Notice of Violation of IEEE Publication Principles

"Heuristic Change Propagation Model Encompassing Ripple Effect (HRE),"
by A.A. Al-Rababah, M.A. AlMaaitah, M.A. Al-Rababah,
in the Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications, and Computer Science, 2006. TCSET 2006., pp.28-30, Feb. 2006

After careful and considered review of the content and authorship of this paper by a duly constituted expert committee, this paper has been found to be in violation of IEEE's Publication Principles.

This paper was found to have been previously published. The lead author resubmitted the paper without crediting original coauthor and adding new coauthors. The conference publication chair was not informed of the previous publication nor of the change in authorship.

Due to the nature of this violation, reasonable effort should be made to remove all past references to this paper, and future references should be made to the following article:

"Heuristic Change Propagation Model Encompassing Ripple Effect (HRE),"
by M.Al-Shareef, A. Al-Rababah,
in Asian Journal of Information Technology 4 (10): 950-953, 2005 Grace Publications

Heuristic Change Propagation Model Encompassing Ripple Effect (HRE)

Ahmad A. Al- Rababah, Mohammad A. AlMaaitah, Mohamad A. Al- Rababah

Abstract - **Software maintenance is one of the major concerns of software developers and industries. Maintenance highly depends on the understanding of the nature of the system and the relation between modules. However, these relations rely on many factors and all of them can't be determined and studied clearly. One of the most important issues in software maintenance is to propagate the changes when a module is modified within the system that is to determine the modules which are affected from the change and determine the next module to trace from the set of affected modules. For this, the modules within the system are represented using a directed graph. When a module is modified a heuristic function will be used to determine the next module to be modified.**

The change studied in this research is the modification within modules not the insertion or deletion of modules.

Keywords - **Change propagation, Ripple effect, Software maintenance, Heuristic function, Change propagation**

I. INTRODUCTION

Software maintenance is the general process of changing a system after it has been delivered. The changes made to the software may be simple changes to correct coding errors, or may be changes to correct design errors or significant enhancement to correct specification errors or accommodate new requirements [1]

Whatever the cause of the change! The change will propagate through system modules. To cope with this, the relations and dependencies between modules must be understood.

When a programmer makes a change in software he starts by changing a specific module of the software, which may cause the module to no longer fit with other modules in the system.

This is because it may not provide what other modules require, or it may require different services from the modules it depend on. The dependencies which do not_satisfy the provided relationships are called inconsistent dependencies and they may arise whenever a change is made in the system.

The change propagation process keeps track of the inconsistencies and the locations where the secondary change are to be made. The process in which the change spreads through the software is sometime called the ripple effect of the change [2]

Many problems maybe faced while maintaining the software, such problems maybe arise because most computer programs are difficult and expensive to maintain or software changes are poorly designed and implemented also the repair and enhancement of software often injects new bugs that must later be repaired [3], the developer may go in a loop because the first changed module may required to change again and again because other module that it depend on is changed too. A cause of such problems will be the waste of time and system resources.

How the change is propagating through the system software is studied in many researches. In [4] several heuristics to predict the change propagation was propose. In [2][5] Vaclav gave a model for change propagation and discuss two processes of propagation of ripple effect in object oriented systems. Other researches involved in studying the impact analysis to determine the potential effects of change on the software systems as in [6][7], also many research studied the estimation of maintenance cost taking into account the ripple effect [8] ,[9], [10]

Our goal is to let the change propagate in a consistent way by determining the next module to be traced from a set of modules that may be affected from a change depending on a heuristic function calculated by costing the arrow of a directed graph taking into account the ripple effect.

In the next section the proposed model for change propagation is explained. In section 3 a small case study is discussed to let the idea of the HRE model be clear. The conclusion and future work in shown in section 4.

II. PROPOSED MODEL

In this research a new way to follow a propagation of the change will be constructed. The proposed model will guide the developer to determine the path to follow in tracing and updated the modules that are affected during the maintenance. This protocol will save the time and resources in the maintenance stage.

II.A GENERAL DESCRIPTION

The system will be represented by a directed graph, where each node in the graph represents a module, and each directed arrow represent a relation between two modules.

When a module changed a set of modules will be affected and need to be traced in order to keep consistency between system modules. A specific path will be followed to handle the change propagation. To determine this path a set of all affected modules will be generated according the module neighbors. A module is considered to be a neighbor to other module if there is an arrow between the two modules. Each element in the set will be given two values, number of marks plus a cost. Then a heuristic function will be calculated to determine the module that will be select from the set and handled. After handling the module the set will need to be updated then the steps will be repeated to select another module to be handled. Figure 1 show these steps.

*Ahmad A. Al- Rababah-Faculty of IT, Al-Ahliyyah Amman University Post Office, Amman-19328, Jordan.
E-mail : Ahd_68@yahoo.com

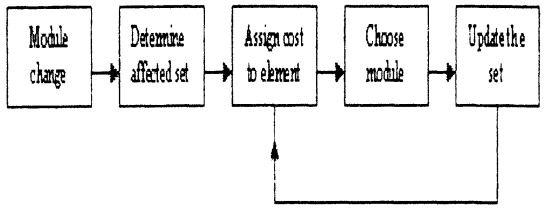

Fig.1: Steps to follow to propagate the change

II.B HEURISTIC MODEL ENCOMPASSING RIPPLE EFFECT (HRE)

To keep track on the ripple effect, and to be sure that the change propagation in a way that keep the dependency between module consistent, a model is constructed to choose the module which will be traced from a set of affected modules. This model depends on the graph representation and the heuristic function.

The following steps show how this model works:

1- Define the set of affected modules, the set will look like:

{(module$_1$, mark$_1$, cost1), (module2, mark$_2$, cost2)... (module$_i$, mark$_i$, cost$_i$) .....}

Where:

module$_i$ The ith module in the set mark$_i$ Number of marks assign to the ith module, initially set to 1 cost$_i$ The cost assign to the ith module, initially set to 0

These modules in the set will be in the module's neighbor. Module neighbors set in the set of all modules with have direct arrow to the changed module. For example if module C in figure 2 is changed then the module neighbors set will contain A, B, D, M. All these modules will be marked as 1, and have a cost equal to zero.

2- The cost of the module in the set will be changed according to a heuristic function. The directed arrow between two node in the graph mean that there a relation between these two node. The Relation will be association, aggregation, inheritance and/or invocation or any other king. Direction of arrow reflects dependency that is which module depends on which. For example in figure 2 the arrow between M to C mean that module M invoke module C.

For further information about relation and dependencies between system modules refer to [1],[11]

The heuristic function that will calculate the cost depend on:

- Number of ongoing arrows to the module (I): which mean that many modules depend on this module, and any change to this module will affect many modules. For example in figure 2 I for module A will be 4, that is any change to module A will affect four modules : H, B, C, N . This number I actually give an importance to module A .

- Number of outgoing arrow from the module (O): which mean that the module depends on many modules. For example in figure 2 O for module B will be 2, that is any change to either module H or A will affect module B.

- Number of referring of module from the changed module (R): assume that Module C invoke A three times, this mean that module C highly depend on Module A.

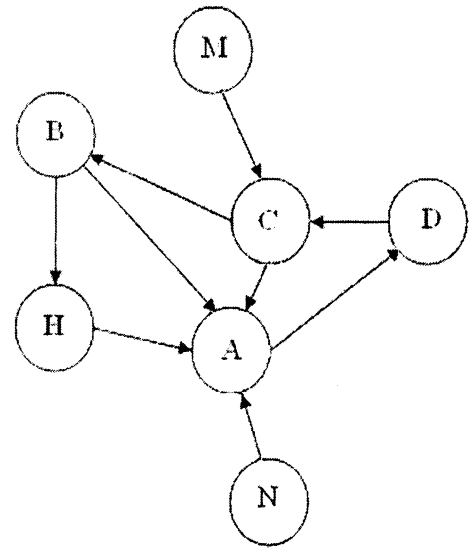

Fig.2: directed graph reflect the dependency between modules

- Number of times the module is marked (M): Every time a module is changed, a number of modules will be marked. When a module mark n times this mean is has been affected by n changed modules.

Then the heuristic function F(x) will be calculated as:

$$F(x) = I + O + R + M \quad \ldots\ldots\ldots 1$$

This heuristic function will be calculated for every element in the set for every change, this value will replace any existing value of the cost.

3- Pick the module from the set which have the largest cost. This module will be removed from the set then traced to determine if it is really affected by the change. If yes it will be changed and the set will need to be updated. If it does not have to be change then the element of the set will not be affected by this module and these is no need to recalculate the cost and mark.

4- Update the set if needed, the set will need to be updated after every change to any module with restriction that the update will be only to mark or adding new element to the set, but no deletion is allowed. While updating the set if the module already exists in the set then increase the mark by one. If the element is not in the list then add it and set the cost to zero and the mark to one. The cost will be reset to zero for all element because it will be recalculated.

The steps from 2 to 4 will be repeated until the set be empty. Note that if no change to the module in step 4, then the cost will remain the same.

III. CASE STUDY

To illustrate HRE model an example will be followed step by step

Figure 3 is really an expansion to figure 2; it shows a graph which represents seven modules in a system.

Assume that a change is done on Module C, and then the following iteration will tale place:

Iteration 1:

1- The affected set will be constructed: {(A,0,1),(B,0,1),(D,0,1) ,(M,0,1) }

2- The cost for every element in the set will be calculated

{(A,5,1),(B,3,1),(D,0,2), (M,0,1) }

3- Remove Module A from the set and trace it, assume that it needs a change.

4- Update the set {(B,0,2),(D,0,2),(M,0,1), (H,0,1), (C,0,1), (N,0,1) }

Iteration 2:

1- The cost will be calculated for all elements in the set, note that while recalculate the cost I and O will remain the same in this example, but actually it may change during the system maintenance phase.

{(B,5,2),(D,4,2),(M,1,1) ,(H,2,1) ,(C,4,1) ,(N,1,1) }

2- Trace module B. Again assume that it needs a change.

3- Update the set:

{(D,0,2),(M,0,1),(H,0,2) ,(C,0,2) ,(N,0,1),(A,0,1) }

Iteration 3:

1- Recalculate the cost

{(D,4,2),(M,1,1) ,(H,4,2) ,(C,6,2) ,(N,1,1),(A,5,1) }

2- Trace module C. Change.

3- Update the set:

{(D,0,3),(M,0,2) ,(H,0,2) ,(N,0,1),(A,0,2),(B,0,1) }

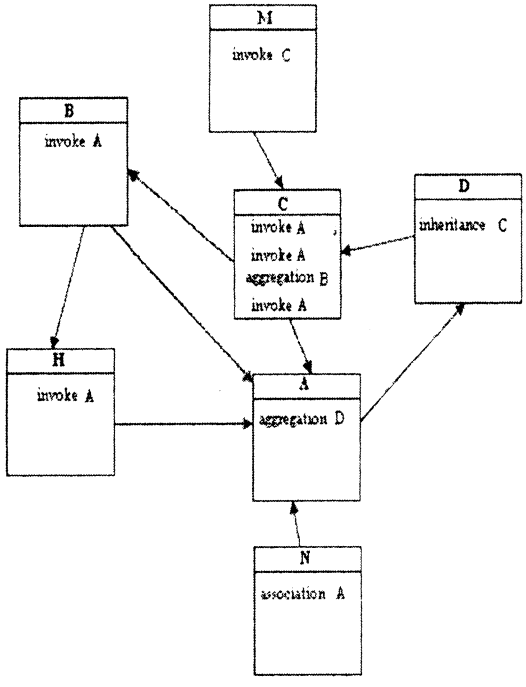

Fig.3: Directed graph reflect the dependency between modules

Iteration 4:

1- Recalculate the cost

{(D,5,3),(M,3,2) ,(H,4,2) ,(N,2,1),(A,7,2),(B,4,1) }

2- Trace module A. No need to Change.

Iteration 5:

{(D,5,3),(M,3,2) ,(H,4,2) ,(N,2,1),(B,4,1) }

Trace module D. No need to Change.

Iteration 6:

{(M,3,2) ,(H,4,2) ,(N,2,1),(B,4,1) }

Choose between module H and module B. Trace module H. No need to Change.

Iteration 7:

{(M,3,2) ,(N,2,1),(B,4,1) }

Trace module B. No need to Change.

Iteration 8:

{(M,3,2) ,(N,2,1) }

Trace module M. No need to Change.

Iteration 9:

{(N,2,1) }

Trace module N. No need to Change.

IV. Conclusion

Propagation of the changes during maintenance in software systems has been gaining a lot of attention to the researchers. In this work, we define a new protocol that can handle in a systematic way and study the change propagation taking into account the ripple effect, without wasting the software system resources.

This model can be applied to any kind of similar problems including object-oriented system, or procedural systems, and also to databases.

The technique has been presented with a simple case-study showing a variety of relations between system modules and found proven to have no loop and no waste of resources.

References

[1] I. Sommerville, "Software engineering", Addison Wesley, Seventh Edition, 2004

[2] V. Rajlich, "A Model for Change propagation Based on Graph Rewriting", International Conference on Software Maintenance (ICSM '97), IEEE Computer Society 1997, ISBN 0-8186-8013-X, 1997, pp. 84-91

[3] Software maintenance power point slide (wk07se6Maint), www.csse.monash.edu.au/courseware/cse2201/html/lectures_files/wk07se6Maint.ppt

[4] A. Hassan and R. Holt, "Predicting Change Propagation in Software Systems", 20th International Conference on Software Maintenance (ICSM 2004), IEEE Computer Society 2004, ISBN 0-7695-2213-0, 2004, pp. 284-293

[5] V. Rajlich, "Propagation of Change in Object-Oriented Programs", ESEC/FSE'97 Workshop on Object-Oriented Reengineering Zurich, 1997

[6] J. Law and G. Rothermel, "Incremental Dynamic Impact Analysis for Evolving Software Systems", 14th International Symposium on Software Reliability Engineering (ISSRE 2003), IEEE Computer Society 2003, ISBN 0-7695-2007-3 , 2003, pp. 430-441.

[7] H. Kabaili, R.K. Keller, and F. Lustman, "A Change Impact Model Encompassing Ripple Effect and Regression Testing", Proceedings of the Fifth International Workshop on Quantitative Approaches in Object-Oriented Software Engineering , 2001, pp. 25-33

[8] F. Xia and P. Srikanth, "A Change Impact Dependency Measure for Predicting the Maintainability of the Source Code", 28th Annual International Computer Software and Applications Conference (COMPSAC'04), 2004, pp.22-23

[9] M. Polo, M. Piattini, F. Ruiz, "Using Code Metrics to Predict Maintenance of Legacy Programs, International Conference on Software Maintenance (ICSM 2001)", IEEE Computer Society, ISBN 0-7695-1189-9, 2001, pp. 202-208

[10] S. Black, "Computation of Ripple Effect Measures For Software", PhD thesis, South Bank University, 103 Borough Road, London SE1 0AA, 2001

[11] H. M. Deitel and P. J. Deitel, "Java How to Program", Prentice Hall, Fifth edition, 2003