結構化軟體系統之使用模式雛型的建構研究

An Environmental Prototype of Usage Modeling for Structured Software Systems

Wen-Yan Chang(張文彥)*, Jyh-Yeuan Hsiao(蕭智遠)*, Wen-Kui Chang(張文貴)*, & Bai-Jiun Tu(涂百鈞)**

*Dept. of Computer & Information Sciences, Tung Hai University, Taichung, Taiwan

**Dept. of Information Management, National Taichung Commercial College, Taiwan

摘要

許多文獻已證明，統計使用測試對於軟體系統的品質驗證，非常實際而有效。但是，要執行統計使用測試，首先必須建立軟體使用模式。在本研究中，將針對結構化軟體系統，探討馬可夫鏈使用模式之建立，以便執行使用測試，做為估計軟體可靠度的依據。

在本文建構的使用模式雛型中，不但可以決定預期的使用路徑，更可以計算出每一節點的預期使用頻率，並提供這些資訊給軟體工程師，以協助系統的開發與測試，增進軟體產品的品質。本系統最大的好處是，能快速建構客戶所要求系統之大綱結構的雛形，即使其規格是模糊而不夠明確時，亦可透過本系統的修改功能，簡易地調整其大綱結構。

關鍵詞: 軟體品質驗證, 統計使用測試, 軟體使用模式, 馬可夫鏈

Abstract

Literatures demonstrate that statistical usage testing has been a practical and effective tool for software quality certification. Before performing statistical usage testing, a software usage model has to be established in advance. In this research, the building of a Markov usage model is investigated for structured software systems, which may be used for software testing to estimate software reliability.

The presented prototype for usage modeling may not only determine the intended usage distribution, but also compute intended usage frequency for each node in the model. Such information will aid software engineers with the critical functions during system development or in software testing in order to enhance product quality. The most advantage of the proposed system lies the fact it may build usage hierarchy rapidly. In case of fuzzy or unclear requirements provided by customers, this system is capable of adjusting the usage hierarchy effortlessly by the built-in editing facility, too.

Keywords: software quality certification, statistical usage testing, software usage model, Markov chain

A. Introduction

Statistical usage testing [11] has been successfully used for software quality certification. To perform the usage testing, a Markov usage model [13,14] of describing usage specifications has to be built first. However, it is usually difficult to build completely a Markov usage model, especially for a large and complex system. This paper will explore the building of a Markov usage model for the structured software system by an environmental prototype.

By facilitating system developer, the proposed prototype may formulate rapidly the framework of user's desired system structure, and then automatically record usage distribution among the structure. In addition, the suggested prototype possesses the capability of editing structure. Thus, it may be applied to the developing environment under which the requirements are usually uncertain at the beginning.

B. Software Quality Certification

a) Statistical Usage Testing

Recent research on related literature justifies model-based statistical usage testing [2,11,12,15] has been justified and widely applied to software quality certification [4,8,9].

Conceptually, statistical usage testing is a somewhat functional testing in a statistical aspect. It focuses on the external system behavior, not the internals of design and implementation. It is a formal process with objectivity and completeness in test cases selection

The rationale of statistical usage testing lies in the fact that the test environment is statistically representative of the real operating case, and the most frequently used operations receive the most testing. Thus, the failures occurred most frequently in practical use will be found early during the test cycle.

b) Software Usage Model

A software usage model is the kernel part in a usage testing process. Expectedly, the model is to characterize

all operational uses of a software system. An operational use [6] is a skeleton for the intended use of the software in an intended environment. Thus, all possible operational uses of a software system constitute a population. If a usage sample of test cases is drawn statistically from the usage population, performance on this sample may then be analyzed for the evaluation of software quality. In 1993, Whittaker suggests [13] that software testing is rather suitable to be treated as a stochastic process and, its usage be modeled by a finite state, discrete parameter, time homogeneous, irreducible Markov chain.

c) Usage Modeling

Study of the process of building usage models from system specifications has been an ongoing area of research. For instance, research [1,12] employs mathematical programming technique to generate usage models that satisfy a given set of software usage constraints while optimizing desired test objectives.

Although a Markov model [14] is adequate to manifest possible usage pattern, it is usually difficult to specify completely all-possible transition states in a Markov chain due to the complexity of a software system, in particular, for those of newly developed or highly complicated systems.

This research investigates to construct an organization chart as a system structure from software specification. Through the established organization chart, various users may manually simulate their future intended use patterns. Consequently, expected usage frequency may be computed immediately, and a software usage model is then completely built and ready for further quality certification.

C. System Mechanism

In essence, there are four principle parts included in the Environmental Usage Modeler (EUM) system, as shown in Fig. 1:

a) Organizer

To build initially a new system organization chart, EUM will provide a tree structure with one root and two children so that a user may easily establish his desired chart. Alternatively, once a system organization chart has been already built, the user loads the chart and then modifies it as needed through the system mechanism.

Occasionally, a system organization chart may be too complicate to be browsed as a whole on the screen. By choosing certain node to show only the sub-organization chart, EUM will display its ancestor and all its children. This feature will make user to view flexibly his chart structure while he is editing the system organization chart.

b) Usage Identifier

The main function of the Usage Identifier is to build usage hierarchy chart. Like sub-organization chart, a usage hierarchy chart is essentially a sub-chart of system organization chart. Inside of a usage hierarchy chart, however, it is not allowed to perform any node operation such as addition, deletion, or modification. Instead, usage distribution may be provided to some selected nodes for tricking usage history.

c) Usage Distribution Simulator

Actually usage distribution specifies one complete path which denote possible usage occurred on these connected nodes. For all nodes specified in the usage distribution, EUM will automatically increment their pass frequency count if it is needed. Accordingly, usage frequency may be collected immediately for all nodes.

d) Usage Profiler

To provide all usage distribution before running a statistical usage testing, the Usage Profiler will collect all frequency history, and maintain a usage profile that is ready for building a usage model.

D. Data structure

For data structure, an N-way tree structure is employed in this research, as shown in the Fig. 2.

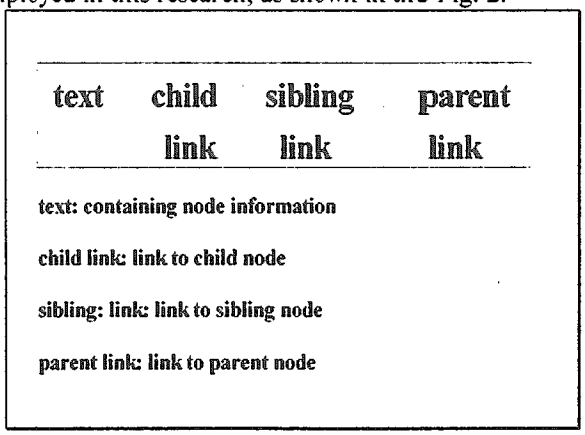

Fig. 2 Implemented tree structure

Furthermore, EUM is implemented by the JAVA language [3,7]. Class declaration for a node in an N-way tree is specified as in Fig.3.

E. Design issues

In designing the environmental system, there are several considerations to be studied:

a) Chart layout

In general, the tree structure built by a user is not necessary to be complete [5]. Besides, node operation

```
public class TreeNode
{ TreeNode sibling;     //use to link its sibling
     TreeNode child;       //use to link its child
     TreeNode parent;     //use to link its parent
     String    key;        //to denote the desired node
     String    note;       //comment for the node
     int       level = 1; // use to save the level of this
node
     int       numberOfChildren=0;
     int       x,y;        // to denote node position
public TreeNode(String init_key , String init_note )
{ //constructors
      key=init_key;
      note=init_note; } }
```

Fig. 3 Class declaration for a node

may be applied arbitrarily to any existing node. Accordingly, the resultant tree structure will be unbalanced and the chart layout may be in the ill appearance. Some techniques are, thus, needed to enhance demonstration of the various chart layouts. Two different approaches are made use in the suggested system.

1) Static mode

While constructing a usage structure, the number of nodes may be increased so rapidly that the whole structure cannot be displayed in one screen, since some nodes may be beyond the screen boundary. With the static mode, node size in a chart will be narrowed down in order to show the complete snapshot of a structure chart.

In performing the static mode, a screen is first split horizontally into several bands, which represent different levels. Meanwhile, each band is further sliced vertically to many cells, like the chess lattice. Through the adjusting mechanism for the primitive lattice, size of all nodes may then be reflected with respect of the number of nodes.

On the other hand, static mode is capable of enlarging any specified node for browsing the detailed information, as the call graph used in Logiscope [10] by VERILOG.

Advantages of using static mode lie from the following facts:

(1) It provides clearly user the complete outlook of a chart, and relationship between nodes as well,

(2) It gives high attention on nodes alone to simplify chart presentation, without considering the scroll for the chart.

2) Dynamic mode

As in static mode for the layout design, dynamic mode also separates horizontally a full screen into several levels. What is different, however, dynamic mode keeps the size of each lattice fixed, whose space is sufficient to contain any node. In addition, the displayed location of a node on the screen will be determined by the corresponding index of the leaf node. Without adjusting the size of a node, the screen may be scrolled to browse further other parts of a tree organization as the number of nodes is increasing.

b) Node location determination

Before getting the detailed information of an interested node, its location should first be specified. On determining the relative node position on a screen, two different approaches are considered:

1) Iterative searching

When clicking at any point on the screen by a user, this approach starts from the root node to check whether the interested point is in the tree organization chart. If it goes through to all terminal nodes, it implies that the specified point does not match any node on the tree structure.

2) Fitting searching

Following the leveling mechanism of chart layout, this approach will first use the coordinate for the clicking point and then map its Y-coordinate to the corresponding level. Once the desired level is located, its X- coordinate is further used to determine fitness of a node on the tree at the same level.

For comparison, the first approach, in general, will take a little more execution time. In particular, the worst complexity of the first approach will be O(n) if it is a skew tree. For the second approach, however, its worst complexity will be less, even if all nodes are at the same level. Accordingly, the second approach is employed in this research for saving effort on node searching.

c) Parent node deletion

If a parent node is deleted during chart modification, in general there are two alternatives to handle its children nodes:

1) To uplift children nodes

After deleting a parent node, its all children nodes are uplifted one level.

2) To delete children nodes

All children nodes are taken away followed the parent node is deleted.

In this research, the second alternative is made use of. The rationale of this choice lies the observation that the relationships between a node and its children nodes are

fully dependent. Hence, if a node is removed from an organization chart in EUM, all its children nodes will not necessarily exist any more. For instance, in any software system with the editing function at its menu bar, if the editing function is removed for safety consideration, then its accompanying subfunctions such as copy, cut, paste, etc. are definitely meaningless and have to be removed, too.

F. Illustration and discussion

a) Illustrated example

For demonstration, system organization chart is constructed for an application by the proposed EUM. As shown in Fig. 4, it includes 17 nodes, each of which is labeled by a distinct identifier. It is noted that there is no any correlation between the order of the labeled identifier and building sequence for nodes.

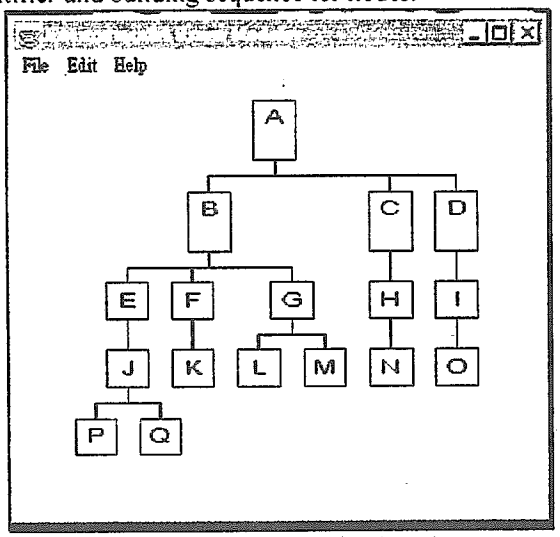

Fig. 4 Complete system organization chart

Suppose that the system organization chart is too complicate to display in one screen and we are interested in the part tree consisting of Node-J, EUM will present the sub-organization chart with its ancestor nodes (A, B, E) and children nodes (P, Q), as illustrated in Fig. 5.

On a usage hierarchy chart, a specified usage distribution may be determined by scanning straightly a desired usage path. As an illustration, as we go through the nodes A->B->E->J->Q, a usage distribution is established accordingly.

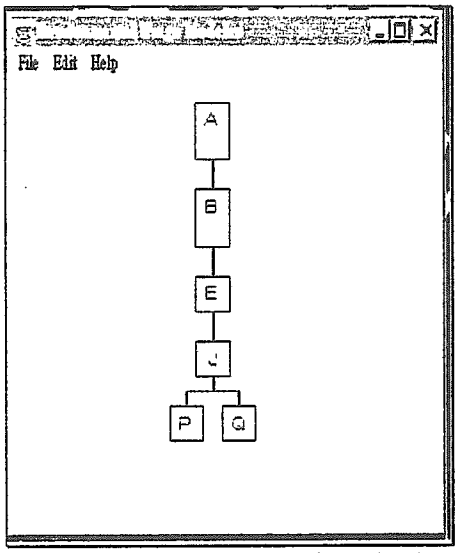

Fig. 5 Sub-organization chart after selecting J-node

Once all usage distributions are established, EUM will perform computation of usage frequency, and provide a resultant summary with a usage profile as given in Table 1. The usage model for this illustrated example is shown in Table 2.

A	22	G	5	M	3
B	13	H	7	N	7
C	7	I	2	O	2
D	2	J	3	P	1
E	3	K	5	Q	2
F	5	L	2		

Table1.Usage profile for the illustrated example

b) Discussion

In this research, there are still the following problems to be further investigated:

1) Expansion on the number of nodes included

As stated, dynamic mode for displaying the chart nodes over the screen is via fixing the node size. Thus, it may happen difficulties for redesigning chart layout as the number of the added nodes is beyond a limit.

2) Chart excess over the screen

Suppose that a node is inserted and its resultant node position exceeds the screen range, the following additional steps will be performed:

(1) To enlarge the size of all node objects,

(2) To scroll the considered chart in an adequate position,

(3) To redraw the whole organization chart for better appearance.

Consequently, the processed time for chart layout will take longer when node insertion is beyond the screen boundary.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
A	0.59	0.31	0.1													
B				0.24	0.38	0.38										
C							1									
D								1								
E									1							
F										1						
G											0.4	0.6				
H													1			
I														1		
J															0.33	0.67

Table 2. Usage model for the illustrated example

G. Conclusion

It has been shown that statistical usage testing based on a usage model provides a cost-effective effort over software testing. Most importantly, the testing result may support highly confident certification of software reliability measures in operational use by statistical inferences. In practice, usage modeling is the first task to perform statistical usage testing.

This paper investigates the possibility of establishing and modifying visually a Markov usage model for structured software systems. For a considered software system, the implemented modeling environment establishes first an organization chart. Then it may construct a usage path by specifying the practical usage pattern over the organization chart and generate a usage hierarchy chart immediately, which becomes a baseline for usage modeling.

The instrument mechanism may be applied not only at the specification stage, but also during design phase. Based upon the accumulated usage frequency, software designer may understand in advance which functions will be used more frequently. So that he has to pay much effort over these functions and the resultant software product will be with much higher quality accordingly.

In order to make use of statistical usage testing more practicable and applicable, further research will continue in studying the proposed prototype for extending the larger usage hierarchy chart to more general cases.

References

[1] Chang, Wen-Kui, "A Quadratic Programming Approach to Usage Modeling for Software Reliability Certification," Tunghai Journal, Vol.38, pp. 65-78, July 1997.

[2] Dyer, M., The Cleanroom Approach to Quality Software Development, John Wiley & Sons, 1992.

[3] Flanagan, D., *JAVA in a Nutshell,* O'Reilly & Associates Inc., 1996.

[4] Hamlet, D., "Are We Testing for True Reliability?" IEEE Software, pp. 21-27, July 1992.

[5] Horowitz, E., Sahni, S, Mehta, D., *Fundamentals of Data Structure in C++*, W.H. Freeman and Company, 1995.

[6] Musa, J.D., "Operational profiles in Software Reliability Engineering," IEEE Software, March 1993, pp. 14-32.

[7] Naughton, P., *The JAVA Handbook,* McGraw-Hill Inc., 1996.

[8] Poore, J.H. and H.D. Mills and D. Mutchler, "Planning and Certifying Software System Reliability," IEEE Software, Jan. 1993, pp. 88-99.

[9] Poore, J.H. and C.J. Trammell. *Cleanroom Software Engineering: A Reader.* Blackwell Publishers: Oxford, England, 1996.

[10] VERILOG Inc., LOGISCOPE Reference Manual, 1993.8.

[11] Walton, G. H., J. H. Poore and C. J. Trammell. "Statistical Testing of Software Based on a Usage Model," Software Practice and Experience, January 1995, vol. 25(1), pp. 97-108.

[12] Walton, G.H., *Optimizing Software Usage Models*, Ph.D. Dissertation, Department of Computer Science, University of Tennessee, May 1995.

[13] Whittaker, J. A. and J.H. Poore. "Markov Analysis of Software Specifications," ACM Transactions on Software Engineering and Methodology, January 1993, vol. 2(1), pp. 93-106.

[14] Whittaker, J. A. and M. G. Thomason. "A Markov Chain Model for Statistical Software Testing," IEEE Transactions on Software Engineering, October 1994, 20 (10), pp. 812-824.

[15] Wohin, C. and P. Runeson. "Certification of Software Components," IEEE Transactions on Software Engineering, June 1994, 20 (6), pp. 494-499.

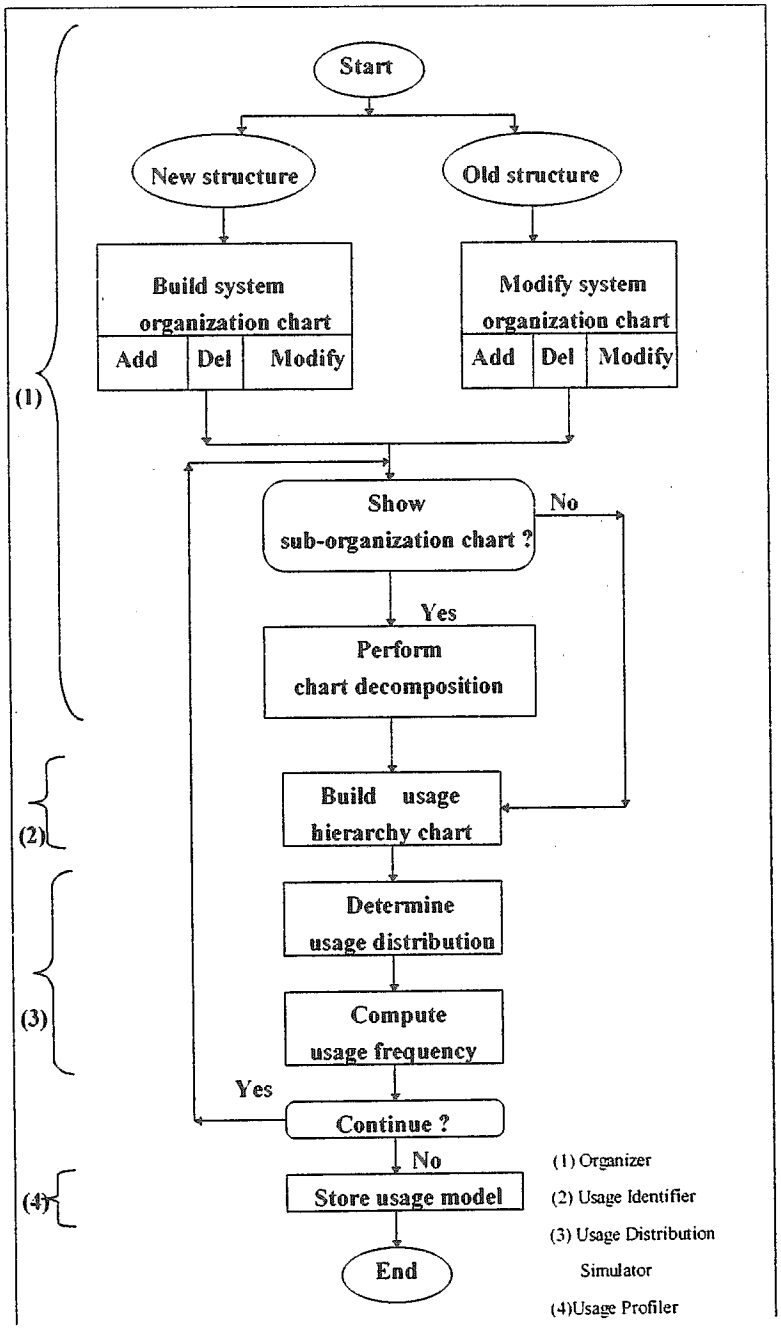

Fig. 1 Usage modeling process