Code-Based Graph Representations And Software Reengineering

Violeta T. Bojikova[1], Mariana Ts. Stoeva[2]

Abstract - **The purpose of this work is to summarize the role of software reengineering in software maintenance process and to show the place of the software clustering process in the software-reengineering model. The code-based graph representation is one of the open problems, because the results of clustering process depend on this representation. In this paper we discuss and present our observations about the code-based graph representation problem.**

Keywords – **software reengineering, software-clustering algorithms, software representation**

I. INTRODUCTION

Companies often have legacy systems, which have to be maintained. Since the 1950's over 100 billion LOC have been written. Legacy software is a valuable asset, which cannot be easy discarded or redeveloped. Something has to be done to help keep the cost of maintenance down. When legacy software has a high business value and low changeability the reengineering is the recommended variant to its development (figure 1). Chikofsky and Cross define software reengineering as "the examination and alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form". This definition is technology oriented while Arnold's definition is goal-oriented: Software reengineering is defined by Arnold as "any activity that:

- improves one's understanding of software, or,
- prepares or improves the software itself, usually for increased maintainability, reusability, or evolve ability".

Although the two definitions do not necessarily correspond, both the above authors provide good introductions to software reengineering.

In figure 1 we try to summarize the characteristics of software reengineering activity: role, processes, goals, actual status, main problems and their solutions and show the place of the software clustering process in the software-reengineering model.

As we see in figure 1, code-based graph representation algorithms and tools (Code Analysis tools) are needed to help software maintainers.

II. THE GRAPHS IN THE SOFTWARE REENGINEERING AREA

Large software systems tend to have a rich and complex structure. Designers typically present the structure of software systems as one or more directed graphs. For example, a directed graph can be used to describe the modules, classes, functions, operators of a system and their static interrelationships using nodes and directed edges, respectively. Graphs are commonly used not only for system's visualization [10], but also as input for automatic clustering algorithms [1,2,3,10], the goal of witch is to extract the high level structure of the program under study. The problem of code-based graph representation is one of the open problems in software clustering area, because the results of clustering process depend on this representation.

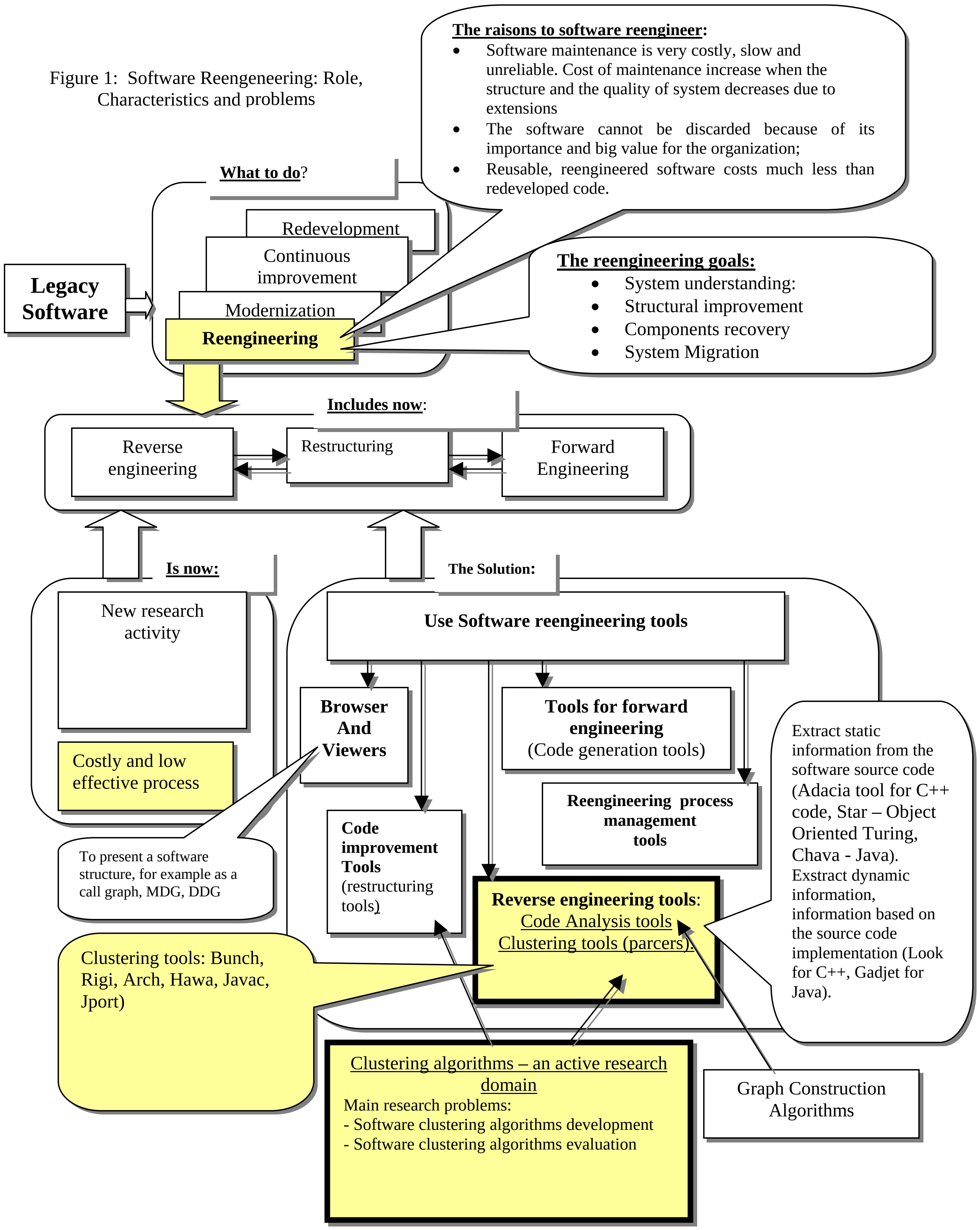
In Bunch (software clustering tool) [1,2,3] are used module dependency graphs (MDGs). MDG is directed graph that represents the software modules (e.g., classes, files, packages) as nodes, and the relations (e.g., function invocation, variable usage, class inheritance) between modules as directed edges (figure 3). The MDG of the clustered software systems in [1,2,3] are recovered automatically from its source code using tools such as CIA [8] (for C), Acacia [7] (for C and C++) or Star [9] for Turing [4]. However, even the MDGs of small systems can be complex. Then appropriate abstractions of their structure are needed to make them more understandable and, thus, easier to maintain [1,2,3]. Once the MDG is created, Bunch's clustering algorithms can be used to create the partitioned MDG.

A number of software clustering algorithms is developed to separate the graph's nodes (i.e., modules) into clusters (i.e., subsystems). Decomposing source code components and relations into subsystem clusters is an active area of research. Numerous clustering approaches have been proposed in the reverse engineering literature, each one using a different algorithm to identify subsystems than producing an architectural view.

The software reengineering literature shows that the problem of automatically creating abstract views of software structure is very computationally expensive (NPhard), so a hope for finding a general solution to the software is unlikely.

[1] Violeta Bojikova is with the Department of Computer Science Varna Technical University, Bulgaria,
e-mail: vbojikova2000@yahoo.com
[2] Mariana Stoeva is with the Department of Computer Science Varna Technical University, Bulgaria
e-mail: mariana_stoeva@abv.bg

Figure 1: Software Reengeneering: Role, Characteristics and problems


The raisons to software reengineer:
• Software maintenance is very costly, slow and unreliable. Cost of maintenance increase when the structure and the quality of system decreases due to extensions
• The software cannot be discarded because of its importance and big value for the organization;
• Reusable, reengineered software costs much less than redeveloped code.
What to do?
Redevelopment
Continuous improvement
Modernization
Reengineering
Legacy Software
The reengineering goals:
• System understanding:
• Structural improvement
• Components recovery
• System Migration
Includes now:
Reverse engineering
Restructuring
Forward Engineering
Is now:
New research activity
Costly and low effective process
The Solution:
Use Software reengineering tools
Browser And Viewers
Tools for forward engineering (Code generation tools)
Reengineering process management tools
Code improvement Tools (restructuring tools)
Reverse engineering tools: Code Analysis tools Clustering tools (parcers).
To present a software structure, for example as a call graph, MDG, DDG
Extract static information from the software source code (Adacia tool for C++ code, Star – Object Oriented Turing, Chava - Java). Exstract dynamic information, information based on the source code implementation (Look for C++, Gadjet for Java).
Clustering tools: Bunch, Rigi, Arch, Hawa, Javac, Jport)
Clustering algorithms – an active research domain
Main research problems:
- Software clustering algorithms development
- Software clustering algorithms evaluation
Graph Construction Algorithms

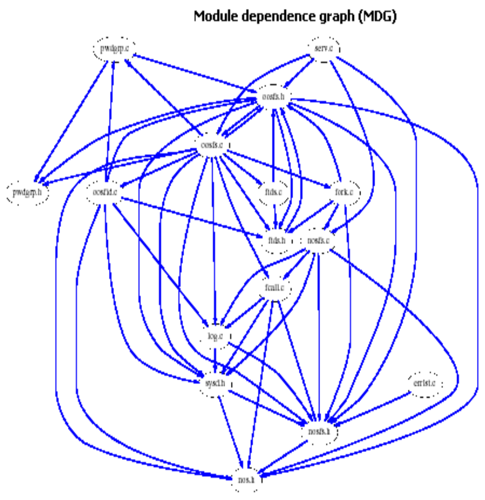
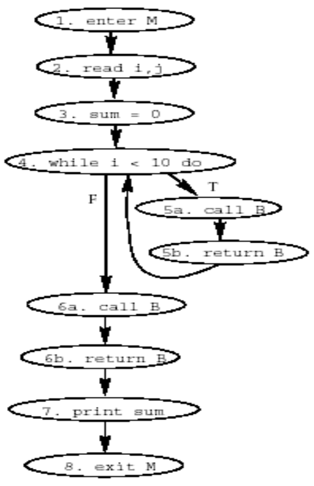


figure 3.



Control-flow graph — procedure M

```
  procedure M
1 begin M
2     read i,j
3     sum = 0
4     while i < 10 do
5         call B
      endwhile
6     call B
7     print sum
8 end M
```

figure 2.
Procedure M and the resulting CFG

In [6] are used call graphs (CG) [6] as input of automatic clustering algorithms. Call graphs and control flow graphs (CFG) are commonly used in the fields of program slicing (figure 2) and regressing testing [5]. The observation is that determining the call graph for a procedural program is fairly simple. However, this is not the case for programs written in object-oriented languages, due to polymorphism. A number of algorithms for the static construction of an object-oriented program's calls graph have been presented in the literature in recent years. In [6] for example, are presented three such algorithms on the automatic clustering of a Java program. The problem is that object-oriented programs have an inherently richer structure than those written in procedural languages, and so even medium sized programs produce large graphs. Because software clustering is a NP-hard problem, existing clustering tools are not able to process large graphs with weighted arcs and nodes, at the same time.

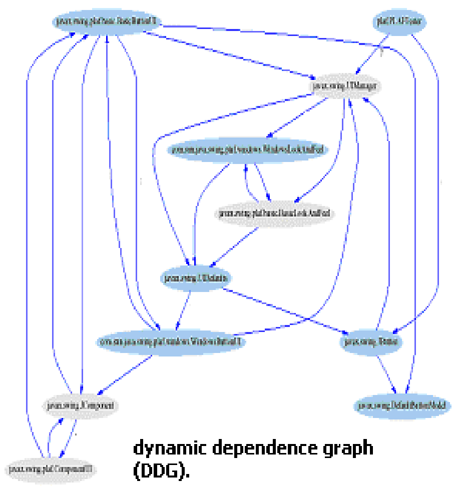


figure 3.

Understanding the dynamic structure of a system is helpful during software maintenance. The observation is that a number of algorithms for the static construction of an object's oriented call graph have been developed in the software reengineering literature in the recent years. The problem is that the source representation algorithms and tools must help software engineers extract also the dynamic structure of object oriented programs. Dynamic analysis is an open problem in software clustering. Our observation is that static analysis, using a tool such as Chava [11] reveals part of the complete design. It not reveals the relationships between the classes at runtime. Dynamic analysis can be used to complement static source code analysis, which may not provide all of the software information.

In [12] Juan Gargiulo and S. Mancoridis develop a tool, called Gadget, which try to extract the dynamic information of an Object Oriented Java program. A DDG (dynamic dependence graph) is created (figure 4), where nodes represent classes or objects and edges represent relations (i.e. method invocation) between two objects or between a static class and an object. The problem is that Gadget presents only the dynamic (running) relations and does not show the static relations such as inheritance relations.

III. Conclusion

Software supports many business, government, and social institutions. As the processes of these institutions change, so must change the software that supports them. Changing software systems that support complex processes can be quite difficult, as these systems can be large (e.g., thousands or even millions of lines of code) and dynamic. There is a need to develop sound software reengineering methods and tools to help software engineers understand and maintain large and complex systems so that they can modify the functionality or repair the known faults of these systems.

In this paper we try to summarize the characteristics of software reengineering activity: role, processes, goals, actual status, main problems and their solutions and to show the place of the software clustering process (figure 1) in the software-reengineering model.

In this paper we underline that the oriented graphs are commonly used in much software reengineering activities: automatic-clustering algorithms, in the fields of program slicing and regressing testing.

The observation is that a number of algorithms for the static construction of an object's oriented graph have been developed in the software reengineering literature in the recent years. But these clustering tools are not able to process large graphs with weighted arcs and nodes, at the same time. The problem is that even medium object-oriented programs produce such graphs because of polymorphism.

Dynamic analysis is an open problem in software reengineering. The source representation algorithms and tools must help software engineers extract also the dynamic structure of object oriented programs but there isn't enough experience in this field.

References

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

[2]. Spiros Mancoridis, Brian Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, Using Automatic Clustering to Produce High-Level System Organizations of Source Code, IEEE Proceedings of the 1998 International Workshop on Program Understanding (IWPC'98)

[3]. Spiros Mancoridis, Brian Mitchell, Y. Chen, and E. R. Gansner, Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures, IEEE Proceedings of the 1999 International Conference on Software Maintenance (ICSM'99)

[4]. Mary Jean Harrold, Cregg Rothermel, "A Coherent Family of Code-Based Graph Representations for Object-Oriented Software", Dep. Of Computer and Information Science, Ohao State University

[5]. Sinha, S., Harrold, M. J., and Rothermel, G. 2000. Interprocedural control dependence. Technical Report GIT-CC-00-17 (June), College of Computing, Georgia Institute of Technology.

[6]. Derek Rayside, Steve Reuss, Erik Hedges, and Kostas Kontogiannis. The effect of call graph construction algorithms for object-oriented programs on automatic clustering. In Margaret-Anne Storey, Anneliese von Mayrhauser, and Harald Gall, editors, IWPC'00, pages 191–200, Limerick, Ireland, June 2000.

[7]. Y. Chen, E. R. Gansner, and E. Koutsofios. A C++ Data Model Supporting Reachability Analysis and Dead Code Detection. In Proceedings of the European Conference on Software Engineer-ing/ Foundations of Software Engineering, 1997

[8]. B. Krishnamurthy. Practical Reusable Unix Software. John Wiley & Sons, Inc., New York, 1995.

[9]. S. Mancoridis, R. C. Holt, and M. W. Godfrey. A Program Understanding Environment based on the "Star" Approach to Tool Integration. In Proceedings of the Twenty Second ACM Computer Science Conference, pages 60–65, March 1994.

[10]. В.Т.Божикова, М.Н.Карова "Създаване, визуализация и операции на програмни структури", Proceedings of the Int'l Scientific Conference on Energy and Information Systems and Technologies, Vol.3., Bitola, pp. 813-819, June 7-8, 2001

[11]. Jeffrey Korn etc. "Chava: Reverse engineering and Tracking of Java Applets"

[12]. Spiros Mancoridis and Juan Gargiulo, "Gadget: A Tool for Extracting the Dynamic Structure of Java Programs", ACM/IEEE Proceedings of the 2001 International Conference on Software Engineering and Knowledge Engineering (SEKE'01)