An Approach to Measure the Cost of Program Restructuring

Violeta T.Bojikova[1] and Milena N.Karova[2]

Abstract - **Program restructuring tools make it easier to rewrite software, and so should be a key part of every software development environment. The problem is that software developers do not know how to restructure programs. This paper presents an approach for measuring the costs of program restructuring. The notion "distance" between the initial and the final software decisions was introduced**.

Keywords – **restructuring, design structure, program decomposition;**

I. INTRODUCTION

Changing the internal structure of a program without changing its behaviour is called restructuring [4]. Program restructuring is usually thought of as an activity for legacy software. However, it can also be used to develop new software and to improve his quality and is invaluable for software that is only a few years old but is being actively extended. Restructuring becomes unavoidable if the system is to survive its growing entropy. A way for restructuring programs is "decomposition". "Decomposition" is an operation to divide software structure or its components into two or more smallers program slices [1]. Inadequate structuring makes maintenance of the system expensive and difficult. Improving the structure of a program is a form of preventive maintenance that is often necessary when the system undergoes new releases. Low coupling between parts and high cohesion inside each part (module) are the key features of good software design. The reasonable distance between the original and the final structure improve the quality of the first structure and measure the restructuring cost.

Many Software Engineering - techniques use graph-oriented presentation in software, by means of different types of graphs G(X,U,Q) [2], [3]: control flow graphs, call graphs, program dependence graphs etc. A graph-based tool supporting module restructuring was created [3]. Part of the restructuring problems is the lack of tool support [4]. If program restructuring is properly supported, it becomes widely used. If it is widely used, it will improve the quality and decrease the cost of software. The created tool is based [3] on the graph description of software structure and applies a quick and easy method for program structure decomposition that uses famous graph algorithms for node visiting (node numeration) and is a combination from consecutive and iterative steps. The final solution ($\mathbf{r_{rac}}$–> $\mathbf{G^c}$) of the structure decomposition process satisfies the general requirements of the program structure design: high cohesion inside each part (MAXIMUM of links – inside) and low coupling between parts (MINIMUM of links (relationships) – outside). [3].

II. "DISTANCE" BETWEEN THE FIRST AND THE RATIONAL PROGRAM SOLUTION

This paper presents an approach for measuring the costs of structure decomposition (restructuring) application. For this reason the notion "distance" between two software decisions was introduced. Indications of these costs we can find in the number of the operations that can apply to the first software structure to produce the second structure. If we have structure $\mathbf{r_{apr}}$ -> $\mathbf{G=\{g_1,g_2,g_3\ldots\ g_M\}}$ [3], a new structure $\mathbf{r_{rac}}$ –> $\mathbf{G^c=\{g_1^c,g_2^c,g_3^c\ \ldots g_M^c\}}$ can be generated by applying an elementary operation "**opp**" on **G**. The four elementary operations that we apply [3] are: "Single move"- to move one object from a part $\mathbf{g_k}$ into another different part $\mathbf{g_i}$; "Part's Union" - to move one part $\mathbf{g_k}$ into another different part $\mathbf{g_i}$; "Block Move" - to move a "block" from part $\mathbf{g_k}$ into another different part $\mathbf{g_i}$; "Exchange" – changing the places of two objects, belonging to different parts.

The distance between the primary structure $\mathbf{r_{apr}}$ -> $\mathbf{G=\{g_1,g_2,g_3\ldots\ g_M\}}$ and the rational structure $\mathbf{r_{rac}}$ –> $\mathbf{G^c=\{g_1^c,g_2^c,g_3^c\ \ldots g_M^c\}}$ is the minimum number of elementary operations that can apply to the first structure to produce the second structure.

$$\mathbf{Distance(G, G^c)}=\mathrm{Min}(\mathbf{G} -^{opp1}> \mathbf{G^1} -^{opp2}> \mathbf{G^{2}} \ldots -^{oppi}> \mathbf{G^i} -^{oppi+1}> \mathbf{G^{i+1}} \ldots -^{oppc}> \mathbf{G^c}),$$

where "c" is natural number. Being a natural number, the structure distance is greater than or equal to zero:

$$\mathbf{Distance(G, G^c)} >= 0$$

The distance is zero when the structure **G** can be transformed into $\mathbf{G^c}$ with zero elementary operations, i.e. when **G** and $\mathbf{G^c}$ do not differ. Fig 1 shows the pseudocode of the algorithm that computes the distance between the two program structures. It is iterative algorithm beginning with the initial structure - **G** end ending when no operation is

[1]Violeta T.Bojikova is with the Department of Computer Science, Varna –Technical University, Bulgaria, e-mail:Bojikov@nat.bg;VBojikova@windmail.net
[2]Milena N. Karova is with the Department of Computer Science, Varna –Technical University, Bulgaria e-mail:mkarova@ms.ieee.bg

performed on the current structure - **G** (Until not Change, i.e. Change=False).

Procedure **Operate**(G, G^c,distance);
{Return(distance, G^c);}
{ G – initial structure, current structure,
G^c – final "rational" structure}
{distance - "**distance**"between G and G^c}
Begin
distance=0;
Repeat {1}
Change=False; {no operation is performed on the current structure - **G**}
if Ins_Segment(G…) then {Is it possible to execute "***Part Union***"}
Begin Change:=True; G=Operation(G); Inc(distance);End;
For SEG:=0..M-1 do {Cycle: For ∀ part SEG=0..(M-1) do}
begin
repeat {2}{ Cycle: For ∀ node I □SEG; I:=0 – >initial value}
repeat {3}{ Cycle: For ∀ part K=(SEG+1)..M}
repeat {4}{Cycle: For ∀ node J□K do **operations:**}
Is it possible to execute operations "***Moves***" from ***SEG to K;***
{**slivane, block, wmuk, razm** – counters for operations "**Part's unite", "Block Move", "Single move", "Exchange"**}
- Is it possible to execute operation "***Block Move***" from SEG (block around I□SEG) to K i.e. call procedure Ins_Block(Seg->K) **than** Change=True;G=Operation(G);Inc(distance) **else**
- Is it possible to execute operation "***Single Move***" from SEG (unit I□SEG) to K **than** -> Change=True;G=Operation(G); Inc(distance);

Is it possible to execute operations "***Moves***" from ***K to SEG;***
- Is it possible to execute operation "***Block Move***" from K (block around J□K) to SEG i.e. call procedure Ins_Block(K->SEGK) **than** Change=True; G=Operation(G); Inc(distance) else
- Is it possible to execute operation "***Single Move***" from K (unit J□K) to SEG **than** -> Change=True;G=Operation(G); Inc(distance);

Is it possible to execute operation "***Exchange***" of ***J□K with I□SEG ;***
- "**Exchange**" is possible than Change=True; G=Operation(G); Inc(distance);

G^c =G;
INC(J); until (J<=|g_k|){end repeat 1}
INC(K); until (K<=M){{end repeat 3}
INC(I); until (J<=|g_{seg}|){ {end repeat 2}
INC(**slivane**);
end; {For};
until not Change;{end repeat 1}
End;

Fig.1 Pseudocode of an algorithm that computes the distance between two software structures

Let us consider the structure partition $\mathbf{r_{apr}}$ **-> G={g_1,g_2,g_3,g_4} (**Example 1 – Fig.5**),** associated with 4 parts **g_1={x_6}, g_2= {x_7 ,x_8,x_3}, g_3={x_1,x_2}, g_4={x_4,x_5}**. The value of **k ("encapsulation violations" -** number of the links between all units **g_1,g_2,g_3,g_4)** is $\mathbf{k_{apr}}$ = 8. The value of $\mathbf{W_0}$ – "restrictive condition" (Fig.3) is 6. After restructuring **G** [3] we obtain the structure **G^c={g_1,g_2,g_3,g_4}** with the parts **g_1={x_3,x_6}, g_2={x_7,x_8, x_3}, g_3 = {x_1,x_2}, g_4 ={x_4, x_5}**. The value of **k ("encapsulation violations" -** number of the links between all units **g_1,g_2,g_3,g_4)** is $\mathbf{k_{apr}}$ = 8.

The distance between the two structures can be computed by applying the algorithm in Fig. 1. The distance between **G** and **G^c** for this example is equal to the sum of all operations that are applied on **G.** There number is 5 (Fig.4 and Table1).

Fig. 2. Graph G before decomposition

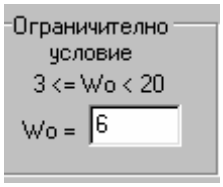

Fig. 3. The restrictive condition's form

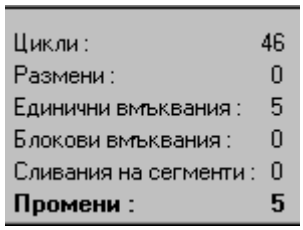

Fig. 4. The operation's counters

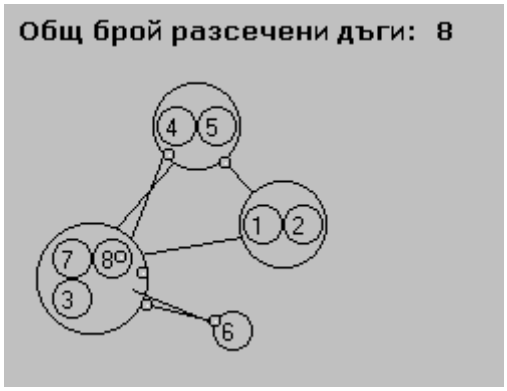

Fig.5 Initial Solution - $\mathbf{r_{apr}}$

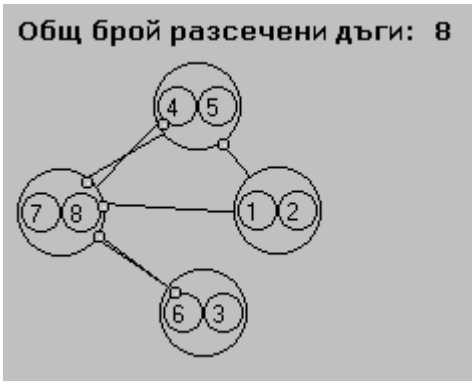

Fig.6 Rational Solution - $\mathbf{r_{rac}}$

Example1. The initial structure (Fig.5) - $\mathbf{r_{apr}}$-> $\mathbf{G=\{g_1,g_2,g_3,g_4\}}$ and the rational structure (Fig.6) $\mathbf{G^c=\{g_1,g_2,g_3,g_4\}}$ that non minimize the number of of the links between all (modules, subsystems) parts $\mathbf{g_1,g_2,g_3,g_4}$ but improve the solution (balance the weight of parts); The graph **G** before decomposition -> Fig.2;

TABLE 1

Operation - type	Number
Single move	5
Block Move	0
Part's union	0
Exchange	0
Total	5

The above notion of distance between software structures is appealing in the context of restructuring software system structure composed of several subsystems in different levels of abstractions. There are different kinds of structures: module structure, process structure, conceptual structure, uses structure; call structure, function structure, and class structure…

It is applicable for example in the context of module structure because every operation type can be used for module restructuring: to move one function from a module into another module, to move a module into another module, to exchange some functions from different modules. The "distance" can be considered a unit of measure for the restructuring effort paid when the decision is to reorganize the parts (slices) by moving some units (part's components) across parts.

III. CONCLUSIONS

This paper discusses the program-restructuring problem in the context of the restructuring costs. But the cost is not the only factor to examine during the structure decomposition: the presence of encapsulation violations and the level of modularization has to be evaluated. The level of decomposition M (number of parts) depends of the restrictive condition W_0. In conclusion, to get the whole picture of costs and benefits of a module restructuring intervention, the encapsulation – "k" and decomposition level - M should be compared with the initial ones and the costs of each restructuring alternative should be estimated. The proposed approach to estimation the costs for restructuring is combined with the method of structure decomposition and than watch for the value of the goal function and the restrictive condition. The algorithm was applied for a great number of structure graphs. The presented tool [3] was expanded and adapted for distance evaluating. Program restructuring tools make it easier to rewrite software, and so should be a key part of every software development environment. The distance between the initial ($\mathbf{r_{apr}}$ -> **G**) and the rational ($\mathbf{G^c}$) solution depends from the number of operations (cycle 4 – Fig1.) performed on the initial structure ($\mathbf{r_{apr}}$ -> **G**). Then the total number of these operations (the algorithm's complexity) is equal to:

$$\mathbf{C}=\square(\mathbf{n}_k*\square(\mathbf{n}_t)),\ k=0..(M-1),\ t=(k+1)..M$$

M-number of parts (level of decomposition)

$$N=|G|=\square(\mathbf{n}_t),\ t=1..M;$$

N - number of the graph's units (graph's nodes);

$\mathbf{n}_t$ - number of part's ($\mathbf{g_t}$) units (sugraph's nodes);

In the software practice there is many unsolved restructuring problems. One of them is that the "program restructuring is language dependant"[4]. The presented approach and the created tool are an attempt to stimulate progress in program restructuring area. Being graph-based, they are universal and language independent and can be used by the software architect at different level of program's abstractions.

REFERENCES

1. Len Bass, Paul Clemens "Software architecture in practice", Addison-Wesley Longman, 1998г.
2. David Binkley "The application of program slicing to regression testing", J.Information and Software Technology 40 (1998) 583-594
3. V.Bojikova, MKarova "Създаване, визуализация и операции на програмни структури", Int'l Scientific Conference on Energy and Information Systems and Technologies 2001 – Volume III, Bitola, June 7-8, 2001, 813 str.
4. Ralph E. Johnson "Program Restructuring", University of Illinois at Urbana-Champaign, Johnson@cs.uiuc.edu