SOFTWARE RE-STRUCTURING
An Architecture-Based Tool

Violeta Bozhikova, Mariana Stoeva, Anatoly Antonov and Vladimir Nikolov
Technical University, Str.“Studentska” N^{o}1, Varna, Bulgaria

Keywords: Software re-structuring, software decomposition, program understanding, software clustering algorithms.

Abstract: The practice shows that many software systems have been evolving for many years and are now large and complex. Because the structure of these systems is usually not well documented, great research effort is needed to find appropriate abstractions of their structure, that we can simplify their maintenance, evolution and adaptation. A variety of techniques and tools are developed trying to effectively solve this problem. In this paper an Architecture-Based Framework for software re-structuring is discussed. Next, how this framework is implemented in an ever evolving and user-driven tool that can effectively support the software re-structuring process is commented.

1 INTRODUCTION

Since many software systems are now large, complex and poorly documented appropriate abstractions of their structure are needed to simplify program understanding and software re-structuring (D. Doval et al., 1999). A lot of techniques and tools are now developed to effectively support software architecture decomposition than facilitating software maintenance, software evolution and re-engineering.

Since software decomposition is known to be NP-hard, obtaining a good partition by exhaustive exploration of the search space is unlikely. That is why the researchers focused on using heuristic search techniques (B. S. Mitchell et al., 2002) to find “good enough” solutions quickly.

A lot of software clustering approaches can be found in the reverse engineering literature, each one using a different algorithm to identify clusters. Cluster analysis has been used in many disciplines to support grouping of similar objects (highly dependent objects) of a system. The resulting groups are called clusters.

With Formal Concept Analysis (FCA) we can identify similarities among a set of program objects based on their attributes. Using Program Slicing we can locate within the source code that objects that use common data items.

We have developed an architecture-based framework that integrates a group of heuristic techniques to solve this problem. A tool is developed (Божикова В.Т, 2001) that implements this framework and can be used to effectively support software re-structuring process. The newer version of this evolving tool (V.Bozhikova et al., 2007) is more interactive and flexible, enabling to better manage the re-structuring process. We can choose appropriate analysis technique; we can enter weights of components; we can change the restrictive conditions. In the paper, we briefly describe the framework and show how this framework is implemented in a software re-structuring tool. Using such tools we hope effectively extend the usable life of a legacy application recovering or re-organizing its structure.

2 ARCHITECTURE DECOMPOSITION

The problem of how to re-structure a software system can be seen as an architecture decomposition problem. Software architecture defines the components of the software system and embodies information (L. Bass et al., 1998), about how the components interact with each other. Software architecture is a collection of different structures (module structure, process structure, conceptual structure, uses structure, call structure, class structure etc.), different kind of components (modules, processes, procedures, objects…) and

Bozhikova V., Stoeva M., Antonov A. and Nikolov V. (2008).
SOFTWARE RE-STRUCTURING - An Architecture-Based Tool.
In *Proceedings of the Third International Conference on Software and Data Technologies - SE/GSDCA/MUSE*, pages 269-273
DOI: 10.5220/0001873402690273

relationships among the components (calls, synchronization relations, relation type inherits-from…), more than one kind of context (development time, runtime). Decomposition is one of the main operations in the software architecture theory. This operation is used to separate the system's structure or a large system's component into more, smaller ones sub-components. The result of this operation is a higher-level software structure.

3 ARCHITECTURE-BASED FRAMEWORK (ABFSR)

We have developed an architecture-based clustering framework, called ABFSR that can be used to come to a solution of the early defined problem. Central to ABFSR are the architectural structures.

Some notions in the architecture-based framework described below are adopted from Symphony (A. van Deursen et al., 2004). Like Symphony the process of ABFSR contains 2 stages. Let us describe briefly this process:

1. Decomposition Design
2. Decomposition Performance

During the first stage:

- Target structure is selected;
- Source structure is defined;
- Mapping rules are defined;
- Evaluation techniques and reference structures are defined.

The source structure can be extracted from software artefacts (A. van Deursen et al., 2004), such as source code, build files, configuration information, documentation or traces. The target structure is the structure that we hope to solve the problem. The mapping rules define how the target structure is created from the source structure and depends on the analysis method used.

During the second stage the mapping rules are applied to source structure to obtain the target structure, earlier defined (during the reconstruction design). This stage has only 2 steps:

- Knowledge extraction
- Information interpretation

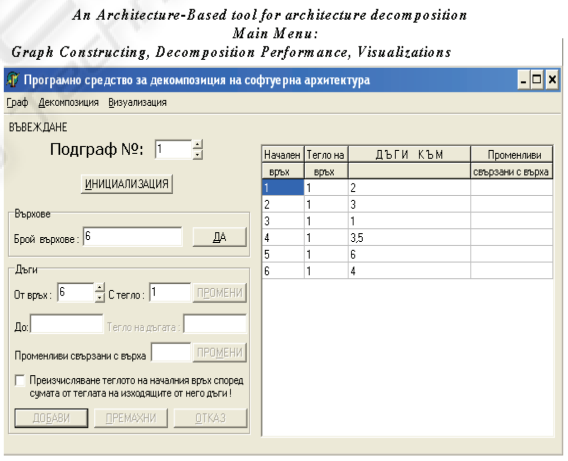
During the knowledge extraction step, the mapping rules, defined during reconstruction design are applied to the source structure and a target structure is obtained. During the next step the created target structure is analyzed and evaluated. Comparison with a reference structure is made. Results from this stage lead to a refined decomposition design, to choose a new analysis method or to choose a new mapping rule to eventually come to a satisfactory solution of the problem.

4 A TOOL FOR SOFTWARE RE-STRUCTURING

It is important to note that ABFSR has been practically in use since 2001. The re-structuring tool described in (Божикова et.al., 2001) is the first tool version and the first implementation of this methodology. Some advantages of the new version of the tool for software restructuring may be summarized as follows:

a) It is more interactive enabling the user to better manage the re-structuring process. Figure 1 shows the main window. Users can specify the source structure as weighted oriented graph.

b) It is more flexible, enabling the user to choose an appropriate analysis method and the appropriate algorithm that based on a particular analysis method. Figure 3 shows the window that appears when choosing a cluster analysis method. Figure 4 appears when a FCA method is chosen.


An Architecture-Based tool for architecture decomposition
Main Menu:
Graph Constructing, Decomposition Performance, Visualizations
Програмно средство за декомпозиция на софтуерна архитектура
Граф
Декомпозиция
Визуализация
ВЪВЕЖДАНЕ
ИНИЦИАЛИЗАЦИЯ
Върхове
ДА
Дъги
ПРОМЕНИ
ПРОМЕНИ
ДОБАВИ
ПРЕМАХНИ
ОТКАЗ

Подграф №: 1

Брой върхове: 6

От връх: 6 С тегло: 1

До: Тегло на дъгата:

Променливи свързани с върха

☐ Преизчисляване теглото на началния връх според сумата от теглата на изходящите от него дъги !

Начален връх	Тегло на връх	ДЪГИ КЪМ	Променливи свързани с върха
1	1	2	
2	1	3	
3	1	1	
4	1	3,5	
5	1	6	
6	1	4	

Figure 1: The main window.

When choosing a particular analysis method we can use different mapping rules (algorithms) to transform the source structure to a target structure. Mapping rules (i.e. algorithms) depend on the analysis method that is used. The tool enables the user to create the target structure using: a cluster analysis method (fig.2), FCA method and program slicing technique (fig.3). For each analysis method static relations between entities are taken as source viewpoint. Special external tools can be used to extract static information from software artefacts and to present this information graphically (Mancoridis et al., 2001). We start with inputting the source structure presented as a weighted oriented graph. Below the main window is presented (fig.1):

The re-structuring process is supported through a set of heuristic algorithms. The task for finding the optimal solution to the software re-structuring problem is known to be NP-hard. That is why many researchers focused on using different heuristic techniques (Mancoridis, 2001) that find "good enough" solutions quickly.

c) An improvement of the early included algorithms is realized in the tool.

We aim at improving our algorithms permanently. The "Tabu-Search" clustering algorithm is (V.Bozhikova et.al, 2007) an improvement of an early developed descent-hill climbing algorithm for software clustering. It is based on weighted Module Dependence Graph – MDG=(X, U). The components of the source structure are modelled as the set of graph's nodes (X, N=|X|), and the source code dependences (inherit, call, instantiated) are modelled as the set of graph's edges (U). The quality of the resulting target structure is evaluated through a quality function that measures the number "k" of the static dependences between the clusters (1):

$$k = \frac{1}{2} \sum_{i=1..M} \sum_{j=1..M} k_{ij} = \min, \forall i \neq j \qquad (1)$$

In the case, "k" is based on the "inter-connectivity" - the solution with a lowest value of "k" would be the best solution to the problem. Let x_i denote the node with an index i (i=1…N) and a weight of w_i. Let "M" is the number of clusters in the target structure. The weight W_i (2) of each cluster "i" is the sum of the weights of all nodes in the cluster "i". W_i must be less then W_0, where W_0 is a user-defined restrictive condition.

$$W_i \leq W_0 \qquad (2)$$

Figure 2.a shows the target structure that is a result of re-structuring the presented MDG in (Doval et al., 1999). We observe that our Tabu-Search algorithm produces a partition with a better quality - k=29, comparing the reference structure in figure 2-b. The result of applying the similarity measurement technique (Rainer Koschke et al., 2000) is also encouraging: the similarity between the two partitions is good.

2.a) k=29 2.b) k=32

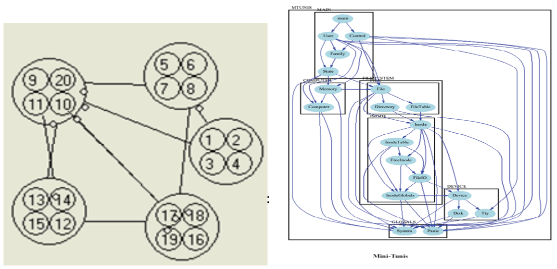
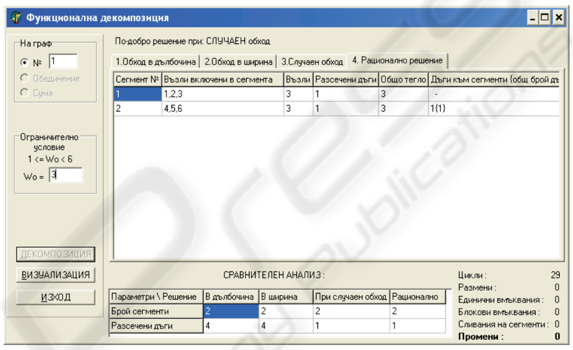


Figure 2: The target (a) and the reference (b) structures. Similarity ≈ 66%.



На граф
[✓ №] 1
Обединение
Сума

Ограничително условие
1 <= Wo < 6
Wo = 3

По-добро решение при: СЛУЧАЕН обход

1.Обход в дълбочина | 2.Обход в ширина | 3.Случаен обход | 4. Рационално решение

Сегмент №	Възли включени в сегмента	Възли	Разсечени дъги	Общо тегло	Дъги към сегменти {общ брой дъ
1	1,2,3	3	1	3	-
2	4,5,6	3	1	3	1{1}

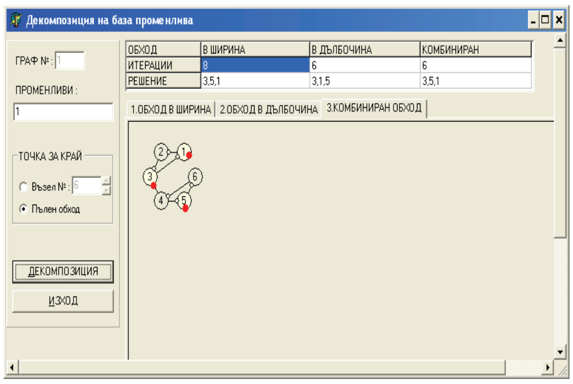
СРАВНИТЕЛЕН АНАЛИЗ :

Параметри \ Решение	В дълбочина	В ширина	При случаен обход	Рационално
Брой сегменти	2	2	2	2
Разсечени дъги	4	4	1	1

Цикли :	29
Размени :	0
Единични вмъквания :	0
Блокови вмъквания :	0
Сливания на сегменти :	0
Промени :	**0**

Figure 3: Using a Cluster Analysis method.

We have developed an approach that integrates FCA and Program Slicing and we show below (figure 4) the window visualizing the target structure. The source structure includes the smallest architectural components – subprograms with their features (the data used by the component, i.e. global variables, constants and user-defined types). To find the target view, our algorithm groups subprograms having a common features, i.e. using a common set of global data elements. These groups of subprograms become candidates for modules, i.e. buildings blocks for larger architectural components, at the next higher architectural level.



ГРАФ №: 1

ПРОМЕНЛИВИ : 1

ТОЧКА ЗА КРАЙ
Възел №: 6
[✓ Пълен обход]

ОБХОД	В ШИРИНА	В ДЪЛБОЧИНА	КОМБИНИРАН
ИТЕРАЦИИ	8	6	6
РЕШЕНИЕ	3,5,1	3,1,5	3,5,1

Figure 4: Using a FCA method.

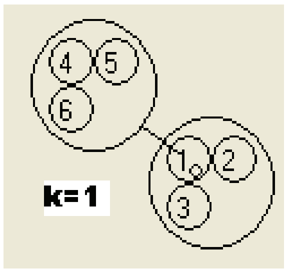


Figure 5: The visualized target structure using Cluster Analysis Method

We find it is not convenient to create fully automatic tools; we believe it is better to develop user-driven tools. Our tool offers an improved and more flexible user interface. To improve the final result we can visualize the source structure (figure 1), three intermediate structures and the target structure (figures 2a, 4 and 5). We consider graphical visualizations an important aid to support the processes of program understanding and re-structuring.

5 EVALUATION TECHNIQUES

Now that a lot of approaches to software re-structuring exist, the validation of produced target structures is starting to interest the researchers. Recently, researchers have begun developing an infrastructure to evaluate clustering techniques, in a semi-formal way by proposing similarity measurements (Rainer Koschke et al., 2001), (Mancoridis et al., 2001), (B. S. Mitchell et al. 2002). These measurements enable the results of clustering algorithms to be compared to each other, and preferably to be compared to reference structure agreed upon ("benchmark" standard). We emphasize that the reference structure need not be the optimal solution in a theoretical sense. Rather, it is a solution that is perceived as being "good enough".

There are a lot of similarity measurements for this purpose: Anquetil (Mancoridis et al., 2001) has published a similarity measurement technique, named "Precision and Recall"; Mojo has developed a technique that measures the distance between two decompositions of a software system by calculating the number of operations needed to transform a decomposition to an other; Tzerpos and Holt introduce also a distance quality measurement based on MoJo; Koschke and Eisenbarth (Rainer Koschke et al., 2000) have developed a framework for experimental evaluation of clustering techniques that we have used in our case study (figure 2). We have evaluated our algorithm on several small size systems (where N≤20) with similar success to the one demonstrated in figure 2. In the future, we intend to conduct further validation of our technique using other systems.

6 CONCLUSIONS

It is widely known that more than 50% of the costs of a software system have to be attributed to maintenance. There is an urgent need for methods and tools which assist software understanding and software restructuring, reducing the maintenance costs.

In the paper our Architecture-Based Framework (ABFST) for software re-structuring is briefly described. ABFST puts a strong emphasis on software architecture. We would like to point out that having a framework like ABFST can help practitioners by giving them guidance in performing an architecture re-structuring. In addition, ABFST can help researchers by providing a unified approach to re-structuring, with consistent terminology and a basis for improving, refining, quantifying, and comparing re-structuring processes.

Next the paper describes our ever evolving tool as an implementation of ABFST and marks some new features. The paper demonstrates how the source structure of a software system can be effectively re-structured and understood using our tool (figure 2). In the future, we plan to conduct and document more intensive studies to demonstrate the effectiveness and the limitations we find in the tool version. We hope to further improve both its performance adding new heuristics and its visualization futures.

ACKNOWLEDGEMENTS

This work is supported by a project in Technical University of Varna, untitled "Software Intensive Systems Research", with number 481 /2008.

REFERENCES

A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: View-driven software architecture reconstruction, *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture,* Washington, 2004.

L. Bass, P. Clemens, R. Kazman, *"Software architecture in practice"*, Addison-Wesley Longman, 1998.

Божикова В.Т., Карова М.Н., "Създаване, визуализация и операции на програмни структури", *Proceedings of the Int'l Scientific Conference on Energy and Information Systems and Technologies,* Bitola, 2001, Vol.3., pp. 813-819.

Mancoridis, Mitchell, "Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements", *IEEE Proceedings of the 2001 International Conference on Software Maintenance (ICSM'01).* Italy, 2001, pp. 744 -753.

Doval, Mancoridis, Mitchell. "Automatic Clustering of Software Systems using a Genetic Algorigthm", by In the IEEE *Proceedings of the 1999 International Conference on Software Tools and Engineering Practice (STEP'99)*, Pittsburgh, PA, 1999. pp. 73-81

Rainer Koschke and Thomas Eisenbath, "A Framework for experimental evaluation of clustering techniques", *International Workshop on Program Comprehension,* 2000.

B. S. Mitchell, S. Mancoridis, M. Traverso. "Search Based Reverse Engineering" In *ACM Proceedings of the 2002 International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, Ischia, Italy, 2002. pp. 431-438.

V. Bozhikova, M. Stoeva, "Applying Tabu-Search heuristic for software clustering problem", *ICEST'2007 - Proceedings of papers*, Volume 2, pp 861-864, Ohrid, Macedonia.