Approach to Software Maintainability Prediction Versus Performance

Ram Kumar Singh, Akanksha Balyan

ABSTRACT:- The software maintainability is one of the most significant aspects in software evolution for the software product. Due to the complexity of chase maintenance demeanor, it is difficult to accurately anticipate the price and risk of maintenance afterward delivery of the software products. The value of a software system results from the interaction between its functionality and quality attribute (performance, reliability and security) and the market-place. The software maintainability is viewed considered as an inevitable evolution procedure driven through maintenance demeanor. Traditional product cost model have focused on the short term development cost of the software product. A HMM (Hidden Markov Model) is applied to simulate the maintenance demeanor demonstrated as their potential occurrence probabilities. The software metric function is the measurement of the software quality products and its measurements results of a software product existence delivered combined to from health index of the software product. When the occurrence probabilities of maintenance demeanor reach certain number which is calculate as the denotation of worsening position of software product, the software product can be considered as obsolete. The longer time, more beneficial the maintainability would be. We believe on the architectural approach to price-modeling will be able to capture these concerns so that the software can reason about the risk I the system and price of mitigating them.

KEYWORDS: Software maintainability, HMM (Hidden Markov Model), Performance modes between availability and Software metrics.

I. INRODUCTION

The software evolution is inseparable by software maintainability which emphatically goes worsening as time goes along and alters keep implemented. Due to the unpredictability of modification happening and kind of faults, the range and price of software product maintainability are indefinite later on software product existence delivered. However, it is sensible to set a threshold as the quantitative standards of software product maintainability and then to decide the health condition of a software product if there is a path to know the potential rate of the software acquiring deteriorated. A HMM is preferred to reflect the process of the software maintainability of a software products, simulating the process through its maintenance demeanors. The increase of its probabilities of maintenance demeanors on with the Hidden Markov Model evolvement is do as the delegate of the software product's deterioration rate. This paper, a potential threshold for the deterioration rate is based upon the empirical data in table1.

Manuscript received on April 14, 2012.

Ram Kumar Singh, S.I.T.E, Swami Vivekanand Subharti University, Meerut(Email: kumarcool88@gmail.com) ,
Akanksha Balyan, S.I.T.E, Swami Vivekanand Subharti University, Meerut (Email: chaudharyakanksha@gmail.com).

In other case, software metrics measuring out the properties of the software products covering influential element that make construct impacts on software maintainability technically and economically, e.g. price, hardware and environment so as to circuitously assist assessment of a software product in its circumstance on certain time. It can also allow for the initial specification of however beneficial a software product is on delivery time. In table surely can be related on how long time a software product can be end. Thus the attribute of a software product are evaluated and forged in one fixed number. The number will affect the growth of the probabilities of a software product's maintenance demeanors that finally will reach the committed threshold. The time of a software product's maintenance demeanors reaching the committed threshold is and delineate of the life cycle of the software product. In the coming sections, the details of this approach are expanding.

II. ESTABLISHING THRESHOLD FOR SOFTWARE MAINTAINABILITY

Software evolution has its own course of instruction other than that bechancing in our natural biological cosmos. The differences lie in the important element interacts in their evolution. The software evolution is inseparable by software maintainability. The influential element in software evolution can be reckoned as existence equivalent to those of software maintainability and then the effect of factor interaction is denotable through probabilities of happening of maintenance demeanor. Thus according to ISO/IEC (International Organization for Standardization/International Electro-technical Commission) 14764:2006, 2006, software maintainability is all around modification management and families' maintenance as following,

- Corrective Maintenance: Any modification in software product after being presented to find and correct any existent error.
- Adaptive Maintenance: Any modification to a software product after being presented to accommodate it to modified or modifying environment.
- Perfective Maintenance: Any modification in a software product after being presented to better performance or maintainability.
- Preventive Maintenance: Any modification in a software product after being presented to find and correct any possible error.

Thus, it is reasonable to compute dissimilar type of maintenance as the fundamental factors influencing software maintainability and thereby software evaluation. The next step will be to determine the threshold.

According to the economical study of dissimilar type maintenance work in table [1], the modification engaging flexible data structure pattern and customer report generation potentiality are almost influential in software maintenance. These two elements are subject to modified by customers after delivery of products, which lay in the class of adaptive maintenance. Therefore a mean percentage of adaptive maintenance passing in software products can ponder the health status of a characterized software product. The data can derive by distributed table from [1]. Firstly, it is potential to categorize the software maintenance exertion by nature into the four type of maintenance as below,

Table 1 PERCETAGE MAINTENANCE EFFORT AND TYPE OF MAINTENANCE

Software Maintenance Effort	Percentage of Distribution	Types of Maintenance	Percentage of Distribution
Emergency program fixes	11.4	Corrective maintenance	11.4
Routine debug	8.3	Preventive maintenance	8.3
Accommodation changes to hardware, OS	16.4	Adaptive maintenance	
Accimmodation change to input data files	5.2	Adaptive maintenance	
Enhancement from user	40.8	Adaptive maintenance	
Improve code documentation	5.5	Perfective maintenance	8.5
Improve code efficiency	4	Perfective maintenance	
Others	2.4		

So the percentage of statistical distribution of dissimilar type of maintenance depicts that 65.4% of maintenance attempts come down into adaptive maintenance. That is to enounce, a qualified specified software product in its life cycle ought to none exceed its adaptive maintenance all over 65.4% for the sake of price and complexity. Longer the time it accepts to attain 65.4% of adaptive maintenance, more beneficial software maintenance of a software product would-be. Thus, the time towards a software product to reach 65.4% of adaptive maintenance later on its delivery is utilized here as threshold as the evolution software maintenance and thereby software evolution.

III. SOFTWARE PERFORMANCE VERSE AVAILABILITY ENGINEERING

Thirteen year later the term engineering was introduced, Connie Smith coined the term Software Performance Engineering (SPE) in her seminal paper published in 1981. That paper brought attention to the fact that software development was expressed out with the “fix-it-later “attitude while it come to performance. In other word, performance was never a design condition, but reconsideration. The reason how come performance in SPE is not yet redundant is that twenty year later Smith’s introduction of the concept backside SPE, SPE has not yet been integrated into the exercise of Software Engineering. Thus, it is still significant to talk regarding SPE until the P of SPE turns redundant, i.e., until it actually mix into SE.

The reasons why performance does not receive proper attention during software design:

- Lack of Scientific Principle and Model: Conventional engineering must use technological principle and model depends upon mathematics, physics and computational science, to hold their design process. There have been many developments in terms of formal model to support the software life cycle. Most of this work is centered on methodologies to manage the complexity of the process of software development, testing, maintenance and evolution.
- Education: Graduate of computer science and related engineering program are often unprepared to address the software engineering problem faced by industry. Majority of undergraduate computer science and computer science related curricula do not include any required course in computer system performance evaluation and offer only minimal performance-related hours, generally in operating system and computer network courses.
- IT Workforce: U.S Information Technology (IT) workforce estimate range from 2 to 10 million depending upon source and definition of IT worker. A more accurate estimate for “core IT workforce” that only include only computer engineers, the computer system analysts and scientists, computer programmers and computer science teachers place the number at 2.5 million people in 1999 in the U.S

IV. MEASURING SOFTWARE QUALITY

In software quality model includes the measurement of the attribute of constancy, analyzability, changeableness and testability as sub-characteristics of a software product. Each sub-characteristic can be measured by order through many methods of metrics and each method of metrics can be utilized to more than one sub-characteristics. Though Multiplication Rule of Statistics, the indexes of all attributes can be multiplied to get a joint statistics of all the properties mixed. Based on this, the Table 2 below gives a listing of metrics as each attribute so as to measure the character of a completed software product, which is health position of a completed software product on the time of delivery.

Table 2 METRICS MEASURING THE QUALITY OF COMPLETED SOFTWARE PRODUCTS

Property	Metrics Implemented	Result Analysis
Changeableness	1. LOC 2.Cyclomatic Complexity	1. Changing requires understanding of an entire software entity. The trouble rises naturally if LOC rises. 2. CC calculates the number of linearly autonomous path and each change must be right for all execution path
Testability	1.LOC 2.Cyclomatic Complexity	1. Finish testing needs coverage of all possible codes. The trouble rises if LOC increases. 2. Complete testing need coverage of all execution paths. So testability reject if CC raises
Analyzability	1.LOC 2.Cyclomatic Complexity	1.LOC instantly has hit on the time and attempt needs to diagnose mistakes, and modules associated to them and required to be changed 2. CC rises than analyzability declines, which means the higher complexity of the control flow.
Constancy	Coupling between Objects	Modules with high coupling can affect the constancy, So constancy reductions when the coupling among objects rises.

The quantification of the attributes in Table 2 would be,

1. CKLOC[1]
   =Ratio of CKLOC
   IA CKLOC[2]
   [1]CKLOC-Bugs in 1k lines of codes;
   [2]IA CKLOC-Industry average CKLOC [5] =15 20CKLOC
2. CC[3]/10[4]=Ratio of CC
   [3]CC-Cyclomatic Complexity;
   [4]10-The threshold value recommended though McCabe in [6]
3. CBO[5]
   [5]CBO-Coupling between Objects

Whenever a software product has more beneficial constancy, analyzability, changeableness and testability, it surely will price less for its maintenance late on delivery, especially in the aspect of adaptive maintenance. These sub-characteristics can compose a perfect weight on the effect of maintenance demeanors. Therefore, the method is to forge the measurements of sub-characteristics into a constant C as a weight upon the evolution process of a software product. The constant symbolizes the health status of a software product once delivered. The smaller C represents a more beneficial health.

C=Ratio of CKLOC + Ratio of CC+CBO

V. CREATING A HMM

A HMM is a matrix with cells representing the states of a matter in dissimilar timestamps exhibiting a process of a matter's status evolution. The status evolution of software maintainability display in order, a HMM is a assemble, using the probabilities of four cases of maintenance in Table 1 as the row detail and the probabilities of how the maintenance demeanors induced by those in final timestamps are oriented in the adjacent timestamp as the column detail of HMM.

To make a HMM, the state's s_1, s_2,……s_n are arrange as row details and each state shows the probabilities of each form of maintenance demeanors coming individually. The column details are the probabilities of a software product changing from one kind of maintenance demeanor to another, namely, merging the probabilities of the occurrence of the two kinds of maintenance demeanor. Through multiplication rules of statistics, the multiplication of two probabilities can give the outcome of the probability of one maintenance demeanor cased by another one. Beginning from the initial state, the matrix can develop and give prevision of the maintenance orientation to show how the maintainability develops.

The algorithm code beginnings with initial states required. Let us set s_i $(1 \leq I \leq 4)$ to be the percentage of each form of maintenance as the row details. And $P(p_{t+1}=s_i|p_t=s_i)$ shows the probability of the state s_i causing that of the state s_j from to t+1. Thus, the cells of the HMM matrix can be computed as below,

$$b_{ij} = s_i * s_j \quad 1 \leq i, j \leq 4$$

Among which, s_i and s_j are the percentages yielded by Table 1 because the probability of the occurrence of two forms of maintenance demeanors. The multiplication of s_i and s_j can yield the results of the probability of the maintenance demeanor s_j caused through s_i. Thus the initial states of maintenance would be like given $1 \leq I \leq 4$,

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

As needed by a HMM, the total of each row ought to be. So the model is normalized by

$$bij = si*sj \ / \ \sum_{j \square 1}^{4} s_i * s_j \qquad 1 \leq i \leq 4$$

And the model becomes,

i		$P(p_{t+1}=s_1\|p_t=s_i)$	$P(p_{t+1}=s_2\|p_t=s_i)$	$P(p_{t+1}=s_3\|p_t=s_i)$	$P(p_{t+1}=s_4\|p_t=s_i)$
1	s_1	b_{11}=0.015	b_{12}=0.012	b_{13}=0.081	b_{14}=0.012
2	s_2	b_{21}=0.012	b_{22}=0.009	b_{23}=0.061	b_{24}=0.009
3	s3	b31=0.081	b32=0.061	b33=0.428	b34=0.062
4	s 4	b41=0.012	b42=0.009	b43=0.062	b44=0.009

From one moment t, the model evolves in moment t+1 beginning from this initial position. To computes however long it accepts to achieve the threshold by 65.4% of adaptive maintenance, the traditional algorithm is to assume that

Since 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;

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

Now, given each software product its have evolving rate, the weight representing the quality of a software product can be applied here to build impact on the process of software maintainability, dissimilar from the traditional HMM algorithm above, Hence, the algorithm is changed as below,

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

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

Let assume a disciplinary maintenance demeanor beginnings maintenance process of a software product towards the cause that the first modification of a freshly delivered software product is quite frequently build for correction of any existing problem coming about usage. Therefore, the algorithm be begins;

1. $p_0(1) = P(q_0=s_1) = 1*C$
2. And, the model goes at the time of t+1,

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

3. It the threshold is achieved, the time t shows the period of time. Other than, precede step 2.

The step 2 and 3 are carried out recursively until the threshold is achieved. The outcome is significant to outline the evolution of a software product.

Finally, complete content and organizational editing earlier formatting.

VI. CONCLUSION

From the algorithm, it can be complete that software maintenance is maybe measurable through using the HMM algorithm given in this paper by more study should be set into the data analysis of the occurrence of maintenance demeanor in dissimilar types, in which case the practical ground for the algorithm can be more solid and robust. Therefore, the outline of software evolution is given in a quantitative manner. With the many methods devised in software metrics, the constant C could give accurate indication of software product. Advance study shall be expressed out to address matters. With the study in maintenance demeanor analysis the algorithm can be further refined.

REFERENCES

[1] S. Balsamo, P. Inverardi and B. Selic, Proc. Third ACM Workshop on software and Performance, Performance, Italy Rome, July 24-27, 2002.

[2] Boehm B. , "Software Engineering Economics", Prentice Hall (1981).

[3] M. Hilton, "Information Technology Workers in the New Economy", Monthly Labor Review, June 2001, pp. 41-45.

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

[4]R.L Glass, "Frequently Forgotten Fundamental Facts about Software Engineering", IEEE Software, May/June 2001.

[5] Cem Kaner, and Walter P. Bond "Software Engineering Metrics: What Do They Measure and how Dow We Know?", 10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM METRIS, 2004.

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

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

[9] Thomas McCabe, "A complicity Measure", IEEE Transaction on Software Engineering, VOL, SE-2, No. 4, 1986.