Asian Journal of Information Technology 4 (10): 950-953, 2005
© Grace Publications, 2005

Heuristic Change Propagation Model Encompassing Ripple Effect (HRE)

[1]Mai Al-Shareef and [2]Ahmad Al- Rababah
[1] Amman Arab University for Graduate Studies
[2]Faculty of information technology , Al-Ahliyya Amman University, Amman, Jordan

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 study is the modification within modules not the insertion or deletion of modules.

Key words: Change propagation, ripple effect, software maintenance, heuristic function, change propagation

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. Hassan and Holt proposed[4] several heuristics to predict the change propagation Rajlish[2] and Vaclav[5] gave a model for change propagation and discuss two processes of propagation of ripple effect in object oriented systems. Other studies involved in studying the impact analysis to determine the potential effects of change on the software systems as in Law and Rothermel[6] and Kabaili *et al.*[7], also many research studied the estimation of maintenance cost taking into account the ripple effect[8-10]

Aim of our present study 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.

PROPOSED MODEL

In this study a new way to follow a propagation of the change will be constructed. The proposed model will

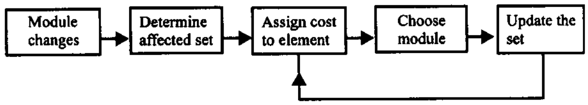

Fig. 1: Steps to follow to propagate the change

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.

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 (Fig.1).

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:

- Define the set of affected modules, the set will look like:
  {(module$_1$, mark , cost), (module ,$_2$mark ,$_2$cost $_2$). (module$_i$, mark$_i$, cost$_i$) ......}

Where:

module$_i$;	The I th 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 Fig. 2 is changed then the module neighbors set will contain A, B, D and M. All these modules will be marked as 1, and have a cost equal to zero.

- 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 Fig. 2 the arrow between M to C mean that module M invoke module C.

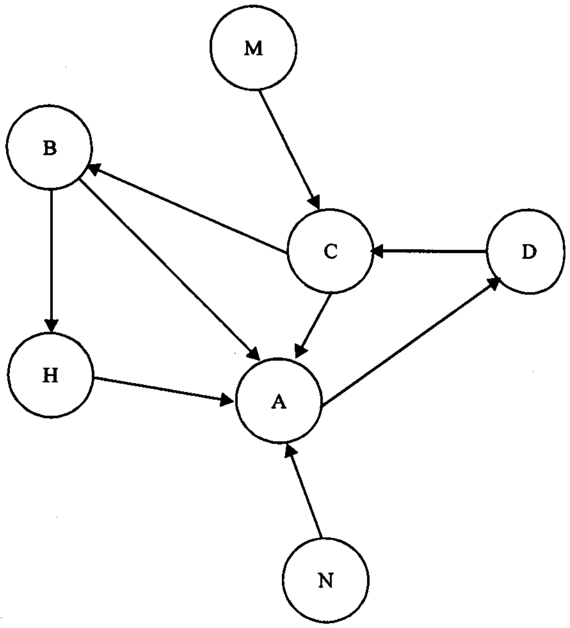

Fig. 2: Directed graph reflect the dependency between modules

For further information about relation and dependencies between system modules refer to Sommenille[1] and Dietel and dietel[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 Fig.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 Fig. 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.

- 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:

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

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.

- 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.
- 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.

RESULTS AND DISCUSSION

To illustrate HRE model an example will be followed step by step Fig. 3 is really an expansion to Fig. 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:

- The affected set will be constructed: {(A,0,1),(B,0,1),(D,0,1) ,(M,0,1) }
- The cost for every element in the set will be calculated {(A,5,1),(B,3,1),(D,0,2), (M,0,1) }
- Remove Module A from the set and trace it, assume that it needs a change.
- Update the set {(B,0,2),(D,0,2),(M,0,1),(H,0,1),(C,0,1) ,(N,0,1) }

Iteration 2:

- 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) }
- Trace module B. Again assume that it needs a change.
- Update the set: {(D,0,2),(M,0,1) ,(H,0,2) ,(C,0,2) ,(N,0,1),(A,0,1) }

Iteration 3:

- Recalculate the cost {(D,4,2),(M,1,1) ,(H,4,2) ,(C,6,2) ,(N,1,1),(A,5,1) }
- Trace module C. Change.
- Update the set: {(D,0,3),(M,0,2) ,(H,0,2) ,(N,0,1),(A,0,2),(B,0,1) }

Iteration 4:

- Recalculate the cost {(D,5,3),(M,3,2) ,(H,4,2) ,(N,2,1),(A,7,2),(B,4,1) }
- Trace module A. No need to Change.

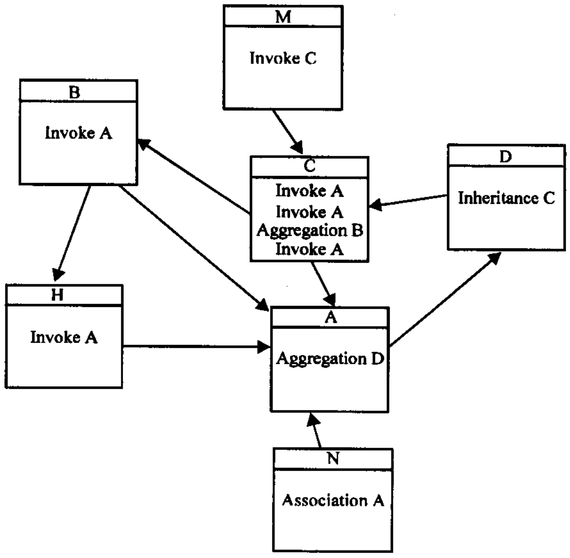

Fig.3: Directed graph reflect the dependency between modules

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.

CONCLUSIONS

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. Sommerville, I., 2004. Software Engineering, Addison Wesley, 7th Edn.
2. Rajlich, V., 1997. A model for change propagation based on graph rewriting, international conference on software maintenance (ICSM '97). IEEE Computer Society, ISBN 0-8186-8013-X, pp: 84-91
3. Software Maintenance Power Point Slide (wk07se6Maint),www.csse.monash.edu.au/courseware/cse2201/html/lectures files/wk07se6Maint.ppt
4. Hassan, A. and R. Holt, 2004. Predicting change propagation in software systems: 20th international conference on software maintenance (Icsm 2004). IEEE, Computer Society, ISBN 0-7695-2213-0, pp: 284-293.
5. Rajlich,V., 1997. Propagation of change in object-oriented programs. ESEC/FSE'97 Workshop on Object-Oriented Reengineering Zurich.
6. Law, J. and G. Rothermel, 2003. Incremental dynamic impact analysis for evolving software systems, 14th international symposium on software reliability engineering (ISSRE 2003). IEEE, Computer Society, ISBN 0-7695-2007-3 , pp: 430-441.
7. Kabaili, H., R.K. Keller and F. Lustman, 2001. A change impact model encompassing ripple effect and regression testing. Proceedings of the Fifth International Workshop on Quantitative Approaches in Object-Oriented Software Engineering , pp: 25-33
8. Xia, F. and P. Srikanth, 2004. A change impact dependency measure for predicting the maintainability of the source code, 28th Annual International Computer Software and Applications Conference (COMPSAC'04), pp: 22-23.
9. Polo, M., M. Piattini and F. Ruiz, 2001. Using code metrics to predict maintenance of legacy programs, International conference on Software Maintenance (ICSM 2001). IEEE, Computer Society, ISBN 0-7695-1189-9, pp: 202-208.
10. Black, S., 2001. Computation of ripple effect measures for software. Ph.D Thesis, South Bank University, 103 Borough Road, London SE1 0AA.
11. Deitel, H.M. and P.J. Deitel, 2003. Java How to Program. Prentice Hall, 5th Edn.