2010 International Forum on Information Technology and Applications

A Quantitative Approach to Software Maintainability Prediction

Liang Ping[1]
[1] Southwest University for Nationalities (SWUN), Chengdu 610041, China
liang_p@hotmail.com

ABSTRACT: Software maintainability is one important aspect in the evaluation of software evolution of a software product. Due to the complexity of tracking maintenance behaviors, it is difficult to accurately predict the cost and risk of maintenance after delivery of software products. In an attempt to address this issue quantitatively, software maintainability is viewed as an inevitable evolution process driven by maintenance behaviors, given a health index at the time when a software product are delivered. A Hidden Markov Model (HMM) is used to simulate the maintenance behaviors shown as their possible occurrence probabilities. And software metrics is the measurement of the quality of a software product and its measurement results of a product being delivered are combined to form the health index of the product. The health index works as a weight on the process of maintenance behavior over time. When the occurrence probabilities of maintenance behaviors reach certain number which is reckoned as the indication of the deterioration status of a software product, the product can be regarded as being obsolete. Longer the time, better the maintainability would be.
KEYWORDS: software maintainability$_1$; hidden markov model$_2$; software metrics$_3$

I. INTRODUCTION

Software evolution is inseparable from software maintainability which definitely goes deteriorated as time goes by and changes keep implemented. Due to the unpredictability of change occurrence and varieties of faults, the scope and cost of software maintainability are indefinite after a software product being delivered. However, it is reasonable to set a threshold as the quantitative criteria of software maintainability so as to determine the health status of a software product if there is a way to know the possible rate of the software product getting deteriorated. A Hidden Markov Model is chosen to reflect the process of software maintainability of a software product, simulating the process by its maintenance behaviors. The increment of its probabilities of maintenance behaviors along with the HMM evolvement is served as the representation of the product's deterioration rate. In this paper, a possible threshold for the deterioration rate is established by the empirical data according to the figures in [1].

In another hand, software metrics measuring the properties of software products cover influential factors that make impacts on software maintainability economically and technically, e.g. cost, hardware, and environment, so as to indirectly help assessment of a software product in its circumstance at a certain time. It can provide the initial specification of how good a software product is at the time of delivery. Its figure certainly can be associated with how long a product can last. So the properties of a software product are measured and forged into one constant number. The number shall affect the increment of the probabilities of a product's maintenance behaviors that eventually shall reach the given threshold. The time of a product's maintenance behaviors reaching the given threshold is an outline of the life cycle of the software product. In the following sections, the details of this approach are elaborated.

II. ESTABLISHING THRESHOLD FOR SOFTWARE MAINTAINABILITY

Software evolution has its own course other than that happening in our natural biological world. The differences lie in the influential factors and how the factors interact in their evolutions. The software evolution is inseparable from software maintainability. The influential factors in software evolution can be reckoned as being equivalent to those of software maintainability so that the results of factor interaction are denotable by the probabilities of occurrence of maintenance behaviors. So according to ISO/IEC 14764:2006, 2006 [3], software maintainability is all about change management and categorizes maintenance as following,

- Corrective maintenance: Any change to a software project after being delivered to detect and correct any existing fault.
- Adaptive maintenance: Any change to a software project after being delivered to adapt it to a changed or changing environment;
- Perfective maintenance: Any change to a software project after being delivered to improve performance or maintainability;
- Preventive maintenance: Any change to a software project after being delivered to detect and correct any potential fault.

Therefore, it is reasonable to reckon different types of maintenance as the key factors influencing software maintainability and thereby software evolution. The next step would be to find the threshold.

According to the economic analysis of different types of maintenance work in [1], the changes engaging flexible data structure design and customer report generation capability are most influential in software maintenance. These two factors are subject to changes by customers after delivery of software products, which lay in the category of adaptive maintenance. So an average percentage of adaptive maintenance occurring in software products can reflect the health status of a qualified software product. The data can be deduced from the distribution figures from [1]. Firstly, it is possible to categorize the software maintenance efforts by nature into the four types of maintenance as below,

TABLE I. PERCENTAGE OF MAINTENANCE EFFORT AND TYPE OF MAINTENANCE

Software Maintenance Effort	Percentage of distribution	Type of Maintenance	Percentage of distribution
Emergency program fixes	12.4	Corrective maintenance	12.4
Routine debug	9.3	Preventive maintenance	9.3
Accommodation changes to input data, files	17.4	Adaptive maintenance	65.4
Accommodation changes to hardware, OS	6.2	Adaptive maintenance	
Enhancement from users	41.8	Adaptive maintenance	
Improve documentation	5.5	Perfective maintenance	9.5
Improve code efficiency	4	Perfective maintenance	
Others	3.4		

So the percentage of distribution of different types of maintenance shows that 65.4% of maintenance efforts fall into adaptive maintenance. That is to say, a qualified software product in its life cycle should not exceed its adaptive maintenance over 65.4% for the sake of cost and complexity. Longer the time it takes to reach 65.4% of adaptive maintenance, better the software maintenance of a software product would be. Therefore, the time for a software product to reach 65.4% of adaptive maintenance after its delivery is used here as a threshold for evaluation of software maintenance and thereby software evolution.

III. MEASURING SOFTWARE QUALITY

A software quality model includes the measurement of the properties of stability, analyzability, changeability and testability as sub-characteristics of a software product. Each sub-characteristic can be measured properly by many methods of metrics and each method of metrics can be applied to more than one sub-characteristic. By Multiplication Rule of Statistics, the indexes of all properties can be multiplied to produce a joint statistics of all the properties combined. Based on this, the Table 2 below gives a list of metrics for each property so as to measure the quality of a completed software product, that is, the health status of a completed software product at the time of delivery.

TABLE II. METRICS MEASURING THE QUALITY OF COMPETED SOFTWARE PRODUCTS

Attribute	Metrics Implemented	Result Analysis
Changeability	1. LOC 2. Cyclomatic Complexity	1. Changing requires understanding of an entire software entity. The difficulty increases naturally when LOC increases. 2. Cyclomatic Complexity computes the number of the linearly independent paths and each modification must be correct for all execution paths. Therefore, Changeability declines when Cyclomatic Complexity increases.
Testability	1. LOC 2. Cyclomatic Complexity	1. Complete testing requires coverage of all possible codes. The difficulty increases when LOC increases. 2. Complete testing requires coverage of all execution paths. So testability declines when Cyclomatic Complexity increases.
Analyzability	1. LOC 2. Cyclomatic Complexity	1. LOC directly has impact on the time and effort required to diagnose errors or faults, and the modules related to them and needed to be modified. 2. Analyzability declines when Cyclomatic Complexity increases, which means the higher complexity of the control flow.
Stability	1. Coupling Between Objects	1. Modules with a high coupling can affect the stability. So stability decreases when the coupling between objects increases.

The quantification of the properties in Table 2 would be,

1. $$\frac{\text{BKLOC}^{1}}{\text{IA BKLOC}^{2}} = \text{Ratio of BKLOC}$$

[1] BKLOC – Bugs in 1K lines of code;
[2] IA BKLOC – Industry average BKLOC [5] =15-20BKLOC

2. $\text{CC}^{3}/10^{4}$ = Ratio of CC

[3] CC – Cyclomatic Compexity;
[4] 10 – The threshold value recommended by McCabe in [6]

3. CBO^{5}

[5] CBO - Coupling Between Objects

If a software product has better stability, analyzability, changeability and testability, it certainly will cost less for its maintenance after its delivery, particularly in the aspect of adaptive maintenance. These sub-characteristics can compose a perfect weight on the effect of maintenance behaviors. Therefore, the method is to forge the measurements of sub-characteristics into a constant ***C*** as a weight on the evolution process of a software product. The constant represents the health status of a software product when delivered. The smaller C represents a better health.

$$\boldsymbol{C} = \text{Ratio of BKLOC} + \text{Ratio of CC} + \text{CBO}$$

IV. Creating a HMM

A HMM [4] is a matrix with cells representing the states of a matter in different timestamps displaying a process of a matter's status evolution. In order to display the status evolution of software maintainability, a HMM is set up, using the probabilities of four types of maintenance in Table 1 as the row items and the probabilities of how the maintenance behaviors caused by those in last timestamps are oriented in the next timestamp as the column items of a HMM.

To create a HMM, the states $s_1, s_2, \ldots, s_n$ are set up as the row items and each state indicates the probabilities of each kind of maintenance behaviors occurring individually. The column items are the probabilities of a software product switching from one kind of maintenance behavior to another, that is to say, combining the probabilities of the occurrence of two kinds of maintenance behaviors. By Multiplication Rule of Statistics, the multiplication of two probabilities can give the result of the probability of one maintenance behavior caused by another one. Starting from an initial status, the matrix can evolve and give prediction of the maintenance orientation to show how the maintainability develops.

The algorithm starts with the initial states needed. Let us set s_i ($1 \leqslant i \leqslant 4$) to be the percentage of each kind of maintenance (the percentages of others are insignificant and ignored according to [1]) as the row items. And $P(p_{t+1}=s_j \mid p_t= s_i)$ indicates the probability of the state s_i causing that of the state s_j from time t to t+1. Therefore, the cells of the HMM matrix can be calculated as below,

$$a_{ij} = s_i * s_j \quad 1 \leqslant i, j \leqslant 4$$

Among which, s_i and s_j are the percentages generated from Table 1 as the probabilities of the occurrence of two kinds of maintenance behaviors. The multiplication of s_i and s_j can give the result of the probability of the maintenance behavior s_j caused by s_i. So the initial states of maintenance would be like given $1 \leqslant i \leqslant 4$,

i		$P(p_{t+1}=s_1 \mid p_t= s_i)$	$P(p_{t+1}=s_2 \mid p_t= s_i)$	$P(p_{t+1}=s_3 \mid p_t=s_i)$	$P(p_{t+1}=s_4 \mid p_t= s_i)$
1	s_1	a_{11}=0.128	a_{12}=0.096	a_{13}=0.677	a_{14}=0.098
2	s_2	a_{21}=0.128	a_{22}=0.096	a_{23}=0.677	a_{24}=0.098
3	s_3	a_{31}=0.128	a_{32}=0.096	a_{33}=0.677	a_{34}=0.098
4	s_4	a_{41}=0.128	a_{42}=0.096	a_{43}=0.677	a_{44}=0.098

As required by a HMM, the sum of each row should be 1. So the model is normalized by

$$a_{ij} = s_i * s_j / \sum_{j=1}^{4} s_i * s_j \quad 1 \leqslant i \leqslant 4$$

And the model becomes,

i		$P(p_{t+1}=s_1 \mid p_t= s_i)$	$P(p_{t+1}=s_2 \mid p_t= s_i)$	$P(p_{t+1}=s_3 \mid p_t=s_i)$	$P(p_{t+1}=s_4 \mid p_t= s_i)$
1	s_1	a_{11}=0.015	a_{12}=0.012	a_{13}=0.081	a_{14}=0.012
2	s_2	a_{21}=0.012	a_{22}=0.009	a_{23}=0.061	a_{24}=0.009
3	s_3	a_{31}=0.081	a_{32}=0.061	a_{33}=0.428	a_{34}=0.062
4	s_4	a_{41}=0.012	a_{42}=0.009	a_{43}=0.062	a_{44}=0.009

From one moment t, the model would evolve into the moment t+1 starting from this initial status. To calculate how long it takes to reach the threshold by 65.4% of adaptive maintenance, the traditional algorithm is to assume that

For each state s_i, define $p_{t(i)}$ = Probable state is s_i at time t = $P(q_t = s_i)$

The algorithm would be,

$p_0(i) = P(q_0=s_i) = 1$ if s_i is the start state, or 0 if otherwise;

$$p_{t+1}(j) = P(q_{t+1}=s_j) = \sum_{i=1}^{4} a_{ij} * p_t(i)$$

Now, given each software product its own evolving rate, the weight representing the quality of a software product can be applied here to make impact on the process of software maintainability, different from the traditional HMM algorithm above,. So, the algorithm is modified as below,

$p_{0(i)}=P(q_0=s_i)= 1*C$ if s_i is the start state, or 0 if otherwise;

$$p_{t+1(j)} = P(q_{t+1}=s_j) = \left(\sum_{i=1}^{4} a_{ij} * p_t(i)\right)*C$$

Let assume a corrective maintenance behavior starts maintenance process of a software product for the reason that the first change of a newly delivered software product is quite often made for correction of any existing problem occurring in usage. So, the algorithm starts,

1. $p_0(1) = P(q_0=s_1) = 1*C$
2. And, the model becomes at the time of t+1, $p_{t+1(j)}$ = $P(q_{t+1}=s_j) = \left(\sum_{i=1}^{4} a_{ij} * p_t(i)\right)*C$
3. If the threshold is reached, the time t shows the time period. Otherwise, go to step 2.

The step 2 and 3 are carried out recursively till the threshold is reached. The result t is significant to outline the evolution of a software product.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

V. Conclusion

From the algorithm, it can be concluded that software maintenance is possibly measurable by applying the HMM algorithm given in this paper though more studies should be put into the data analysis of the occurrence of maintenance behaviors in different types, in which case the practical ground for the algorithm can be more solid and robust. So, an outline of software evolution is given in a quantitative way. With more methods invented in software metrics, the constant C could give more precise indication of software product. Further study shall be carried out to address these matters. With the study in maintenance behavior analysis, the algorithm can be further refined.

Acknowledgment

The paper is sponsored by The Returned Overseas Scholars Innovation Project of Southwest University for Nationalities.

References

[1] Boehm B. , “Software Engineering Economics”, Prentice Hall (1981)

[2] ISO/IEC 14764:2006, http://www.iso.org/iso/catalogue_detail.htm?csnumber=39064

[3] Cem Kaner, and Walter P. Bond, “Software Engineering Metrics: What Do They Measure and How Do We Know?”, 10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, METRICS, 2004

[4] Lawrence R. Rabiner (February 1989). "A tutorial on Hidden Markov Models and selected applications in speech recognition", Proceedings of the IEEE 77 (2): 257–286, 1989.

[5] Steve Cornwell, “Code Complete: A Practical Handbook of Software Construction”, Microsoft Press; 2nd edition (June 9, 2004).

[6] Thomas McCabe, “A Complicity Measure”, IEEE Transaction on Software Engineering, VOL. SE-2, No. 4, 1976