component and low coupling between components. Empirical data shows that software artifacts possessing these properties are easier to maintain. Current practice in design metric evaluation relies on extracting structural metrics from individual UML views. This paper defines a dynamic approach that collects metrics during execution of a model that integrates both UML Class and Sequence Diagrams. These design metrics are used to evaluate component choices by examining cohesion and coupling properties. We base our design metrics on code metrics that have been positively correlated with maintainability and quality. We provide an empirical study that demonstrates a positive correlation between design and code metrics.*

## 1 Introduction

Software design increasingly includes component based software development [7]. This happens from two perspectives: (1) a software designer wants to use components and needs to define how components fit with the remainder of the design. (2) a software designer wants to design components for reuse. This could be part of a product line architecture or part of a set of components developed for reuse. In either case, one needs to determine all component boundaries and interfaces before implementation. Often maintainability and quality are the important drivers when making design decisions concerning component boundaries. Arisholm et al. [1] showed that dynamic coupling and cohesion metrics are linked to software maintainability and quality. Our goal is to define dynamic coupling and cohesion measures for classes and components that can be used in the design phase to evaluate maintainability and quality of the components based on these metrics.

Our measures consider both structural and behavioral as-

pects of coupling and cohesion. This requires (1) combining class diagrams and sequence diagrams into a model that reflects both structure and behavior, and (2) executing this model based on an expected operational profile to collect measures.

Section 2 describes existing work on components and maintainability metrics. Section 3 describes how we use an integrated UML model to evaluate components. We define cohesion and coupling metrics based on an integrated model and an operational profile. Section 4 illustrates our method on an example. Section 5 describes the results of an empirical investigation correlating design metrics to code maintainability and quality metrics. Section 6 draws conclusions and suggests further work.

## 2 Background

In [5], Briand and Wust summarize empirical results concerning code coupling and cohesion metrics that have been shown to correlate with software quality and maintainability. Of the 37 different coupling metrics used in empirical studies only a few had a positive, statistically significant relationship with quality and maintainability ( $p < 0.01$ ). The metrics that showed a positive relationship included Response For Class (RFC) [6], Other Method-Method Import Coupling [9], and Information-Flow-Based Coupling (ICP) [8]. In addition, of the 12 cohesion metrics used in empirical studies only one had two empirical results showing a significant relationship with quality, the Information-Flow-Based Cohesion (ICH) [8]. These metrics rely on both structural and behavioral (dynamic) information. For example the OMMIC metric relies on inheritance information (structural) and the actual number of invoked methods (behavioral) in a class.

In [1], Arisholm et al. define and investigate dynamic object-oriented code metrics for both code and design. They propose that static metrics do not properly reflect modern object-oriented code due to the increased use of inheritance and dynamic binding. Empirical data links their code metrics to quality and maintainability. They state that applying

these metrics to UML Models introduces some problems for guarded messages. Therefore, some coupling may be underestimated. To avoid this problem, our approach uses an operational profile to estimate path usage. In addition, our approach uses properties from both structural and behavioral diagrams allowing us to calculate more sophisticated metrics for designs.

The work described in this paper not only extends the work described above, but provides a new way to evaluate UML components using dynamic metrics. Previous work is concerned more with methodology than a detailed approach to quantitatively evaluate designs via metrics.

### 3 Component Evaluation Approach

During design in Component Based Software Engineering (CBSE), the designer has to define components and component interfaces. Often there are choices and ideally all components should have high cohesion and low coupling for quality and maintenance purposes. Candidate components are evaluated based on their cohesion and coupling. When designing a component based system using the UML, cohesion and coupling measures are computed for each candidate component and they form the basis for evaluating candidate components and for choosing between them. Because cohesion and coupling metrics use information from multiple UML views, it is necessary to combine them into an integrated design model.

Pilskalns et al. [11] provide a testing model that transforms Sequence Diagrams into an Object Method Directed Acyclic Graph (OMDAG). This approach uses test-cases to traverse different paths in the graph. Here we adapt this approach to component evaluation. We replace test-cases with an operational profile, add Class Diagram information (structural information) to the OMDAG, and collect metrics for cohesion and coupling. Our approach to component analysis consists of the following steps:

1. Build an integrated model (OMDAG) of Class Diagrams and Sequence Diagrams.
2. Identify candidate components in the integrated model.
3. Define an operational profile (work load).
4. Execute the operational profile (traverse paths in graph) and collect metrics.
5. Compare candidate components and make decisions.

Sections 3.1 through 3.5 describe each step. Figure 1 outlines the approach.

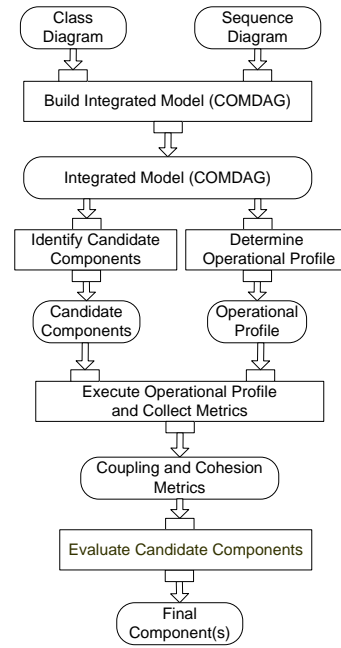


Figure 1. Component Evaluation

#### 3.1 Building the integrated model (COMDAG)

Building the integrated model consists of three steps. The first step maps Class Diagrams into tuples, called Class Tuples (CT). A class tuple is a mathematical representation of a class and is similar to the idea of representing a class using the XMI specification. The second step consists of mapping Sequence Diagrams into an Object Method Directed Acyclic Graph (OMDAG). The third step consists of combining each OMDAG with the CT information. This results in the CT Object Method Acyclic Graph (COMDAG). The COMDAG represents the integrated model that combines Class Diagrams and Sequence Diagrams.

##### 3.1.1 The Class Tuple (CT)

The CT consists of a class name, attributes (represented as tuples if non-primitive) from the class and super classes (if applicable), and methods (represented as tuples) for the class and super classes (if applicable). Classes may contain non-primitive attributes that are defined by other classes. Thus, CTs may contain other CTs by definition. A Class Tuple of a class ( $c$ ), the Attribute Tuple, the Method Tuple, and the method's Parameter Tuple have the following forms:

$$CT(c) = \langle \{ \{ Parent\ CT \} \}, \{ \{ Attribute \} \}, \{ \{ Method \} \} \rangle \quad (1)$$

$$Method = \langle method\ name, return\ type, \{ Parameter \} \rangle \quad (2)$$

$$Attribute = \langle attribute\ name, attribute\ type \rangle \quad (3)$$

$$Parameter = \langle parameter\ name, parameter\ type \rangle \quad (4)$$

Any or all of the tuple elements can have null values denoted by a null place holder for that element. The *Parent CT* has the same structure as the CT, thus part of the CT is recursively defined.

### 3.1.2 The Object Method Directed Acyclic Graph (OMDAG)

The OMDAG maps the dynamic information in a Sequence Diagram to a directed acyclic graph. The OMDAG is created by mapping object and sequence method calls from a Sequence Diagram to vertices and arcs in a directed acyclic graph. The mapping between Sequence Diagram and OMDAG preserves the relationships in the Sequence Diagram. The mapping consists of (1) associating methods in the Sequence Diagram with their originating objects, (2) traversing the Sequence Diagram for the purpose of mapping successive method executions to edges of the OMDAG. These edges are also annotated with any conditions the Sequence Diagram may impose on their execution.

The OMDAG is a tuple  $\langle V, E, s \rangle$  where  $V$  is a set of vertices,  $E$  is the set of edges, and  $s$  is the starting vertex. Each vertex,  $v$ , is defined by the tuple  $v = \langle o, m, \{ARGS\}, c \rangle$ , where  $o$  is an object,  $m$  is the method or return call,  $ARGS$  is a set of arguments, and  $c$  is a class name. Arguments are the actual method parameters that are used in a Sequence Diagram. The actual parameters may have values associated with them. The  $ARGS$  tuple is defined with the following triple:  $\langle type, name, value \rangle$ . Note that only the class name is known in the Sequence Diagram; details about the class are not available. An edge,  $E$ , is represented by the tuple  $\langle v_i, v_{i+1} \rangle$ . For details on mapping the Sequence Diagrams to an OMDAG see [11].

### 3.1.3 The CT Object Method Directed Acyclic Graph (COMDAG)

The final step in building an integrated model is to combine OMDAG and CT information. The COMDAG is built by replacing each class name,  $c$ , in each OMDAG vertex  $v$  with the corresponding CT( $c$ ). This results in an expanded vertex definition:

$$v = \langle o, m, \langle ARGS \rangle, CT(c) \rangle \quad (5)$$

## 3.2 Defining Components in the Model

A COMDAG consists of a set of vertices,  $v_1 \dots v_z$ . Each vertex  $v$ , as defined in Equation 5, contains a class  $c$ . A candidate component consists of a proper subset of the vertices in a COMDAG and the classes associated with those vertices. The subset of vertices is called the CV set. Vertices

of the CV set may or may not be connected to other vertices in the set. To define a component, select vertices from the COMDAG and place them into a CV set. The class set that is associated with the CV set is called the component class (CC) set. The CC can be defined as follows:

$$CC = \bigcup_{\forall v \in CV} \{c \mid c \text{ in } v\} \quad (6)$$

Our approach does not automate the process of defining components. Rather it provides the designer with metric feedback once candidate components have been selected. The process of deciding which vertices to include in a component CV set should use domain knowledge. The edges in the COMDAG gives the designer some indication of how classes are coupled, hence placing vertices in a CV that create a large number of *boundary edges* may not be a wise choice. Typically a designer looks for natural boundaries where there is a low number of connections between groups of vertices. A designer typically chooses a set of vertices that has a minimal amount of connections with the rest of the graph. Thus a min-cut algorithm could be employed, if one wanted to automate the generation of a CV. The result of this step is a set of candidate components with their associated CV and CC sets.

## 3.3 Defining an Operational Profile

An operational profile exercises a system under the conditions in which it is expected to operate [10]. In other words, the system is exercised with a suite of test cases and their frequency that represents how the system will be used in practice. In [11], executing a test case results in executing a path through the graph. Thus we can describe an operational profile in terms of a set of paths through the COMDAG and their frequency.

We record an operational profile in a table consisting of a path definition (as a sequence of nodes) and the number of times (frequency) the path should be traversed. The designer determines the number of traversals and which paths represent how the system will be used based on domain knowledge and/or requirements (Use-Cases).

## 3.4 Executing and Collecting Metrics

Operational profile execution consists of traversing the COMDAG using the path and frequency specified in the operational profile. While traversing the COMDAG, the following metrics are collected:

### 3.4.1 The RFC Metric in Designs

The RFC of a class is the cardinality of the set of all method invocations that may be executed in response to a message received by an object of that class. The RFC for designs

measures the cardinality of the response set for a class by traversing paths in the COMDAG. Let  $P = \{v_1 \dots v_n\}$ , where  $P$  is a path in the COMDAG and  $v_i$  ( $i = 1 \dots n$ ) is a vertex defined by Equation 5. A vertex,  $v_i$ , in path  $P$ , contains a method call  $m_i$ . Then there exists a vertex  $v_j$  ( $j > i$ ), which corresponds to a return call for  $m_i$ .  $MA(m_i)$  is the set of methods activated by  $m_i$  before its return. The methods activated set (MA) is defined as:

$$MA(m_i) = \{m_k \mid i < k < j, v_k \text{ in } P\} \quad (7)$$

Then the class response set  $CR(c_n)$  for all methods  $m$  in class  $c_n$  is defined as follows:

$$CR(c_n) = \bigcup_{\forall m \text{ in } CT(c_n)} MA(m) \quad (8)$$

The  $c_n$  represents the class in which the method,  $m$ , resides ( $m$  is used to index the MA sets). The  $MA(m)$  may change each time  $m$  is invoked due to conditional statements within the method.

The set of all methods for a class,  $c_n$ , can be defined as follows:

$$M_{CT}(c_n) = \{m \mid m \text{ in method tuple of } CT(c_n)\} \quad (9)$$

The following equation yields the RFC for a class:

$$RFC_{COMDAG}(c_n) = |CR(c_n) \cup M_{CT}(c_n)| \quad (10)$$

The RFC can also be calculated for a component. This can be done by treating all methods in the component as if they belonged to the same class. Thus when adding methods to a component's response set  $CR(CC)$ , all methods in the CR sets belonging to the classes of the component are added. The  $CR(CC)$  is defined as follows:

$$CR(CC) = \left\{ \bigcup_{\forall c_i \in CC} CR(c_i) \right\} \quad (11)$$

The same applies to the  $M_{CT}$  resulting in a set that includes methods from all of a component's classes. The  $M_{CT}(CC)$  is defined as follows:

$$M_{CT}(CC) = \left\{ \bigcup_{\forall c_i \in CC} M_{CT}(c_i) \right\} \quad (12)$$

The component RFC is the cardinality of the union of the  $CR(CC)$  and  $M_{CT}(CC)$  sets. The RFC is defined as follows:

$$RFC_{COMDAG}(CC) = |CR(CC) \cup M_{CT}(CC)| \quad (13)$$

The RFC for designs is an indirect, absolute measure and can be used to compare both classes (Equation 10) and components (Equation 13) in a system.

### 3.4.2 The OMMIC Metric in Designs

The OMMIC measures the coupling for a class by counting method calls to other classes that are not in its inheritance hierarchy. The OMMIC for designs is measured by traversing paths in the COMDAG. The paths are traversed with a frequency indicated in the operational profile to simulate a workload. The set of all paths can be

defined as  $P = P_1 \dots P_y$ . Let  $P_t = \{v_1 \dots v_n\}$ , where  $v_i$  is a vertex defined by Equation 5. Vertices  $\langle v_i, v_{i+1} \rangle$ ,  $i = 1 \dots n - 1$  define the edges in the COMDAG. The vertex,  $v_i = \langle o_i, m_i, \langle ARG_S \rangle, CT(c) \rangle$  in path  $P_t$  contains an object  $o_i$ , a method  $m_i$ , the method arguments,  $ARG_S$ , and a class tuple  $CT(c)$ . The vertex,  $v_{i+1} = \langle o_{i+1}, m_{i+1}, \langle ARG_S \rangle, CT(d) \rangle$  in path  $P_t$  is defined similarly. The  $CT$  contains all class information including parent class information. We can define the set of all classes in vertex  $v_i$  as follows:

$$classes(v_i) = \{e \mid e \text{ occurs in inheritance structure of } c, \text{ where } v_i \text{ contains } c\} \quad (14)$$

The OMMIC metric relies on distinguishing if a method call between vertices  $v_i$  and  $v_{i+1}$  is invoked by a class within the same inheritance hierarchy. We define the Boolean function *other* to make this determination:

$$other(v_i, v_{i+1}) = \begin{cases} 1, & d \notin classes(v_i) \\ 0, & otherwise \end{cases} \quad (15)$$

The OMMIC for a class  $c$  is summed over all paths in an operational profile. Different paths may have differing method calls, thus interaction between classes may differ with each path. Summing over all paths includes all interactions in a class. The frequency,  $f_t$ , is associated with a path  $P_t$ . The frequency indicates how often that path is traversed. We use the frequency,  $f_t$ , as a weight to give classes with higher usage a higher coupling value. The OMMIC for designs is defined as follows:

$$OMMIC_{COMDAG}(c) = \sum_{t=1}^y (f_t * \sum_{i=1}^n other(v_i, v_{i+1})), \quad (16) \\ \forall v_i \text{ containing } c$$

This means that the OMMIC of  $c$  is increased every time a method call is made to a class outside of the inheritance hierarchy of  $c$ .

The OMMIC can be calculated for a component  $CC$  with classes  $c_1 \dots c_n$  by modifying Equation 15 to include all classes in the same component. The  $classes_c$  set for components can be defined as follows:

$$classes_c(v_i) = \bigcup_{\forall c_i \in CC} \{e \mid e \text{ occurs in inheritance structure of } c_i, \text{ where } v_i \text{ contains } c_i\} \quad (17)$$

The  $classes_c(v_i)$  creates a set of all classes defined in a component, where the class in  $v_i$  is a member of the component.

### 3.4.3 The ICP Metric in Designs

The ICP measures the coupling for a class by counting the number of calls to other classes in a design. To define the ICP, we use an operational profile and the same definitions for paths, vertices, and CTs used in defining the OMMIC. The ICP metric relies on distinguishing if classes associated with vertices  $v_i$  and  $v_{i+1}$  are the same or different. If the classes are different then the method call from  $v_i$  to  $v_{i+1}$  is external. External calls from a class indicate interaction with other classes, which is how we define coupling. Thus, we define the Boolean function *external*:

$$external(v_i, v_{i+1}) = \begin{cases} 1, & d \neq c \\ 0, & otherwise \end{cases}$$

The ICP for a class  $c$  can be calculated by summing over all paths in an operational profile. The ICP uses the number of method parameters,  $ARGS$ , in  $v_i$  as a weight. The frequency,  $f$ , is used as a multiplier to give classes with higher usage a higher coupling value. The ICP is defined as follows:

$$ICP_{COMDAG}(c) = \sum_{t=1}^y (f_t * \sum_{i=1}^n external(v_i, v_{i+1}) * |ARGS|), \quad \forall v_i \text{ containing } c \quad (18)$$

This means that the ICP is increased for class  $c$  every time a method call is made to a class outside of  $c$ .

The ICP can be calculated for a component  $CC$  with classes  $c_1 \dots c_n$  by modifying Equation 18 to exclude all classes in the same component. The  $external_c$  function for components is defined as follows:

$$external_c(v_i, v_{i+1}) = \begin{cases} 1, d \notin CC, \text{ where } c \in CC \\ 0, \text{ otherwise} \end{cases} \quad (19)$$

The ICP design metric is indirect and uses attributes that are measured on an absolute scale.

### 3.4.4 The ICH Metric in Designs

The ICH measures the cohesion in a class by counting the number of method calls internal to a class. To define the ICH, we use an operational profile and the same definitions for paths, vertices, and CTs used in defining the OMMIC. The ICH metric relies on distinguishing if classes associated with vertices  $v_i$  and  $v_{i+1}$  are the same or different. If the classes are the same then the method call from  $v_i$  to  $v_{i+1}$  is internal. Internal calls in a class indicate interaction within the class, which is our definition of cohesion. Thus, we define the Boolean function  $internal$ :

$$internal(v_i, v_{i+1}) = \begin{cases} 1, d = c \\ 0, \text{ otherwise} \end{cases} \quad (20)$$

The ICH for a class  $c$  can be calculated by summing over all paths in an operational profile. The ICH uses the number of method parameters,  $ARGS$ , in  $v_i$  as a weight. The frequency,  $f_t$ , is used as a multiplier to give classes with higher usage a higher cohesion value. The ICH can be declared as follows:

$$ICH_{COMDAG}(c) = \sum_{\forall P} (f_t * \sum_{i=1}^n internal(v_i, v_{i+1}) * |ARGS|), \quad \forall v_i \text{ containing } c \quad (21)$$

This means that the ICH is increased for class  $c$  every time an internal method is called. The ICH can be calculated for a component  $CC$  with classes  $c_1 \dots c_n$  by modifying the Equation 20 to include all classes in the same component. The  $internal_c$  function for components is defined as follows:

$$internal_c(v_i, v_{i+1}) = \begin{cases} 1, d \in CC, \text{ where } c \in CC \\ 0, \text{ otherwise} \end{cases} \quad (22)$$

The ICH design calculation is indirect and uses attributes that are measured using an absolute scale.

## 3.5 Component Evaluation

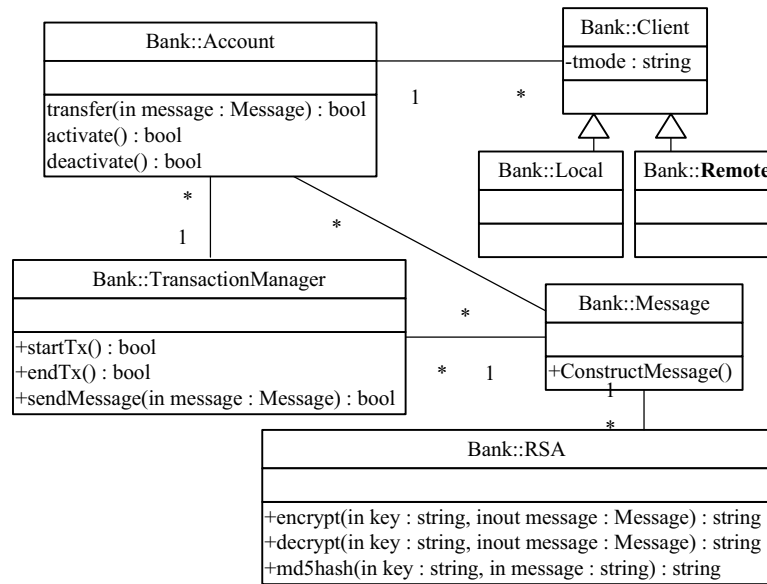
Metric collection produces values of the RFC, OMMIC, and ICP coupling metrics, as well as for the ICH cohesion metrics for every candidate component. There are various ways in which these values can be used by a designer to either analyze candidate components for strengths and weaknesses or to select among component choices.

1. The designer considers the actual values for a design component and determines whether individual values are satisfactory or not. If not, the designer can contemplate redesign to either increase, or decrease specific values. This is the most subjective way of using the metrics.
2. The designer evaluates the metrics against previously agreed upon thresholds for each cohesion and coupling metric. When component choices are to be made, only components that fall within the thresholds are further considered. This states objective requirements for cohesion and coupling. On the other hand, it may take a certain amount of historical data to determine such thresholds. If only one candidate component is evaluated and does not meet all thresholds, the designer must redesign to improve inadequate values of these metrics.
3. Given that there are three coupling and one cohesion metric, there are four variables with which to make a decision. A candidate component may have high coupling and high cohesion, while another may have low coupling and cohesion. Each option is thus good in one area and less desirable in another. How should one choose between the two? In this case, preferences between cohesion and coupling have to be defined. It is possible to use Multicriteria Decision Making Techniques [4] on the two groups of measures. Either lexicographic ranking or the more sophisticated Analytic Hierarchy Process (AHP) [12] can be used.

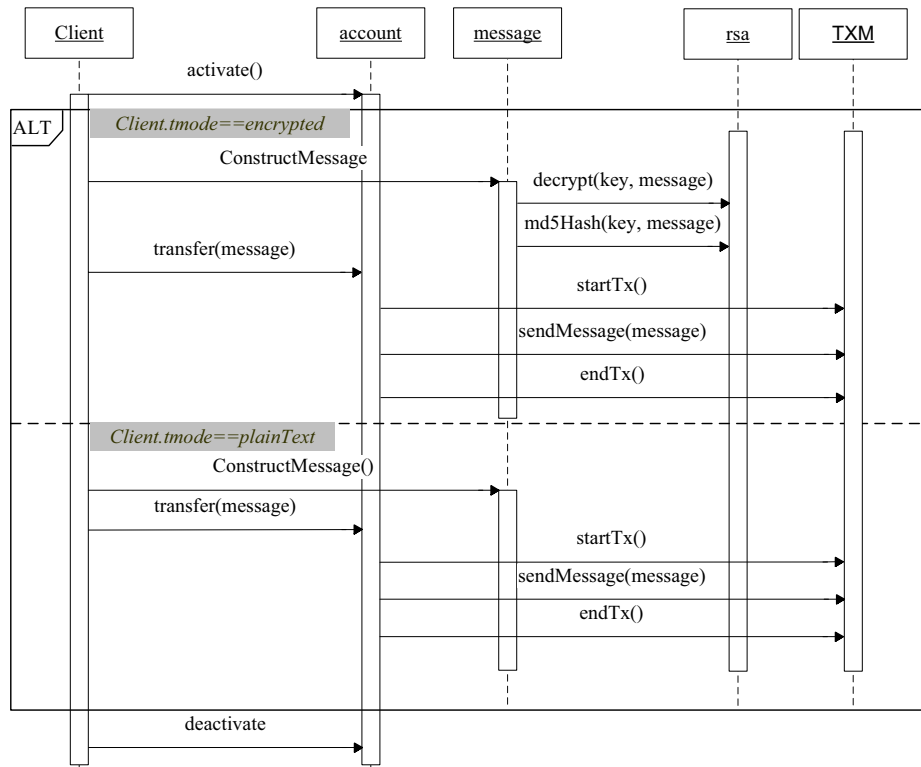
Whether a designer uses simple thresholds or more involved decision making methods, is often up to the specific situation, how many choices exist and how similar or different the collected measures are. It is not possible to recommend one of these methods over another.

## 4 Example

This example, adapted from [3], represents a design for a banking system. Figure 2 shows the Class Diagram and Figure 3 shows the Sequence Diagram. The diagrams outline the structural and behavioral aspects of a simplified banking system. The system contains two types of clients, a local



**Figure 2. Banking System Class Diagram.**



**Figure 3. Banking System Sequence Diagram.**

client and a remote client. The clients interact with the system by sending messages. If the client is remote then the

message needs to be encrypted and signed using an RSA encryption algorithm. Once the message is encrypted (if

remote), it is transferred to the bank account using a transaction manager. The transaction manager can begin and end a transaction. A transaction such as a transfer is not finalized until the transaction manager indicates an end to the transaction.

The designer of the system wants to componentize the system for reuse purposes. The Class Diagram contains one generalization where the *Remote* and *Local* classes inherit from the *Client* class. The *RSA* class provides encryption (decrypt) and signing (md5hash). The *Account* class contains the information about a banking client's account. The *TransactionManager* class starts and stops transactions to provide protection against system failure during a transaction. If the transaction is not complete and the system fails, the transaction manager backs off the transaction. In this example, remote transactions occur 4 times as often as local transactions. Two scenarios are represented as two paths in the Sequence Diagram. One scenario shows a client interacting with a bank account locally and the other shows a client interacting remotely.

#### 4.1 Build An Integrated Model

The first step of the approach is to build an integrated model by combining the Class and Sequence Diagrams into a COMDAG. Each class from Figure 2 is mapped into a CT according to the mappings outlined in section 3. The lower right side of Figure 4 contains the CT information. The Sequence Diagram in Figure 3 is traversed and mapped into the OMDAG according to the procedure in section 3. The information is combined into the COMDAG shown in the left side of Figure 4.

#### 4.2 Define Components

The next step in the approach is to decide on candidate components. In this example the designer wants to see the difference between combining: (1) the messages with the encryption utility into component  $COMP_1$  or (2) combine the messages with transaction management into component  $COMP_2$ . The candidate component set for  $COMP_1$  is the set  $CC_1 = \{Message, RSA\}$ . The candidate component set for  $COMP_2$  is the set  $CC_2 = \{Message, TransactionManager\}$ . By consulting the COMDAG we find that the first component vertex set consists of the following vertices:  $CV_1 = \{v_4, v_5, v_6, v_7, v_8\}$ . The second component vertex set consists of the following vertices:  $CV_2 = \{v_4, v_6, v_{13}, v_{15}, v_{21}, v_{23}, v_{25}\}$ .

#### 4.3 The Operational Profile

The next step in the approach is to define an operational profile, based on the expected system operation. The

COMDAG in Figure 4 contains two paths. According to the example description, remote transactions are 4 times as common as local transactions. This means the first path (vertices 3-16) of the COMDAG is executed 4 times more often. This results in the following operational profile for the banking application:

#### 4.4 Coupling and Cohesion Metrics

We now evaluate the two potential components with respect to coupling and cohesion by applying our coupling and cohesion metrics to each of the candidate components. We only give detailed steps for the RFC metric due to the length of the computations.

##### 4.4.1 The RFC Coupling Metric:

We apply Equation 7 to calculate the RFC for  $COMP_1$ . The  $COMP_1$  class set is  $CC_1 = \{Message, RSA\}$ . The  $MCT(CC_1)$  set is calculated by placing all methods from the  $CC_1$  into a set. This results in the following set:

$$MCT(CC_1) = \{\text{ConstructMessage, encrypt, decrypt, md5hash}\}$$

Next, we execute the operational profile for each path  $P_1$  and  $P_2$  by visiting each vertex in the path. Each vertex contains a method,  $m_i$ . Method  $m_i$  belongs to the class,  $c_n$  in vertex  $v_{i+1}$ . Notice that in the COMDAG, a method is always a member of the class contained in the next node. Thus if  $v_1$  and  $v_2$  are two vertices in the COMDAG, then a method  $m_1$  used in  $v_1$  belongs to the class in  $v_2$ . The  $MA(m_i)$  is the set of methods activated by  $m_i$  before its return. Table 2 illustrates the stepwise execution for calculating the RFC, where column 1 contains  $P_t$ , the current path, column 2 contains  $v_i$ , the currently visited vertex in a path, column 3 contains  $c_n$ , the class associated with  $m_i$ , column 4 contains  $m_i$ , the method call associated with the vertex  $v_i$ , and column 5 contains  $MA(m_i)$ , the set of activated methods. The table contains only non-empty  $MA(m_i)$  sets in Path  $P_1$ . Path  $P_2$  is not shown since it does not contain any non-empty  $MA(m_i)$  sets that belong to  $COMP_1$ . Note that the only message activations not immediately followed by a return are in  $v_3$  (ConstructMessage), with a return in  $v_8$ , and in  $v_9$  (transfer) with a return in the  $v_{16}$ . This results in the following non-empty MA sets:

$$\begin{aligned} MA(\text{ConstructMessage}) &= \{\text{decrypt, md5hash}\} \\ MA(\text{Transfer}) &= \{\text{startTx, sendMessage, endTx}\} \end{aligned}$$

Next we calculate the  $CR(c_n)$  for each  $c_n$ . This is the union of the  $MA$  sets associated with methods of class  $c_n$ . For  $COMP_1$  the  $CR(Message)$  is calculated by taking the union of all  $MA$  sets for each method in the *Message* class. Since there is only one method, *ConstructMessage*, the calculation is trivial. The  $CR$  for each class in  $COMP_1$  is as follows:

$$\begin{aligned} CR(Message) &= \{\text{decrypt, md5hash}\} \\ CR(RSA) &= \text{empty} \end{aligned}$$

The response set  $CR(COMP_1)$  is calculated by taking the union of the  $CR(c_n)$  sets, where  $c_n$  belongs to  $CC_1$ . The following set results:

$$CR(COMP_1) = \{\text{decrypt, md5hash}\}$$

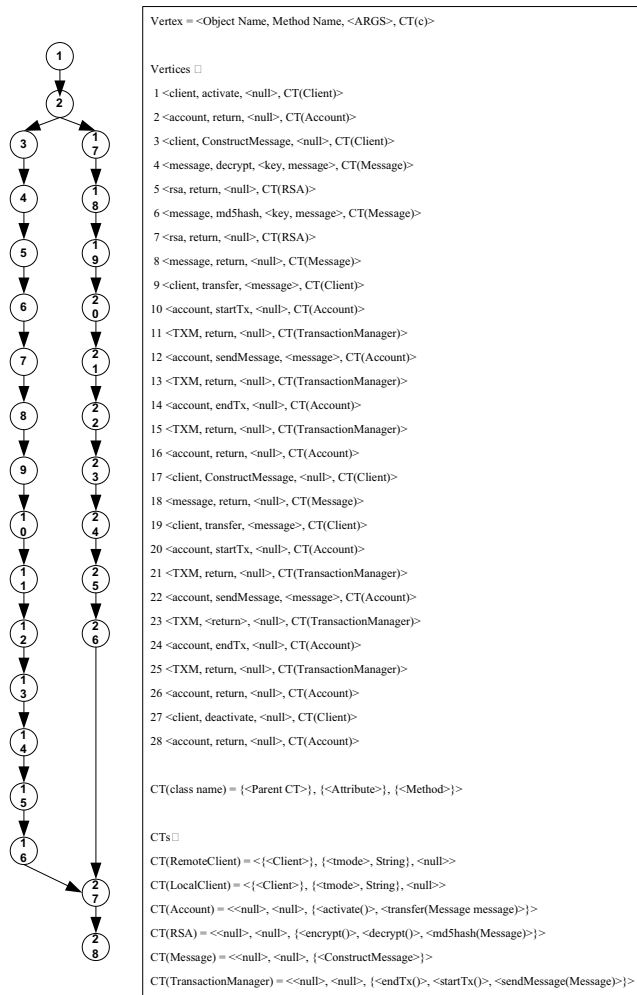


Figure 4. CT + OMDAG = COMDAG.

Table 1. Operational Profile for Banking Operation

Path Definition	Frequency
$P_1 = v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{27}, v_{28}$	4
$P_2 = v_1, v_2, v_{17}, v_{18}, v_{19}, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{27}, v_{28}$	1

The union of the  $M_{CT}(COMP_1)$  set and the  $CR(COMP_1)$  set results in the following set:

$$M_{CT}(COMP_1) \cup CR(COMP_1) = \{\text{ConstructMessage, encrypt, decrypt, md5hash}\}$$

The cardinality of this set is the RFC for component  $COMP_1$ , which is equal to 4.

Component  $COMP_2$  is calculated in the same manner. The  $COMP_2$  class set is  $CC_2 = \{\text{Message, TransactionManager}\}$ . The  $M_{CT}(CC_2)$  set is calculated by placing all methods from  $CC_2$  into a set. This results in the following set:

$$M_{CT}(CC_2) = \{\text{startTx, endTx, SendMessage, Con-}$$

structMessage }

The response set  $CR$  for component  $COMP_2$  is the union of the  $CR(\text{Message})$  set and  $CR(\text{TransactionManager})$  set. The  $CR(\text{Message})$  set has already been calculated in  $COMP_1$ . The  $CR(\text{TransactionManger})$  is empty since there are no nested method-calls made from within the class in either  $P_1$  or  $P_2$ . The  $CR$  for each class in  $COMP_2$  is as follows:

$$CR(\text{Message}) = \{\text{decrypt, md5hash}\}$$

$$CR(\text{TransactionManager}) = \text{empty}$$

The union of the  $M_{CT}$  and  $CR$  sets is

$$M_{CT}(COMP_2) \cup CR(COMP_2) = \{\text{startTx, endTx, SendMessage, ConstructMessage, decrypt, md5hash}\}$$

**Table 2. RFC calculations**

$P_t$	$v_i$	$c_n$	$m_i$	$MA(m_i)$
$P_1$	$v_1$	Account	activate	
$P_1$	$v_2$		return	
$P_1$	$v_3$	Message	ConstructMessage	
$P_1$	$v_4$	RSA	decrypt	
$P_1$	$v_5$		return	MA(ConstructMessage) = decrypt
$P_1$	$v_6$	RSA	md5Hash	MA(ConstructMessage) = decrypt
$P_1$	$v_7$		return	MA(ConstructMessage) = decrypt md5Hash
$P_1$	$v_8$		return	MA(ConstructMessage) = decrypt md5Hash
$P_1$	$v_9$	account	transfer	MA(ConstructMessage) = decrypt md5Hash,
$P_1$	$v_{10}$	TXM	startTX	MA(ConstructMessage) = decrypt md5Hash
$P_1$	$v_{11}$		return	MA(ConstructMessage) = decrypt md5Hash, MA(transfer) = startTx
$P_1$	$v_{12}$	TXM	sendMessage	MA(ConstructMessage) = decrypt md5Hash, MA(transfer) = startTx
$P_1$	$v_{13}$		return	MA(ConstructMessage) = decrypt md5Hash, MA(transfer) = startTx, sendMessage
$P_1$	$v_{14}$	TXM	endTx	MA(ConstructMessage) = decrypt md5Hash, MA(transfer) = startTx, sendMessage
$P_1$	$v_{15}$		return	MA(ConstructMessage) = decrypt md5Hash, MA(transfer) = startTx, sendMessage, endTx
$P_1$	$v_{16}$		return	MA(ConstructMessage) = decrypt md5Hash, MA(transfer) = startTx, sendMessage, endTx

The cardinality of this set (the RFC) for component  $COMP_2$  is 6. We can see that for  $COMP_2$  the  $CR$  set was not a subset of the  $M_{CT}$  set, thus it added to the number of members in the set, resulting in a higher RFC value.

The remaining metrics are calculated and can be found in Table 3.

#### 4.5 Evaluation

The last step is to evaluate which component has better coupling and cohesion measures. In this case simple inspection of the values for cohesion and coupling metrics was all that was needed, since one component choice outperformed the other for all metrics, making the use of AHP unnecessary. According to Table 3  $COMP_1$  is the clear winner. Component 1 has a RFC value that is half that of component 2. The primary difference in the OMMIC, ICP, ICH values for each component can be attributed to calls being external in component 2 and internal in component 1.

Metric	Component 1	Component 2
RFC	4	6
OMMIC	0	8
ICP	0	16
ICH	16	0

**Table 3. Component Metric Summary**

## 5 From Design Metrics to Maintainability

Our approach rests on the assumption that there is a correlation between cohesion and coupling metrics during design, and maintainability and quality of the resulting implementation. Since it has been established elsewhere that certain code coupling and cohesion metrics are correlated with quality and maintainability, one needs to show that our design level coupling and cohesion metrics are correlated with their code counterparts if we want to use them to determine future maintainability and quality at design time. To this end we designed an empirical study and asked the question: is there a correlation between design metrics and code metrics? We designed a study with one factor (the target of metric collection) and two treatments (design or code). The study analyzes a design and its corresponding implementation for the purpose of assessing their metrics with respect to their values from the perspective of a researcher.

The design we used consists of an annotated UML Class Diagram and Sequence Diagrams that describes a software package that can Gouraud Shade Polygons. The Class Diagram contains 15 classes and the Sequence Diagram describes their interaction. Annotations included potential inputs, which describe the format of a polygon. The output of the design is a 300x300 ppm image containing a shaded polygon.

The experiment was conducted in a classroom environment. The subjects were 10 students in a Senior Software Engineering course at Washington State University. The UML design was created based on a real world problem found in computer graphics. The project was part of

a graded test in their Senior Software Engineering Class. They were asked to implement the design using Java. They were not given a time constraint. The dependent variables were the RFC, OMMIC, ICP design and code metrics. All students created working implementations of the design. We collected data from each design and coded class and found that the Pearson Product Moment Correlation (PPMC) was greater than .95 in every case. All students created a central driver class based on the design. Figure 4 shows the results of our findings for the driver class.

Design Metrics				
	RFC	OMMIC	ICP	
	10	14	14	
Code Metrics				
	RFC	OMMIC	ICP	PPMC
student1	24	33	36	0.970725
student2	24	35	38	0.979076
student3	24	35	39	0.966282
student4	25	35	38	0.975417
student5	23	33	36	0.975417
student6	23	33	36	0.975417
student7	22	33	36	0.979076
student8	23	33	36	0.975417
student9	25	35	38	0.975417
student10	24	35	39	0.998461

**Table 4. Empirical Study Data**

We concluded that there is a positive correlation between code and design metrics. We noticed that there is a constant offset between the data sets. We propose that designs have less detail, specifically method calls, compared to the actual code. One of the threats to validity is the completeness of the design. Our design was very detailed and complete. In practice, designs may be incomplete or very simplistic. In addition, the size of the implemented programs were only around 1000 lines of code. Industrial size software is obviously much larger. We plan to address these concerns in future studies.

## 6 Conclusion

We have adapted four different coupling and cohesion metrics to help in defining component boundaries during design. Because they are derived from code metrics which show a significant correlation with quality, there is a reasonable expectation that the design level cohesion and coupling metrics will as well. Besides helping with defining component boundaries during design, these metrics can also be used for design evaluation, by setting thresholds for acceptable cohesion and coupling levels. It would be worthwhile to investigate empirically what reasonable levels for coupling and cohesion are for these measures.

## References

- [1] E. Arisholm, L. Briand, A. Foyen, "Dynamic Coupling Measurement for Object-Oriented Software, IEEE Transactions on Software Engineering, pp. 491-506, August 2004.
- [2] V. Basili, L. Briand, W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, pp. 751-761, October, 1996.
- [3] R. Binder, *Testing Object-Oriented Systems Models, Patterns, and Tools*, Object Technology Series, Addison Wesley, Reading, Massachusetts, 1999.
- [4] P. Bogetoft, Peter Pruzan; *Planning with Multiple Criteria*, North-Holland, 1991.
- [5] L. Briand, J. Wuest, Empirical Studies of Quality Models in Object-Oriented Systems, *Advances in Computers*, Academic Press, vol. 56, 2002.
- [6] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design", *Information and Technology*, Vol 35, No5, pp 232-240, 1991.
- [7] G. Heineman, W. Councill, *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley, Boston, MA, 2001.
- [8] Lee, Y. -S., Liang, B. -S., Wu, S. -F., and Wang, F. -J., "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow", *International Conference on Software Quality* pp. 81-90, 1995.
- [9] W. Li, and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, 23(2), pp 111-122, 1993.
- [10] J. Musa. *Software Reliability Engineering* McGraw-Hill, New York. 1999.
- [11] O. Pilskalns, A. Andrews, R. France, S. Ghosh. "Rigorous Testing by Merging Structural and Behavioral UML Representations", *UML Conference 2003*, pp. 234-248, 2003.
- [12] T. Saaty. *The Analytical Hierarchy Process*, McGraw-Hill, 1990.