I.J. Information Engineering and Electronic Business, **2016, 2, 1-13**
Published Online March 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2016.02.01

Design Migration from Procedural to Object Oriented Paradigm by Clustering Data Call Graph

Mohayeminul Islam, Tajkia Rahman Toma
Jantrik Technologies Limited, Dhaka 1213, Bangladesh
E-mail: mohayeminul.islam@jantrik.com, tajkia.toma@jantrik.com

Md. Selim
Department of Disaster Science and Management, University of Dhaka, Dhaka 1000, Bangladesh
E-mail: selim.iitdu@gmail.com

Alim Ul Gias, Shah Mostafa Khaled
Institute of Information Technology, University of Dhaka, Dhaka 1000, Bangladesh
E-mail: alim@du.ac.bd, khaled@du.ac.bd

Abstract—Management of legacy software and its code, generally written in procedural languages, is often costly and time-consuming. To help this management, a migration from procedural to object oriented paradigm could be a cost effective option. One approach for such migration can be based on the underlying dependency structure of the procedural source code. In this work, we propose a new heuristic algorithm that utilizes such structure for the design migration using agglomerative hierarchical clustering. The dependency structure that has been used involve functions, parameters and global data of the procedural code. Given a procedural code, the proposed approach produces candidate classes for an object oriented design. The proposed algorithm was tested against a database of procedural codes. The results obtained have been empirically validated using Jaccard similarity coefficient. It is observed that the proposed method yields 75.6% similarity with respect to the ground truth in the average case.

Index Terms—Design Migration, Agglomerative Hierarchical Clustering, Code Analysis, Software Maintenance, Reverse Engineering.

I. Introduction

Software maintenance and change management are two of the “never ending” continuous processes in software development life cycle [1]. These processes involve the activities necessary to modify or improve an existing software product and requesting, planning, implementing and evaluating changes to the software system. The cost of such continuous processes largely depend on the design of the software system. Poor design may hinder the maintenance process by adding complexity and requiring additional effort and cost to mitigate those complexity.

The maintenance of legacy software [2, 3] often becomes costly and difficult due to its development paradigm. For an example, procedural programming was highly popular in earlier decades of software development and thus many legacy software were developed using procedural languages. The inherent limitations of procedural languages to deal with complexity and manageability of large code base make it troublesome to maintain such software [4, 5]. Moreover, it is difficult to ensure quality attributes like reusability [6], maintainability [7] and modularity [8] by using procedural paradigm.

Object oriented programming languages, by their design, provide better control on the code with proper use of its basic properties: Polymorphism, Inheritance and Encapsulation. For these reasons, migration to an object oriented languages is sometimes expected. A manual migration process can be very expensive in terms of cost of development, resource and time [9]. For large software system developed in procedural programming language, an automatic migration to object oriented design can effectively reduce this cost. This could be achieved by reverse engineering which helps to analyze a system’s internal elements and its external behavior and creating a structural view of the system.

One of the major challenge in any object oriented design is to ensure Encapsulation. It is defined as the bundling of data and functions that operates on the data together in a single component [10]. Encapsulation provides modularity in an object oriented design that helps in better manageability. Thus, a reverse engineering process of migrating procedural scheme to object oriented one must cover the encapsulation property of classes. To be more precise, it should properly define the class membership for each of the data and functions in the procedural code [11].

To solve the problem, we propose a heuristic approach to migrate from procedural to object oriented design. Our work has three major contributions: first, we represent a procedural program as a special type of graph, Weighted Data Call Graph, and apply Agglomerative hierarchical clustering on the graph. Second, we define a new similarity coefficient to measure similarity between nodes of the graph. Third, we score each level of the hierarchy by measuring cohesiveness [12], coupling [13] and modularity. The clustering level that produces highest score is considered to be the class design for the input procedural program.

Our work has extended the traditional call graph and defined the concept of the Data Call Graph (DCG). A call graph is defined as a human readable graphical representation of a program [14, 15, 16, 17]. In a call graph, each of the functions in the program are represented by a node. If a function calls another function, there is an directed edge from the first function to the later. In case of DCG, it includes data nodes along with function nodes of call graph. A weight is set on the edge of the DCG depending on the relationship represented by the edge and thus a Weighted DCG (WDCG) is produced.

A clustering scheme should be applied to WDCG to generate clusters which are candidate classes in the object oriented design [18]. We have applied Agglomerative hierarchical clustering [19, 20] on WDCG to find the class design from procedural program. Agglomerative clustering requires calculation of similarity between nodes of the graph. There is a number of similarity measures to calculate similarity between nodes of a graph. However, none of them are applicable to weighted graphs. Therefore, we have defined a Weighted Distance Matrix (WDM) to measure similarity among nodes in a weighted graph.

In the migration process we create a hierarchy of clusters on the WDCG using Agglomerative clustering. Then we search a level in the hierarchy that gives maximum output to an objective function based on three attributes of an object oriented design: coupling, cohesion and modularity. That is, we followed the principle that a design should be as modular as possible, the members of a class should be highly co-related and any pair of classes should be as decoupled as possible. These quality attributes are widely accepted in the existing literature [18, 21, 22, 23]. The clusters generated in this way give clues to classes in the object oriented design.

We have performed an empirical evaluation of the design migration process by calculating similarity between output produced by our technique and manual object oriented de- sign. The manual designs are prepared and evaluated by a group of professional software engineers from three Software Development Companies (Jantrik Technologies Ltd. *www.jantrik.com*, Genweb2 www.genweb2.com, Therap (BD) Ltd. *www.therapservices.net*). Five C programs of variable size have been used in our evaluation and the similarity was measured using Jaccard similarity coefficient. It has been seen that in best case, the approach yielded 100% similarity and in average case the similarity turned out to be around 75.6%.

Rest of the paper is organized as follows: Section II dis- cusses the state of the art on design migration. The new terms that have been introduced in this work is presented in Section.

III. Section IV discusses the design migration methodology by clustering Weighted Data Call Graph. Verification and validation of the proposal is presented in Section V. Finally, the concluding remarks along with future research direction is provided in Section VI.

II. Related Works

Since the evaluation of object oriented concept, migration from procedural to object oriented paradigm has been ad- dressed as a major research problem. However, most of these works focus on code-level migration, rather than addressing the migration in design level. By code-level migration, we mean that a program is simply translated from one language to another, whereas we focus on migrating the design from one paradigm to another. This section highlights some of those major contributions in design migration.

Sneed et al. [24] introduced the concept of code to design migration. The work converts a COBOL program to an object oriented design document by capturing and documenting the sequence of operations executed in a COBOL system. The work identified different types of objects like user interface objects, information objects, file objects, view objects, etc. This work is only for COBOL and does not generalize for all procedural languages. They also developed a tool 'ObjectRedoc' using C and MS-Access running under MS-Windows and used it to help downsizing of mainframe legacy systems.

There has been work where researchers aim to decompose a software system to subsystems to understand its behavior [25]. Their work is based on software clustering technique that uses both static and dynamic information from the source code. Nevertheless, their work does not provide any guideline to produce classes for object oriented design. Works of similar nature, in the context of software architecture recovery and modularization, have been reviewed by Maqbool et al. [26]. Their work analyzed the clustering process of multiple clustering algorithms using multiple criteria and showed how arbitrary decisions taken by these algorithms effect the quality of the clusters.

Heroux et al. [27] presented the architectural comparison of commercial software and scientific research software. They found that commercial software are written for the purpose of generating revenue where the underlying algorithms and methodologies are mature. On the other hand the scientific research software in computational science and engineering disciplines are developed for new algorithms and computational capabilities which are less modular and largely unaware of standard industry concepts and practices.

The work in [9] used relational database for transforming object oriented design from procedural system. The migration was done in two phases. Firstly, object identification form procedural program and secondly, translation of the old one into new object oriented system. They implement three algorithms presented in [28, 29] for identifying objects in procedural software namely global based object identifier (gboi), type based object identifier (tboi), receiver based object identifier (rboi). These methods were applied to a 3000 line program written in C, and rboi performed better in identifying classes. Khan et al. [30] compared procedural programming with object oriented programming and proposed a migration approach from procedural to object oriented language. This approach bind object on the basis of hierarchical dependency of classes instead of clustering function together. This study focused on the structural difference between the procedural and object oriented programs and concluded that object oriented programming increases software reliability and modifiability by decoupling specification and implementation, and allows software code to be extensible, reusable and maintainable.

The method of finding object-like feature in code presented in [29] is based on global data or data type. They define class as a set of data, type and function. First method considers only the global variable and represent a program as a graph based on the use of variable in a routine. If a variable is used by a function, then they are connected by an edge. Each of the connected components in the generated graph are considered individual classes. This highly optimistic method is very likely to generate large classes with many responsibilities.

Van Deursen et al. [20] presented clustering with concept analysis for identifying objects in legacy code. This object identification approach uses clustering and concept analysis separately and result is merged together to overcome the shortfall of clustering technique. The method uses concept analysis to determine number of classes and uses Agglomerative clustering to identify classes. The proposed identification technique was applied to a real life legacy COBOL system to evaluate performance. However, their work focuses on code translation, rather than addressing the migration in design level.

A framework for migrating procedural code to object oriented platforms is presented in [21]. This framework generates Abstract Syntax Tree and then this tree is translated in to extensible markup language (XML). This XML is analyzed for identifying class based on the global data and variables. The work focus on minimizing coupling and maximizing cohesion. The method was applied at the IBM Center for Advanced Studies of IBM Toronto Lab for C to C++ migration. This framework can also be used to identify reusable components. The work in [31] recognized the maintenance and evolution of large software as difficult. To make such software understandable, they used directed graphs to represent software modules and their dependencies. This representation, although hides a lot of details, might be complex and therefore, partitioning closely related modules into clusters was considered as a suitable approach to comprehend the complex design. They presented a genetic algorithm based clustering technique to identify the modularity of a procedural system. They success- fully applied this approach in Mini-Tunis [32] software and validated by comparing with the original documented design. In 2011 Dineshkumar et al. [19] presented an empirical approach to migrate from structured program to object oriented design. They introduced a new technique for code to design migration, which creates Agglomerative cluster using Jaccard distance matrices. They initialized the relationship between variables and functions of the structured program using Jaccard distance measure that leads to grouping or clustering. They represented the result in UML and conducted a comprehensive industrial survey to evaluate the accuracy of their proposal. Their use of Jaccard distance as a measure of distance between clusters considers all types of relationships to be equally contributing to the distance between the clusters. In 2013 Siddik et al. [18] modeled structured to object oriented design migration as a optimal graph clustering problem. However, this model was realized to be computationally hard. The work presents eight heuristic algorithms based on Monte Carlo and Greedy approaches and formulated the κ-index for measuring the quality of a cluster. Clustering coefficient (Ψ) and characteristic path length (χ) were also used for assessing the quality. This approach only considers functions to be potential member of class ignoring variables and constants. They used high coupling, low cohesion and large number of objects as the objective.

The work of Siddik et al. [18] was extended by Selim et al. [33] and Siddik et al. [34]. Selim et al. [33] presented a genetic algorithm for finding optimal clustering of call graph with an objective to maximize the number of intra-cluster edge and minimize inter-cluster edge. This work outperforms the previous work [18] of Greedy and Monte Carlo by 40% and by 49.5% respectively. Siddik et al. [34] also extended their previous work [18] and proposed two variations of local search heuristic. Selim et al. proposed an extension over this work based on variable neighborhood search [35]. However, these works ignore variables and constants of the procedural code as decision parameters for determining class.

The review shows that only a few of the works directly proposed a method that focus on migrating procedural design to detailed object oriented design. Those approaches do not yet solve design migration problem. To overcome the limitations in the existing literature, we propose a new heuristic approach to migrate a procedural design to an object oriented design, which considers functions, global data and parameters of the procedural code as a basis for the intended reverse engineering.

III. Proposed Terminologies

In this section we propose the terminologies, notations and equations to support the design migration technique. We propose a new type of graph, Weighted Data Call Graph (WDCG), to represent a procedural program. To compute similarity between two nodes of WDCG, we introduce a new similarity measure called Weighted Distance Matrix (WDM). Moreover, we propose a data structure, Entity Map that incorporates certain weights along with Entity Set [19] to calculate the value of WDM.

These concepts are elaborated in the following subsections.

A. Weighted Data Call Graph

Call graph is a widely used representation for program analysis. A number of related works have used this representation for their design migration techniques [18], [34]. Although very useful, call graph limits the program representation only to functions or classes; however, the variables and the constants of a program are also important members of the design. We extend call graph to Data Call Graph (DCG) where variables and constants are also represented by nodes. We do not identify variable and constant separately and use the term data to mean any of them. In a DCG there are two types of nodes:

Function Node: Node to represent functions. We ignore the system functions and external library functions because they are out of the scope of design migration. Only the functions defined in the program to migrate are considered.

Data Node: Node to represent data. Like the Function Node, we exclude the system data and external library data and include only the data declared within the program to migrate. Data nodes include constants, variables, pointers and dynamically created variables.

There are four types of edges in DCG:

Self-Edge: Every element is connected to itself by a self-edge. Because self-edge exists for all elements, we do not show those in DCG.

Call Edge: When a function calls another function, there is a call edge from the first function to the second. Call edge is identical to the edges in call graph.

Read Edge: There is a read edge from function f to data d if and only if d is read in f. Reading may occur by reading a global variable or reading a variable passed as argument to the function.

Write Edge: There is a write edge from function f to data d if and only if d is written in f. Writing may occur by writing a global variable or writing a variable passed by a pointer or reference to the function.

All types of relationships between a pair of elements cannot be considered equal, therefore we put weight on the edges of DCG depending on the relationship of represented by the edge. We call the resultant graph a Weighted Data Call Graph (WDCG). The weights are not absolute but relative. That means, the value of the weight of different type of edges are not important, but the ratio among them are important. We have considered weight of self-edge to always be 1 and assigned weight to call edge, read edge and write edge from a predefined set of weights.

A sample DCG for a C program named *Pebble Dropping* (*bitbucket.org/mohayemin/designmigration/raw/default/ExperimentalData*) is presented in Figure 1. This example has been used throughout this paper to illustrate different aspects of our work. The program has 9 functions and 6 global variables. Therefore the DCG has 9 function nodes and 6 data nodes. Different edges representing function calls and read/write operations are also shown in Figure 1.

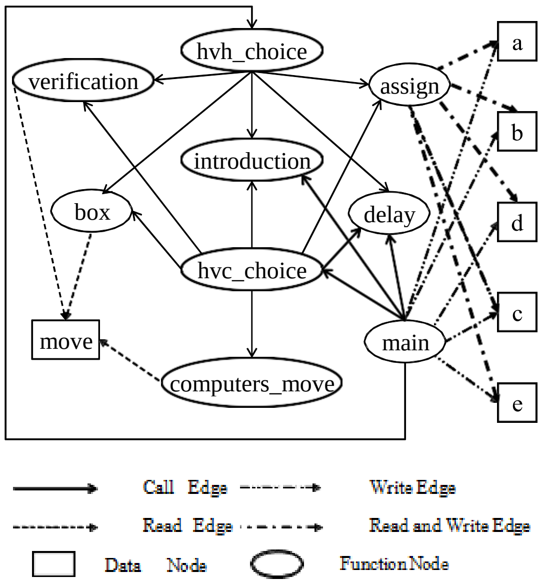

Fig.1. Data Call Graph of the *Pebble Dropping* Program Showing All Types of Nodes and Edges

B. Entity Map

The notion of Entity Map is similar to entity set [19] which is used to calculate similarity between two programming elements. The similarity is used in Agglomerative clustering technique. However, we cannot use entity set directly because it works only with unweighted relationship between programming elements. Therefore, we extend entity set and define entity map. Before discussing entity map, we formally define entity set as follows:

The entity set of a function f, (E_f) is defined as,

$$E_f = \{g : g \text{ is a function called by f}\} \cup \{d : d \text{ is a data used by f}\} \cup \{f\}$$

i.e., entity set of a function includes the functions it calls, data accessed by the function and the function itself.

The entity set of data d, (E_d) is defined as,

$$E_d = \{g : g \text{ is a function that use d}\} \cup \{f\}$$

i.e., entity set of a data includes the functions that access that data and the data itself.

Therefore for the DCG in Figure 1

$$E_{main} = \{main, delay, introduction, hvc\_choice, hvh\_choice, a, b, c, d, e\}$$

$$E_{move} = \{move, verification, box, computers\_move\}$$

While elements of entity set are programming elements (function or data), elements of entity map are ordered pairs where first entry is a programming element and the second entry is a number representing weight of the relationship.

Entity map of an element contains the element itself paired with self-weight (w_s). In addition, entity map of a function contains the set of pair of the function it calls and call weight (w_c), the set of pair of the data it writes and write weight (w_w), and the set of pair variables it reads and read weight (w_r). Entity map of a data contains the set of pair of the function which writes the variable and w_w, and the set of pair of function which reads the variable and w_r. Formally entity map of function f,

$$\varepsilon_f = \{(f, w_s)\} \cup \{(g, w_c): g\ is\ a\ function\ called\ by\ f\} \\ \cup \{(d_w, w_w): d_w\ is\ a\ data\ \ written\ by\ f\} \\ \cup \{(d_r, w_r): d_r\ is\ a\ data\ \ read\ by\ f\}$$

$$\varepsilon_d = \{(d, w_s)\} \\ \cup \{(g_w, w_w): g_w\ is\ a\ fuction\ that\ writes\ d\} \\ \cup \{(g_r, w_r): g_r\ is\ a\ function\ that\ reads\ d\}$$

Therefore in the DCG in Figure 1

$$\varepsilon_{main} = \{(main, w_s), (delay, w_c),\ (introduction, w_c), \\ (huv_{choice}, w_c), (huv_{choice}, w_c), \\ (a, w_r), (b, w_r), (c, w_r), (d, w_r), (e, w_r)\}$$

$$\varepsilon_{move} = \{(move, w_s), (verification\ w_r),\ (box, w_r), \\ (computers\_move, w_r)\}$$

Entity map of a programming element is actually a binary relation, where domain is the programming elements and co-domain is real number. The elements of entity map are called *weighted entry*. For entity map, we define the following terms:

Merged Entity Map: In an entity map weighted entries $(a, w_1), (a, w_2), \dots, (a, w_x)$ can be replaced by the single weighted entry $(a, \sum_1^x w_i)$. This operation is called merge and if we merge all possible merge-able weighted entries in Q and the result is entity map L, we call L is merge of Q. Merge of Q is denoted by $\mathrm{M(Q)}$.

Element Set: Element Set of an entity map is the set of the programming elements in the entity map. Element Set of an entity map Q is denoted by $\mathrm{E(Q)}$.

Weight Sum: Weight Sum of a entity map is the sum of weights of the entity map. Weight sum of an entity map Q is denoted by $\mathrm{S(Q)}$.

Weighted Intersection ($\cap_w$)*:* Weighted Intersection is an operation of two or more entity maps where the result of the operation is an entity map R for which $(a, w) \in R$ if and only if $a \in I$ and $(a, w) \in U$ where I is the intersection of element set of the operands and U is union of all operands. Weighted intersection of two entity maps Q_1 and Q_2 is denoted by $Q_1 \cap_w Q_2$.

Weighted Union ($\cup_w$)*:* Weighted Union is an operation of two or more entity maps where the result of the operation is an entity map R for which $(a, w) \in R$ if and only if $(a, w) \in U$ where U is union of all operands. Weighted union of two entity maps $Q_1$and Q_2 is denoted by $Q_1 \cup_w Q_2$.

C. Weighted Distance Matrix

Before defining the concept of Weighted Distance Matrix (WDM), we need to define Weighted Similarity Coefficient (WSC). WSC is a variant of Jaccard Similarity Coefficient which is a widely used similarity measure in hierarchical graph cluster analysis. We cannot directly use this coefficient because of the weights on the edges of WDCG. Therefore, we define WSC whose semantics is similar to that of Jaccard's coefficient.

Jaccard Similarity between two finite sets A and B is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

We define Weighted Similarity Coefficient (WSC) between two entity maps A and B as follows:

$$J_w(A, B) = \frac{S(A \cap_w B)}{S(A \cup_w B)}$$

WSC hold the following properties:

Range: The value of $J_w(A, B)$ is between 0 and 1. Inclusive the value is 0 when $A \cap_w B$ is empty and the value is 1 when $A \cap_w B = A \cup_w B$. For all other cases, the value is in between 0 and 1.

Symmetry: WSC is symmetric, i.e., for any two entity map A and B,

$$J_w(A, B) = J_w(B, A)$$

We can use the notion of WSC to calculate the weighted distance between two entity maps A and B. It can be formally defined as,

$$D_w(A, B) = 1 - \frac{S(A \cap_w B)}{S(A \cup_w B)} \tag{1}$$

Weighted distance holds the properties of WSC. The conditions range property is reversed, i.e., 1 when $A \cap_w B$ is empty and 0 when $A \cap_w B = A \cup_w B$.

A Weighted Distance Matrix (WDM) X can be calculated for a set of entity maps $\mathrm{M_1}, \mathrm{M_2}, \dots, \mathrm{M_n}$, where for each entry X(i, j) represents the weighted distance between $\mathrm{M_i}$ and $\mathrm{M_j}$. Formally,

$$X(i, j) = D_w(M_i, M_j), \qquad \forall i, j$$

WDM holds the following properties:

Squareness: Each column and each row of a WDM represent one and only one entity map. Therefore number of rows and number of columns are equal to number of entity maps.

Symmetry: In a WDM M, $M(i, j) = D_w(M_i, M_j)$ and $M(j, i) = D_w(M_j, M_i)$. According to Symmetric property of weighted distance, right side of the two equalities are equal. Therefore $M(i,j) = M(j,i)$ which means M is symmetric.

Hollow Matrix: $M(i,i)$ represents the distance from an element to itself which is zero. Therefore, $M(i,i)$ is 0 for all distance matrix M making it a Hollow matrix [36].

IV. Design Migration

In this section we present the proposed design migration method. The migration process is done in three main steps. In the first step we generate a WDCG from a procedural program. The next builds a hierarchical cluster tree of the WDCG using Agglomerative clustering technique. In the last step, we define an objective function and find a level in the hierarchy that maximizes the objective function. This optimal clustering is the object oriented design where each cluster of this clustering is a candidate class. The steps are illustrated in Figure 2 and are elaborated in the subsequent sections.

A. Weighted Data Call Graph Generation

This step takes a procedural program and produces a Weighted Data Call Graph for the program.

We have discussed about the elements of WDCG in Section III.A. Common graph representations like adjacency matrix, adjacency list, incidence matrix and incidence list do not support different type of nodes. Therefore, we have defined a technique to represent WDCG. The technique is a context free grammar [37] with the following rules:

Keyword: There are five keywords: ?function, ?data, calls, reads and writes. The first two keywords are declaration keywords and others are relation keywords.

Identifier: Identifiers can be words except the keywords and may contain any characters except whitespace characters. Also, identifiers cannot start with a question mark (?).

Statement: The statements are separated with semicolon (;). There are two types of statement:

- **Declaration Statement:** Declaration statements are used to state which elements are present in the WDCG. One statement declares exactly one element. Declaration Statement starts with one of the declaration keywords and then an identifier. If the declaration keyword is? unction, then the identifier is name of a function and if it is ?data the identifier is name of a data.
- **Relation Statement:** A relational statement indicates relationship between two elements. Relation statement starts with an identifier then a relation keyword then another identifier. The identifier are names of elements declared before the current statement. A calls relation statement has two function elements linked with a calls keyword that indicated the left function calls the right function. A reads/writes relation statement starts with a function element and then the keyword reads/writes and then a data element. This statement indicates that the function reads or writes the data.

The WDCG notation of the *Pebble Dropping* Program is illustrated in Listing 1. In the listing, initially all the functions of the code are declared. The next section includes all the data declarations. Finally the relation statements are provided.

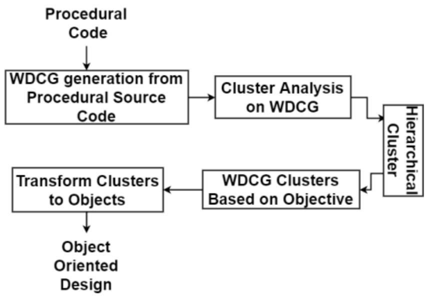
Procedural Code
WDCG generation from Procedural Source Code
Cluster Analysis on WDCG
Hierarchical Cluster
WDCG Clusters Based on Objective
Transform Clusters to Objects
Object Oriented Design

Fig.2. Overall Process for Proposed Design Migration.

Listing 1. WDCG notation for Pebble Dropping program

```
?function main;
?function assign;
?function hvc_choice;
?function hvh_choice;
?function verification;
?function introduction;
?function box;
?function computers_move;
?function delay;

?data a;
?data b;
?data c;
?data d;
?data e;
?data move;

main writes a;
main writes b;
main writes c;
main writes d;
main writes e;
main writes e;
main calls hvc_choice;
main calls hvh_choice;
main calls delay;
main calls introduction;

assign writes a;
assign writes b;
assign writes c;
assign writes d;
assign writes e;
assign reads a;
assign reads b;
assign reads c;
assign reads d;
assign reads e;

hvc_choice calls assign;
hvc_choice calls verification;
hvc_choice calls delay;
hvc_choice calls introduction;
hvc_choice calls computers_move;
hvc_choice calls box;

hvh_choice calls assign;
hvh_choice calls verification;
hvh_choice calls delay;
hvh_choice calls introduction;
hvh_choice calls box;

verification reads move;
box reads move;
computers_move reads move;
```

B. Cluster Analysis

Although significant works have been done on non-hierarchical clustering techniques [38, 39, 40], hierarchical techniques [41, 42] are more appropriate in our case as it gives the opportunity to choose the clustering that gives the best result of a predefined function. Moreover, non-hierarchical techniques usually require information like number of clusters, cluster size, etc. [43] which are not available in our context. We have implemented a classical Agglomerative hierarchical clustering [19, 20] technique for generating clusters from WDCG. For the distance measure, we have used weighted distance measure described in Section III.C.

Algorithm 1 Build Cluster Hierarchy

```
Input: Γ  the set of programming elements
 1: function BuildClusterHierarchy(Γ)
 2:    currentClusters ← Γ
 3:    hierarchy ← {currentClusters }
 4:    while size of currentClusters > 1 do
 5:       $(c_i, c_j) \leftarrow$
          $ClosestClusterPair(ClosestClusters)$
 6:       $currentClusters \leftarrow$
          $(currentClusters\ \{c_i, c_j\}) \cup \{\{c_i, c_j\}\}$
 7:       $hierarchy \leftarrow$
          $hierarchy \cup \quad \{currentClusters\}$
 8:    end while
 9:    return $hierarchy$
10:    end function
11: function ClosestClusterPair(clusters)
12:    $minD \leftarrow \infty$
13:    $minC_i \leftarrow NULL$
14:    $minC_j \leftarrow NULL$
15:    For each $c_i$ $in\ clusters$ do
16:       For each $c_j$ $in\ clusters$ do
17:          $d \leftarrow D_w(c_i, c_j)$          ► See Equation 1
18:          If $d < minD$ then
19:             $minD \leftarrow d$
20:             $minC_i \leftarrow c\_i$
21:             $minC_j \leftarrow c_j$
22:          end if
23:       End for
24:    End for
25:    $mainPair \leftarrow \{minC_i, minC_j\}$
26:    Return $mainPair$
27: End function
```

In our implementation we find one level of hierarchy at a time, starting from each nodes in different clusters to all nodes in one cluster. If we have n nodes, there will be n hierarchies. In each iteration, we find the closest pair of clusters, then create the new level by removing the pair from current level and adding the pair after merging. The iteration stops when the current level has only one cluster, i.e., all nodes are in one single cluster. These steps are presented in Algorithm 1.

The function $BuildClusterHierarchy$ will return a hierarchy of clusters given a set of programming elements. The function $ClosestClusterPair$ finds the closest pair in a set of clusters. The closeness is measured by weighted distance function defined in Equation 1. $BuildClusterHierarchy$ function has one loop that runs n=|Γ| times and invokes $ClosestClusterPair$ with $currentClusters$ inside the loop. Function $ClosestClusterPair$ has a two level nested loop, each of the loops runs m=$|currentClusters|$ times. Therefore the worst case run-time complexity is $O(mn^2)$.

Figure 3 shows the Agglomerative clustering steps with a set of predefined weights over the $Pebble\ Dropping$ program. We assume that self-weight is 1, call weight is 0.1, read weight is 0.2 and write weight is 0.4. The numbers in the small circular nodes indicate the step number, that is, in which step two clusters were merged. These node also indicate a cluster composed of two other clusters from each of which there is an edge to this node. In the very beginning `verification` and `move` are the closest pairs and they merge into 1. In the next step `main` and `assign` are closest, therefore they merge into 2. The process continues until all clusters merge into 14.

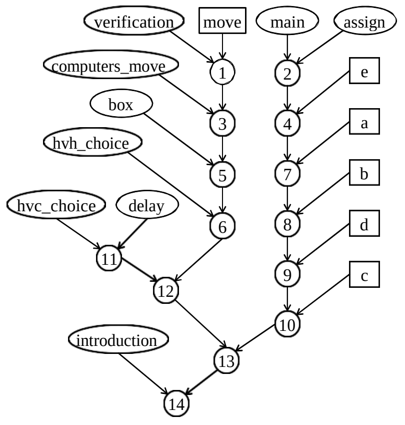

Fig.3. Step by Step Clustering.

C. Identifying the Desired Clusters

Each of the levels in a cluster hierarchy are candidates for being the object oriented design. We measure the quality of a level by three quality attributes of a class: high cohesion, low coupling and high granularity [18, 23]. For this purpose we define an objective function that is evaluated against each of the levels. The level that optimizes the objective function is selected as the object oriented design.

Siddik et al. mathematically formulated the problem in [18] as follows:

Let G(V,E) be the underlying undirected graph of a call graph. The nodes in same cluster are labeled with same number and nodes in different clusters are labeled differently. V and E be the set of vertices and edges respectively. We define variables l, x, y and z as follows:

$$C_v = \begin{cases} 1 \text{ if vertex } v \text{ is head of a cluster} \\ 0 \text{ otherwise} \end{cases}$$

$$X_e = \begin{cases} 1 \text{ if edge } e \text{ connects nodes with same label} \\ 0 \text{ otherwise} \end{cases}$$

$$Y_e = \begin{cases} 1 \text{ if edge } e \text{ connects nodes with different label} \\ 0 \text{ otherwise} \end{cases}$$

$$Z_{v,c} = \begin{cases} 1 \text{ if vertex } v \text{ has label c} \\ 0 \text{ otherwise} \end{cases}$$

The work in [18] formulated the problem as:

$$Maximize \sum_e x_e - \sum_e y_e + \sum_v |C_v|, \forall e \in E, \forall v \in V \quad (2)$$

Subject to:

$$x_e + y_e = 1, \qquad \forall e \in E$$
$$\sum_{c=1}^{L_G} Z_{v,c} = 1, \qquad \forall v \in V$$
$$\sum_{c=1}^{L_G} Z_{a,c} . Z_{b,c} = x_{a,b}, \qquad \forall a,b \in E$$
$$C_v = \bigcup_k Z_{kv}, \ \forall k = 1,2,\dots,n$$

Here C_v indicates if vertex v is head of a cluster, therefore $\sum_v C_v$ is the number of clusters. The objective function presented in Equation (2) maximizes intra-cluster edges, minimizes inter-cluster edge and maximizes number of clusters. The objective function is not normalized. We can perform comparison between clustering on same graph, however, we cannot compare between clusters of different graphs. That means, given a graph, we can determine the best clustering, but cannot state how good this particular clustering is. Because of these limitations, we use a modified version of this objective function.

We call our objective function the Omega function or simply Omega (Ω). Let G(V, E) is the underlying undirected graph of a clustered WDCG. The nodes in same cluster are labeled with same number and nodes in different clusters are labeled with different numbers. The labels start from 1 and ends to number of clusters. We define the following variables for this graph:

$$w_e = \text{weight of an edge } e$$
$$l_G = \text{number of labels in graph } G$$
$$x_e = \begin{cases} w_e \text{ if edge } e \text{ connects nodes with same label} \\ 0 \text{ otherwise} \end{cases}$$
$$y_e = \begin{cases} w_e \text{ if edge e connects nodes with different label} \\ 0 \text{ otherwise} \end{cases}$$
$$Z_{v,c} = \begin{cases} 1 \text{ if vertex } v \text{ has label c} \\ 0 \text{ otherwise} \end{cases}$$

The variables $l_G, x_e$and y_e are normalized as following variables for graph $G(V,E)$:

$$\beta_G = \frac{l_g}{|V|}$$
$$\delta_G = \frac{\sum_e x_e}{\xi_G}, \qquad \forall e \in E$$
$$\theta_G = \frac{\sum_e y_e}{\xi_G}, \qquad \forall e \in E$$

Where ξ_G is defined as,

$$\xi_G = w_e, \qquad \forall e \in E$$

We call variables β_G, δ_G and θ_G cluster density, intra-cluster edge density and inter-cluster edge density respectively. β_G indicates number of clusters per vertex. δ_G Indicates ratio of intra-cluster edge with respect to total number of edges and θ_G indicates ratio of inter-cluster edge with respect to total number of edges. Cluster density corresponds to modularity, intra-cluster edge density corresponds to cohesion and inter-cluster edge density corresponds to coupling of clusters respectively. As these clusters represent the classes in the object oriented design, we want that the design should be as modular as possible, highly cohesive and loosely coupled. Based on these we define our objective function Ω:

$$Maximize\ \Omega(G) = \delta_G - \theta_G + \beta_G \quad (3)$$

Maximizing the values of δ_G and β_G and minimizing value of θ_G will maximize Ω. However, maximizing β_G minimizes δ_G and maximizes θ_G. When each vertices are in different clusters, β_G has its maximum value, which is 1. In this case all the edges are inter-cluster which results δ_G to be 0 and θ_G to be 1. β_G is minimum when all clusters are in same cluster. In this case δ_G has its maximum value 1 and θ_G has minimum value 0. Mathematically:

$$0 < \beta_G \leq 1$$
$$0 \leq \delta_G \leq 1$$
$$0 \leq \theta_G \leq 1$$

When applying Agglomerative clustering on the WDCG, clustering of each step is a candidate object oriented design. We apply the objective function Ω on clustering of each of the steps. The step that produces the maximum value of Ω is considered to be the optimized clustering.

D. Properties of Obtained Classes

Each of the clusters in the optimized clustering corresponds to a class in the object oriented design. In addition to the function and data membership, we can derive has-a relationships among the classes. When a cluster has a inter-cluster edge to another cluster, the class corresponding to the second cluster belongs to the class corresponding to the second cluster, i.e., the first class has the second class. Moreover, we can also identify access levels of the members of a class. If there is one or more inter-cluster edge incident to a member of a class, the member is public. It is private otherwise.

For the *Pebble Dropping* program, step 10 from the Figure 3 produces the maximum value of Ω. Figure 4 shows the class diagram for clusters this step. There has a relationship within the classes and the access level are also shown in the class diagram. The class design proposed by a group of professional software engineers is shown in Figure 5. We call this design the expected class design. It is seen that though the number of classes in the diagrams are not equal, the larger classes are nearly identical.

V. Evaluation

We have implemented the proposed procedural to object oriented design migration technique and performed an empirical evaluation. In this section we present the evaluation process and analyze the result.

A. Experimental Setup

We have analyzed the result with five C programs. Some of the data for the evaluation process are collected from academic projects of Institute of Information

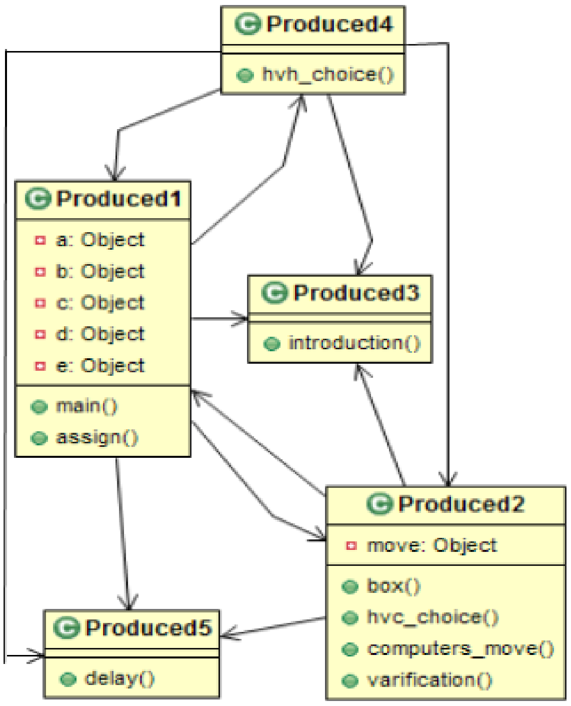

Fig.4. Produced Class Design of *Pebble Dropping* Program

Technology, University of Dhaka and the others are collected from online repositories. The data set are of different size having varying number of programming elements. Size of the individual data set are given in Table 1. The scheme was implemented in Java platform and the source code is hosted on *Bitbucket* (*bitbucket.org / mohayemin/designmigration*) project hosting service.

Four engineers helped us in the evaluation process. Among them two are software engineers having three years of experience. They prepared an object oriented design for each of the dataset. Their designs were reviewed by two other engineers. One of them is a Principal Software Engineer specialized in object oriented design. The other is a Project Manager specialized in design patterns. Both of them have more than eight years of industry experience. In addition to experience in software industry, they are also guest lecturers in their specialized fields at Institute of Information Technology, University of Dhaka. We have considered the designs provided by this group of people as benchmark for result evaluation.

B. Evaluation

For each dataset, we have calculated similarity between design produced by our technique with result provided by a group of software engineers using Jaccard similarity coefficient. For calculating similarity of a dataset, we take classes generated by our technique. We call this class design actual class design and the class design provided by the group of software engineers expected class design. For each of the actual classes, we find a best matching cluster in the expected classes. If no matching pair is found for a class, it is paired with class with no members.

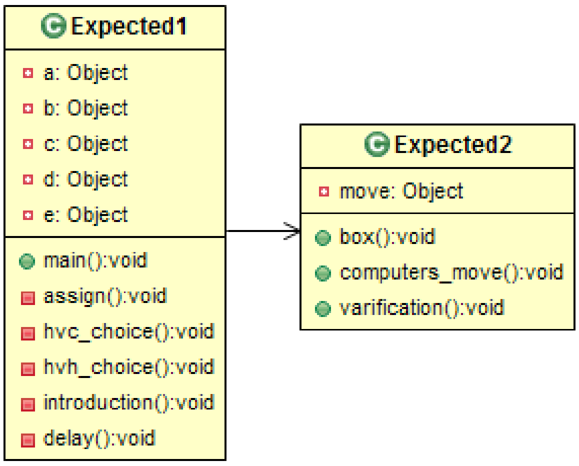

Fig.5. Expected Class Design of *Pebble Dropping* Program

Let, E = {{a, b}, {c, d, e}, {f, g}, {h, i, j, k}} be the expected clustering of an procedural code and A = {{a, b}, {c, d}, {e}, {f, g, h}, {i, j, k}} be clustering generated by our system. Here each $C \in E \cup A$ are clusters and each $e \in C$ are members of cluster. For example, h, i, j and k forms a cluster in expected result. We use p, p_e and p_a to denote a pair, expected set of the pair and actual set of the pair by respectively. We establish the following mapping for E and A:

$pair(p)$	$pair(p)$		$pair(p)$
1	$\{a,b\}$	$\rightarrow$	$\{a,b\}$
2	$\{c,d\}$	$\rightarrow$	$\{c,d,e\}$
3	$\{e\}$	$\rightarrow$	$\{\}$
4	$\{f,g,h\}$	$\rightarrow$	$\{f,g\}$
5	$\{i,j,k\}$	$\rightarrow$	$\{h,i,j,k\}$

We measure distance between a pair with the Jaccard Similarity Coefficient. For a pair p similarity

$$\emptyset_p = \frac{|p_a \cap p_e|}{|p_a \cup p_e|}$$

For example, similarity of pair-5 is $\left|\frac{\{i,j,k\}\cap\{h,i,k,l\}}{\{i,j,k\}\cup\{h,i,k,l\}}\right| = 0.75$. We calculate the similarity between each pair. Direct mean of the pairwise similarity will produce incorrect result. Because we expect that a large expected cluster should have more effect on the final similarity than a small expected cluster. To achieve this, we calculate weighted sum of the pairwise similarities. Weight of a pair is ratio of the cardinality of expected clustering to the total number of elements in all clusters. For a pair p, we define

$$\text{Weighted Pair Similarity}, \varphi_p = \frac{|p_e|}{|E|}.\emptyset_p \qquad (4)$$

Finally, similarity of all pairs *P* of a data set *D*:

$$\psi_D = \sum_p \varphi_p, \quad \forall p \in P \qquad (5)$$

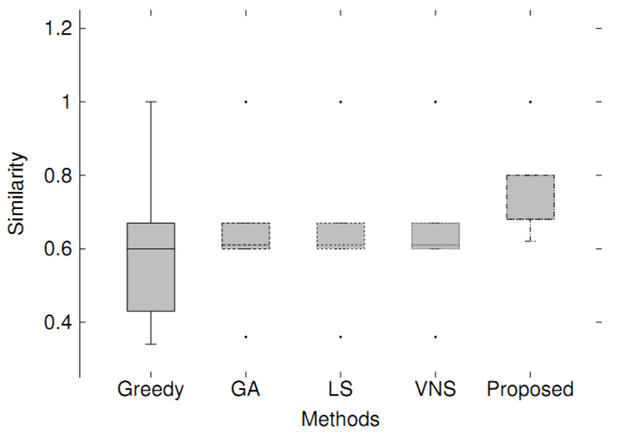

Fig.6. Box Whisker Plot Representing the Similarity between Expected and Actual Class Design for Different Programs using Different Methods.

We run the same process with a single data by changing weights of different type of edges in WDCG. We set value of self-weight to 1. For other three weights, we use values from 0.1 to 0.6 with an interval of 0.1. We find the weight that generates best result for each data and these results are shown in Table 1. We can see that for the best case call weight, read weight and write weights are 0.1, 0.4 and 0.5 respectively.

The proposed approach was compared with recent state of the art techniques. The result is shown in Table 2. It can be seen that the proposed method have outperformed the state of the art techniques. To better visualize the performance, the similarity using each approaches for each of the five programs is presented in Figure 6. From the figure it can be seen that the score of proposed scheme is not only higher but also more consistent than other techniques.

Our experiment found that Ω is biased towards value of β, i.e. the measure of modularity. Generated results created more clusters than the ones suggested by the software engineers. For many cases we found pair p where $p_e = \{\}$ which results φ_p to be 0. Expected clustering for first three programs were all the elements in a single class. These three programs were very cohesive therefore produces high X in Ω. For first two data the cohesiveness were strong enough to overcome bias of modularization thus produced desirable results.

From Equation 4 we can derive, for a pair $p, \varphi_p \propto |p_e$. In most of the cases, the value of p_a turns out to be small when the value of p_e is 0. This is because large classes are more likely to find a matching class than small classes. Therefore larger $\emptyset$ values are multiplied with larger $|p_e|$ values and small $\emptyset$ values are multiplied with zero or small values. This helps to prevent excessive number of small classes in our design that introduces high coupling. In other words, it allows to keep balance between modularity and coupling making the overall similarity satisfactory.

VI. Conclution and Future Work

This paper introduces a new technique for migrating the design of procedural program to object oriented architecture. We have formulated this design migration scenario as a weighted graph clustering problem. We have introduced new concepts in the literature to support our technique: Weighted Data Call Graph, a grammar to represent Weighted Data Call Graph, and a special type of set called entity map and operations specific to entity map. An inverse variant of Jaccard Similarity Coefficient called Weighted Distance Matrix has been proposed to compute similarity between two nodes of a weighted graph. Similar nodes based on high modularity, high cohesion and low coupling are clustered using Agglomerative Hierarchical Clustering. Each of the clusters are then considered as a class.

We have applied our technique on five C programs. The results produced by our technique have been compared with designs provided by a group of professional software engineers. Similarity between engineers' design and the design generated by our system has been calculated using Jaccard Similarity Coefficient. The experiment shows that in best cases the approach yields identical designs and on average case the designs are 75.6% similar. The proposed system can identify optimal clusters as potential classes of a system. Our future plan is to identify interfaces that classes implement.

Table 1. Results of Experiments Illustrating the Performance of the Proposed Scheme

Application Name	Call Edge Weight	Read Edge Weight	Write Edge Weight	similarity Ψ
LCS	0.40	0.20	0.40	0.83
Supermarket				0.68
Student Report Card				0.68
Pebble Dropping				0.28
Calendar				0.33
Average				0.56
LCS	0.20	0.20	0.60	0.83
Supermarket				0.68
Student Report Card				0.68
Pebble Dropping				0.80
Calendar				0.33
Average				0.664
LCS	0.20	0.30	0.50	0.83
Supermarket				0.68
Student Report Card				0.68
Pebble Dropping				0.80
Calendar				0.33
Average				0.664
LCS		0.30	0.60	1.00
Supermarket	0.10			0.68
Student Report Card				0.68
Pebble Dropping				0.80
Calendar				0.33
Average				0.698
LCS	0.10	0.40	0.50	1.00
Supermarket				0.68
Student Report Card				0.68
Pebble Dropping				0.80
Calendar				0.62
Average				0.756

Table 2. Results of Experiments Illustrating The Performance of The Proposed Scheme in Terms of Ψ

Application Name	Greedy [18]	GA [33]	LS [34]	VNS [35]	Proposed
LCS	1.00	1.00	1.00	1.00	1.00
Supermarket	0.67	0.67	0.67	0.67	0.68
Student Report Card	0.43	0.36	0.36	0.36	0.68
Pebble Dropping	0.34	0.61	0.61	0.61	0.80
Calendar	0.60	0.60	0.60	0.60	0.62
Average	**0.608**	**0.648**	**0.648**	**0.648**	**0.756**

Moreover, considering each identified classes as clusters, we can apply a second layer of Agglomerative clustering and mark the result as components.

ACKNOWLEDGMENT

This research was conducted by Optimization Research Group of Institute of Information Technology, University of Dhaka. Our sincere gratitude to Dr. Shahadat Hossain, Associate Professor, Department of Math \& Computer Science, University of Lethbridge for presenting this problem to us.

REFERENCES

[1] Meir M Lehman. *Programs, life cycles, and laws of software evolution*. Proceedings of the IEEE, 68(9):1060–1076, 1980.

[2] Andrea De Lucia, Giuseppe A Di Lucca, Anna Rita Fasolino, Patrizia Guerra, and Silvia Petruzzelli. *Migrating legacy systems towards object- oriented platforms*. In 13th International on Software Maintenance (ICSM 1997), pages 122–129, Bari, Italy, 1997. IEEE.

[3] Harry M Sneed. *Integrating legacy Software into a Service oriented Architecture*. In 10th European Conference on Software Maintenance and Reengineering (CSMR 2006), pages 3–14, Bari, Italy, 2006. IEEE.

[4] K. Bennett. *Legacy systems: coping with stress*. IEEE Software, 12(1):19–23, Jan 1995.

[5] KC Chisolm and JC Lisonbee. *The use of computer language compilers in legacy code migration. In IEEE Systems Readiness Technology*. Conference (AUTOTESTCON'99), pages 137–145, San Antonio, TX, USA, September 1999. IEEE.

[6] Ruben Prieto-Diaz and Peter Freeman. *Classifying software for reusability*. IEEE Software, 4(1):6–16, 1987.

[7] Wei Li and Sallie Henry. *Object-oriented metrics that predict maintain- ability*. Journal of systems and software, 23(2):111–122, 1993.

[8] Robert W Schwanke. *An intelligent tool for re-engineering software modularity*. In 13th International Conference on Software Engineering (ICSE 1991), pages 83–92, Baltimore, Maryland, USA, 1991. IEEE.

[9] S. Pidaparthi and G. Cysewski. *Case study in migration to object oriented system structure using design transformation methods*. In First Euromicro Conference on Software Maintenance and Reengineering (EUROMICRO 97), pages 128–135, Berlin, Germany, Mar 1997. IEEE.

[10] Alan Snyder. *Encapsulation and inheritance in object-oriented programming languages*. ACM Sigplan Notices, 21(11):38–45, 1986.

[11] Shyam R Chidamber and Chris F Kemerer. *A metrics suite for object oriented design*. IEEE Transactions on Software Engineering, 20(6):476–493, Jun 1994.

[12] Andrian Marcus and Denys Poshyvanyk. *The conceptual cohesion of classes*. In 21st IEEE International Conference on Software Maintenance (ICSM 2005), pages 133–142, Budapest, Hungary, 2005. IEEE.

[13] Rachel Harrison, Steve Counsell, and Reuben Nithi. *Coupling metrics for object-oriented design*. In 5th International Software Metrics Symposium (Metrics 98), pages 150–157. IEEE, 1998.

[14] B. G. Ryder. *onstructing the call graph of a program*. IEEE Transactions on Software Engineering, SE-5(3):216–226, May 1979.

[15] Jane Radatz, Anne Geraci, and Freny Katki. *IEEE standard glossary of software engineering terminology*. IEEE Std, 610121990.

[16] Y. Terashima and K. Gondow. *Static call graph generator for C++ using debugging information*. In 14th Asia-Pacific Software Engineering Conference (APSEC 2007), pages 127–134, Aichi, Japan, Dec 2007. IEEE.

[17] Satu Elisa Schaeffer. *Graph clustering*. omputer Science Review, 1(1):27–64, Aug 2007.

[18] Saeed Siddik, Alim Ul Gias, and Shah Mostafa Khaled. *Optimizing software design migration from structured programming to object oriented paradigm*. In 16th International Conference on Computer and Information Technology (ICCIT 2013), pages 187–192, Khulna University, Khulna, Bangladesh, December 2013. IEEE.

[19] V Dineshkumar and J Deepika. *Code to design migration from structured to object oriented paradigm*. International Journal of Information and Communication Technology Research, 1(8), December 2011.

[20] Arie van Deursen and Tobias Kuipers. *Identifying objects using cluster and concept analysis*. In 21st International Conference on Software Engineering (ICSE ’99), pages 246–255, Los Angeles, California, USA, 1999. ACM.

[21] Zou Ying and Kostas Kontogiannis. *A framework for migrating procedural code to object-oriented platforms*. In Eighth Asia-Pacific Software Engineering Conference (APSEC 2001), pages 390–399. IEEE, Dec 2001.

[22] Mark Shtern and Vassilios Tzerpos. *Methods for selecting and improving software clustering algorithms*. Software: Practice and Experience, 44(1):33–46, 2014.

[23] Lionel C. Briand, J u ̈rgen Wu ̈st, Stefan V. Ikonomovski, and Hakim Lounis. *Investigating quality factors in object-oriented designs: An industrial case study*. In 21st International Conference on Software Engineering (ICSE 1999), pages 345–354, Los Angeles, California, USA, 1999. ACM.

[24] H.M. Sneed and E. Nyary. *Extracting object-oriented specification from procedurally oriented programs*. In 2nd Working Conference on Reverse Engineering, pages 217–226, Toronto, Ontario, Canada, Jul 1995. IEEE.

[25] Bill Andreopoulos, Aijun An, Vassilios Tzerpos, and Xiaogang Wang. *Clustering large software systems at multiple layers*. Information and Software Technology, 49(3):244–254, 2007.

[26] Onaiza Maqbool and Haroon A Babri. *Hierarchical clustering for software architecture recovery*. IEEE Transactions on Software Engineering, 33(11):759–780, Nov 2007.

[27] M.A. Heroux and J.M. Willenbring. *Barely sufficient software engineering: 10 practices to improve your CSE software*. In ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE ’09), pages 15–21, Vancouver, BC, Canada, May 2009. IEEE.

[28] P. E. Livadas and P. K. Roy. *Program dependence analysis*. In Conference on Software Maintenance, pages 356–365, Orlando, FL, USA, Nov 1992. IEEE.

[29] S.-S. Liu and N. Wilde. *Identifying objects in a conventional procedural language: an example of data design recovery*. In Conference on Software Maintenance, pages 266–271, San Diego, CA, USA, Nov 1990. IEEE.

[30] Emdad H. Khan, Mansoor Al-A’ali, and Moheb R. Girgis. *Object oriented programming for structured procedural programmers*. Com- puter, 28(10):48–57, Oct 1995.

[31] D. Doval, S. Mancoridis, and B.S. Mitchell. *Automatic clustering of software systems using a genetic algorithm*. In Software Technology and Engineering Practice (STEP 1999), pages 73–81, Pittsburgh, PA, USA, Sep 1999. IEEE.

[32] Marc Eisenstadt, Blaine A Price, and John Domingue. *Redressing ITS fallacies via software visualization*. In Cognitive models and intelligent environments for learning programming, volume 111, pages 220–234. Springer Berlin Heidelberg, 1993.

[33] Md Selim, Saeed Siddik, Alim Ul Gias, M Abdullah-Al-Wadud, and Shah Mostafa Khaled. *A genetic algorithm for software design migration from structured to object oriented paradigm*. In 8th International Conference on Computer Engineering and Application (CEA 2014), pages 187–192, January 2014.

[34] Saeed Siddik, Alim Ul Gias, Md. Selim, Shah Mostafa Khaled, and Kazi Sakib. *A direction of migrating procedural paradigm to object based architecture by forming cluster of functions using local search heuristics*. In 3rd International Conference Informatics, Electronics & Vision, pages 1–6, University of Dhaka, Bangladesh, May 2014. IEEE.

[35] Md Selim, Md Saeed Siddik, Tajkia Rahman, Alim Ul Gias, and Shah Mostafa Khaled. *Approximating object based architecture for legacy software written in procedural languages using variable neighbor- hood search*. In 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014), pages 1–6, December 2014.

[36] Michael W Trosset. *Distance matrix completion by numerical optimiza- tion*. Computational Optimization and Applications, 17(1):11–22, 2000.

[37] Alan Bundy and Lincoln Wallen. *Context-free grammar*. In Catalogue of Artificial Intelligence Tools, Symbolic Computation, pages 22–23. Springer Berlin Heidelberg, 1984.

[38] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. *Graph clustering based on structural/attribute similarities*. VLDB Endowment, 2(1):718–729, Aug 2009.

[39] Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. *Fast algorithms for projected clustering*. SIGMOD Rec., 28(2):61–72, Jun 1999.

[40] Kuo-Lung Wu and Miin-Shen Yang. *Alternative c-means clustering algorithms*. Pattern recognition, 35(10):2267–2278, Oct 2002.

[41] Pierre Hansen and Brigitte Jaumard. *Cluster analysis and mathematical programming*. Mathematical Programming, 79(1-3):191–215, 1997.

[42] Aftab Hussain and Md. Saidur Rahman. *A new hierarchical clustering technique for restructuring software at the function level*. In 6th India Software Engineering Conference (ISEC 2013), pages 45–54, New Delhi, India, 2013. ACM.

[43] Edie M Rasmussen. *Clustering algorithms. Information retrieval: data structures & algorithms*, pages 419—442, 1992.

Authors' Profiles

Mohayeminul Islam completed M.Sc in Software Engineering from Institute of Information Technology with thesis "Design Migration from Procedural to Object Oriented Program by Clustering Data Call Graph" under the supervision of Shah Mostafa Khaled. He received his Bachelor's degree in Information Technology (Major in Software Engineering) from the same institute. Mohayemin completed his internship at M\&H Informatics (BD) Ltd. from July-December 2011. Currently he is working as software engineer at Jantrik Technologies Ltd. Bangladesh.

Tajkia Rahman Toma completed her M.Sc in Software Engineering degree from the Institute of Information Technology. She received her Bachelor's degree in Information Technology (Major in Software Engineering) from the same institute. Tajkia completed her internship at M\&H Informatics (BD) Ltd. from July-December 2011. Tajkia worked as a Honorary Research Assistant at IITDU Optimization Group during June-September 2014. Since September 2014 she is working as software engineer at Jantrik Technologies Ltd. Bangladesh.

Md. Selim completed M.Sc in Software Engineering from Institute of Information Technology with thesis "Source Code Analysis for Design Migration: A Guideline for Procedural to Object Oriented Paradigm Migration" under the supervision of Shah Mostafa Khaled. Earlier Selim completed B.Sc in Software Engineering from same institute with research titling ``A Genetic Algorithm for Design Migration from Structured to Object Oriented Paradigm". He completed his internship during January-June 2013 at the Binary Quest Limited, Bangladesh. He is currently working as a Lecturer at Department of Disaster Science & Management, University of Dhaka.

Alim Ul Gias completed M.Sc. in Software Engineering at Institute of Information Technology with thesis on Adaptive Software Testing. He received his Bachelor's degree in Information Technology (Major in Software Engineering) from the same institute. Alim completed his internship at Grameenphone Ltd. Bangladesh from July-December 2011. He was a student member of ACM and SIGSOFT and at present, a member of IEEE. Currently he has been working as a Lecturer at Institute of Information Technology, University of Dhaka.

Shah Mostafa Khaled completed his B.Sc. and M.Sc. from Department of Computer Science and Engineering, University of Dhaka. Khaled completed his second masters in Computer Science from University of Lethbridge, Canada with a thesis titling "Heuristic Algorithms for Wireless Mesh Network Planning" under the supervision of Robert Benkoczi. Theoretical Optimization, Operations Research and Machine Learning are his areas of interest. Khaled is now the coordinator of IIT DU Optimization Research Group.

How to cite this paper: Mohayeminul Islam, Tajkia Rahman Toma, Md. Selim, Alim Ul Gias, Shah Mostafa Khaled,"Design Migration from Procedural to Object Oriented Paradigm by Clustering Data Call Graph", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.8, No.2, pp.1-13, 2016. DOI: 10.5815/ijieeb.2016.02.01