시스템 크기와 복잡도를 고려한 누적 노력 기반의 소프트웨어 성장 모델

박 중 양† · 김 성 희†† · 박 재 흥†††

요 약

소프트웨어 시스템이 양도된 이후에 시스템의 크기가 성장하는 과정을 나타내는 수학적 모델인 소프트웨어 성장 모델은 시스템의 크기와 계획된 크기를 달성하기 위해 요구되는 노력을 예측하는데 사용될 수 있다. 본 논문은 먼저 시스템의 크기 변화량이 추가되는 노력에 비례하고 시스템의 복잡도에 반비례한다는 가정하에서 소프트웨어 성장 모델을 유도한다. 이 모델에서는 시스템의 복잡도가 중요한 역할을 하는데, 본 논문에서는 시스템 크기의 멱함수 형태인 복잡도를 제안하고 실제 자료에 적용하여 ㄴ고 유용성을 보인다. 멱함수 형태의 시스템 복잡도는 추가로 복잡도를 비교할 수 있게 하는 측도를 제공하는데, 이 측도는 시스템 크기에 무관하므로 크기가 다른 소프트웨어 시스템의 복잡도를 비교하는데 유용하게 사용될 수 있다.

A Cumulative Incremental Effort Based Software Growth Model Considering System Size and Complexity

Joong-Yang Park† · Seong-Hee Kim†† · Jae-Heung Park†††

ABSTRACT

A software growth model, a mathematical model describing the growth behavior of a software system during the evolution process, enables us to predict the future system size and incremental effort required to meet the planned system size. This paper first introduces a software growth model defined with respect to the cumulative incremental effort. It was assumed that the incremental growth of a software system is proportional to the incremental effort and inversely proportional to the system complexity. A key factor is the functional form of the system complexity. A power function of the system size is suggested as a system complexity and then applied to a real data for its validation. Such a system complexity additionally provides us with a measure for complexity comparison. Since the measure is independent of the system size, it is useful for comparing complexities of software systems of different size.

1. Introduction

In recent years software systems such as operating systems, control programs, and application program have become more complex and larger than ever. As the size of a software system increases, the cost of writing and maintaining the software system grows drastically. One of the most important problems at the early phase of software development process is the prediction of the size of a software system and its development effort for software project cost estimation. There have been many re-

※ 본 논문은 1998년 경상대학교 부설 정보통신연구센터 지원에 의해 연구되었음.
† 정 회 원 : 경상대학교 통계학과 교수
†† 준 회 원 : 경상대학교 대학원 전자계산학과
††† 정 회 원 : 경상대학교 컴퓨터과학과 교수
논문접수 : 1997년 12월 15일, 심사완료 : 1998년 11월 10일

searches on the software size estimation problem. Refer to Albrecht and Gaffney[1], Itakara and Takayanagi[2], Levitin[3] and Putnam[4]. The existing software size estimation techniques were recently reviewed by Laranjeira[5]. After a software system is developed and released, it continuously evolves by exposing its subsequent releases. This process is called the software evolution. Similarly to the software development process, we also need to predict the size of future releases and corresponding incremental effort. It seems therefore desirable to develop a model describing growth behavior of a software system during the evolution process. Such a model may be referred to as a software growth model(SGM). Application of a SGM requires a series of software size measurements to a time coordinate. Although SGMs are basically defined with respect to time, it is possible to define with respect to other bases such as release sequence number and effort. Another issue concerned with data collection is how to measure software size. Often total number of code lines and total number of modules are used. Turski[6] recently suggested a SGM relating total number of modules to release sequence number. It was assumed that the incremental effort (usually in man-months) spent on each release remains constant throughout the system evolution.

This brief paper introduces a SGM based on the incremental effort. The model may be viewed as a modified continuous version of Turski model. The Turski model is briefly reviewed in Section 2. Section 3 derives the SGM and suggests a specific system complexity. Then Section 4 fits the suggested SGM to a real data and shows its usefulness. In Section 5 system complexity is discussed and a new measure for complexity comparison is proposed. Conclusions are presented in Section 6.

2. Review of Turski Model

As mentioned in Section 1, it is necessary to research the SGM. Unfortunately Turski[6] is the only literature on the SGM, as far as the authors know. We thus briefly review the Turski model in this section. Denoting the total number of modules of ith release by s_i, Turski[6] relates s_i to release sequence number i. It is assumed that the incremental effort E for each release is constant throughout the system evolution and the incremental growth Δ_i of ith release is proportional to the incremental effort and inversely proportional to the system complexity. Three system complexity measures s_i^k, $k=1,2,3$ were considered by Turski[6]. Since $\Delta_i = s_{i+1} - s_i$, Turski model is therefore expressed as

$$s_{i+1} = s_i + \alpha \frac{E}{s_i^k} . \tag{1}$$

Once $E' = \alpha E$ is estimated for given k, the sizes of future releases can be predicted. Thus a simple method for estimating E' was also proposed. (Turski[6] does not explicitly take the proportional constant α into account. However, identical results are obtained.) Since $E' = \Delta_i s_i^k$ from Equation (1), Turski[6] obtained an estimate of E' as the average of E_i's, where $E_i = \Delta_i s_i^k$.

3. A SGM Based on the Cumulative Incremental Effort

This section derives a SGM under the following assumptions.

Assumptions

(1) The system evolution process begins at the first release.
(2) The system size is an increasing function of the cumulative incremental effort.
(3) The incremental growth of a software system is proportional to the incremental effort.

(3) The incremental growth of a software system is inversely proportional to the system complexity.

(5) The system complexity is an increasing function of the system size.

The software development process and software system evolution process should be distinguished. The software system evolution process begins after a software system is developed and released. Thus assumption (1) holds. Turski model assumes that the incremental effort spent on each release remains constant throughout the system evolution and growth of a software system occurs discretely at each release time as observed in the equation (1). But in general situations it is more realistic to assume that the incremental effort for each release varies and the software system grows continuously during the system evolution process. The incremental growth is directly related to the incremental effort, not to time or release sequence number. These are concerned with assumption (2), which implies that the newly developed SGM will be expressed a continuous function of the cumulative incremental effort. As the software system grows, the software system generally becomes more complex and the marginal productivity of the incremental effort reduces. Consequently the incremental growth due to unit incremental effort reduces. Therefore we are led to assumptions (3), (4) and (5). Assumptions (3) and (4) are also explicitly employed in Turski model. However, assumption (5) is a generalization of the power function type system complexity of Turski model. Two points are noteworthy. First, the system evolution process is considered as a continuous process. Second, the system evolution process depends on the cumulative incremental effort. Let w denote the cumulative incremental effort.

Denote the cumulative incremental effort, system size and system complexity by w, $s(w)$ and $g(s(w))$, respectively. Assumptions (3) and (4) imply that

$$[s(w+dw)-s(w)] = \alpha \frac{dw}{g(s(w))} .$$

Letting dw approach to zero, we have the following differential equation.

$$g(s(w)) \frac{d(s(w))}{dw} = \alpha$$

Let w_1 and s_1 be the cumulative incremental effort and system size of the first release. Solving the above differential equation under the boundary condition $s_1 = s(w_1)$, we get

$$s = G^{-1}(\alpha(w-w_1)+G(s_1)), \qquad (2)$$

where $G(s) = \int_0^s g(x)dx$ and G^{-1} is the inverse function of G. It is worthy of note that the system complexity g is the key factor. If the functional form of g is given for a software system, we can obtain a SGM for the software system by substituting g into equation (2). However, it does not seem possible to develop a universal functional form of g, since the functional form of g may differ software system by software system. Furthermore the functional form of g also depends on how to measure the system size.

For the sake of application and illustration we henceforth assume that $g(s) = s^k$, $k>0$. Although this may seem to be similar to system complexity measures of Turski model, yet it should be note that k is an unknown positive real number. Therefore k should be estimated. Refer to Section 5 for more discussion on the system complexity and k. Equation (2) is then rewritten as

$$s = [\alpha(k+1)(w-w_1)+s_1^{k+1}]^{\frac{1}{k+1}} . \qquad (3)$$

This SGM will be applied to a real data in the following sections.

4. Model Validation

Application of the SGM proposed in the previous section requires measurements on s and w of each release. We denote the observed values of ith release by s_i and w_i. If s_i and w_i, $i=1,2,\cdots$ are available, we can build a nonlinear regression model

$$s_i=[\alpha(k+1)(w_i-w_1)+s_1^{k+1}]^{\frac{1}{k+1}}+\varepsilon_i, \qquad (4)$$

where ε_i is an error term. We should estimate parameters α and k. Since no distributional assumption is made, the least squares method seems to be an appropriate estimation method. Least squares estimates for nonlinear models can be obtained by well known statistical packages such as SAS and SPSS. Substituting the estimates of α and k into equation (2), we can predict either future system size and incremental effort required to reach the planned system size.

Assuming that incremental effort of each release is constant, Turski[6] analyzed a real data, appeared in Lehman and Belady[7], consisting of release sequence number and system size measured by the number of modules. This section fits the suggested SGM to the same data, which is reproduced in Table 1. In case of constant incremental effort, we can write w_i as i without loss of generalization. Then equation (4) becomes

$$s_i=[\alpha(k+1)(i-1)+s_1^{k+1}]^{\frac{1}{k+1}}+\varepsilon_i. \qquad (5)$$

Using NLIN procedure of SAS, the least squares estimates of α and k are obtained as 34870144.49 and 1.78. The results are summarized in Table 2. In order to evaluate the goodness-of-fit, fitted system sizes and predicted system sizes for future releases ($i=22,23,24$) are also computed and presented in Table 1. The fitted and predicted system sizes are denoted by $\hat{s}_i$. Fig. 1 shows s_i and $\hat{s}_i$ plotted against i. In order to visualize the adequacy of the fitted model, the standardized residuals are plotted in Fig. 2. It is apparent that the suggested SGM is adequate for the data and works well.

〈Table 1〉 Data on a system evolution

i	s_i	$\hat{s}_i$	i	s_i	$\hat{s}_i$
1	977	977.00	13	1902	1952.00
2	1344	1126.78	14	2087	2000.93
3	1390	1247.67	15	2151	2047.82
4	1226	1350.70	16	2091	2092.87
5	1246	1441.39	17	2095	2136.27
6	1492	1522.94	18	2101	2178.15
7	1581	1597.40	19	2312	2218.65
8	1800	1666.15	20	2167	2257.88
9	1595	1730.21	21	2315	2295.93
10	1897	1790.31	22		2332.90
11	1832	1847.02	23		2368.85
12	1897	1900.80	24		2403.86

〈Table 2〉 Results of nonlinear least square estimation

ANOVA Table

Source	DF	Sum of Squares	Mean Square
Regression	2	69771821.40	34885910.70
Residual	19	215766.60	11356.14
Total	21	69987588.00	

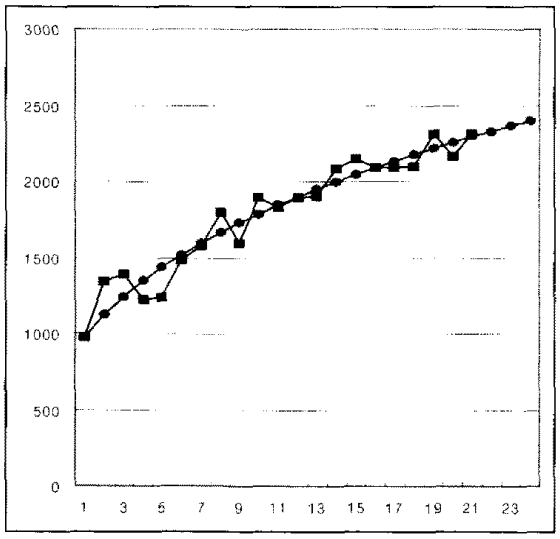
Parameter	Estimate	Asym. Std Error
α	34870144.49	107487505.43
k	1.78	0.42

5. System Complexity and Complexity Index

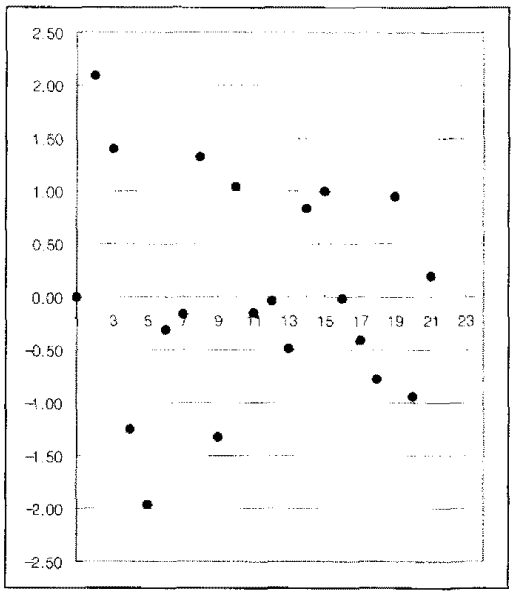
In the previous two sections we considered a SGM of which the system complexity is $g(s)=s^k$. This type of the system complexity was shown to be adequate for the real data in Table 1. An important property of such system complexity is that

k varies software system by software system. This seems to be desirable since two software systems of equal size can have different system complexities. That is, the suggested system complexity implicitly takes account of the characteristics of the software system under consideration.

Now consider a problem of comparing complexities of two or more software systems. For example, suppose that we want to compare the system complexities of an operating system and a statistical package. Statistical packages are usually much larger than operating systems. Therefore the system complexity of a statistical package would be larger than that of an operating system. However, it is generally accepted that operating systems are more complex than statistical packages. That is, the system complexity is not an appropriate measure for complexity comparison. This is because the sizes of software systems are usually unequal and the system complexity depends on the system size. Complexity comparison therefore requires a measure for degree of complexity that is independent of the system size. One possible measure is the parameter k, which is referred to as the complexity index. The larger k is, the more complex is the corresponding



(Fig. 1) s_i and $\widehat{s_i}$ plotted against i.
(■ : s_i, ● : $\widehat{s_i}$)



(Fig. 2) Standardized residuals plotted against i.

system. The complexity index of the data considered in the previous section is estimated as 1.78, while Turski[6] arbitrarily set k to 2. This implies that the software system is not as complex as Turski assumed.

6. Conclusions

A SGM describes the growth behavior of a software system during the evolution process. It is as important as the software size estimation model for the development process. This paper proposed a new SGM based on the incremental effort. The proposed SGM is useful for predicting the future system sizes and required incremental efforts. A key factor of the suggested SGM is the system complexity. We suggested a power function of the system size as a system complexity. Its validity has been empirically shown by a numerical illustration. It also provides us with a complexity measure independent of the system size, referred to as the complexity index, for complexity comparison. However, further researches are necessary for functional forms of the system complexity. Therefore more data on the software system evolution should be collected and analyzed.

References

[1] A. J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction : A Software Science Validation," IEEE Trans. Software Eng., Vol. SE-9, No.6, pp.639-647, 1983.

[2] M. Itakara and A. Takayanagi, "A Model for Estimating Program Size and Its Evaluation," in Proc. Sixth Int. Conf. Software Eng., pp.104-109, 1982.

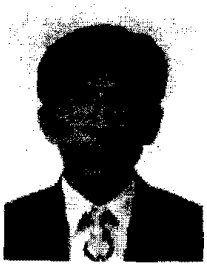
[3] A. V. Levitin, "On Predicting Program Size by Program Vocabulary," in Proc. IEEE COMPSAC, pp.98-103, 1985.

[4] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," IEEE Trans. Software Eng., Vol.SE-4, No.4, pp.345-361, 1978.

[5] L. A. Laranjeira, "Software Size Estimation of Object Oriented Systems," IEEE Trans. Software Eng., Vol.16, pp.510-522, 1990.

[6] W. M. Turski, "Reference Model for Smooth Growth of Software System," IEEE Trans. Software Eng., Vol.22, No.8, 1996.

[7] M. Lehman and L. Belady, 'Program Evolution -Process of Software Change', Academic Press, 1985.



박 중 양

e-mail : parkjy@nongae.gsnu.ac.kr

1982년 연세대학교 응용통계학과(학사)

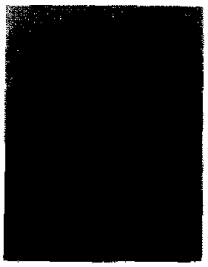
1984년 한국과학기술원 산업공학과 응용통계전공(석사)

1994년 한국과학기술원 산업공학과 응용통계전공(박사)

1984년~1989년 경상대학교 전산통계학과 교수

1989년~현재 경상대학교 통계학과 교수

관심분야 : 소프트웨어 신뢰성, 신경망, 선형 통계 모형, 실험계획법



김 성 희

e-mail : shkim@sys.gsnu.ac.kr

1991년 경상대학교 수학교육과(학사)

1994년 경상대학교 전자계산학과(석사)

1996년~현재 경상대학교 전자계산학과(박사과정)

관심분야 : 소프트웨어 공학(특히, 소프트웨어 신뢰성, 소프트웨어 테스팅, 신경망)

박 재 흥

e-mail : pjh@nongae.gsnu.ac.kr

1978년 충북대학교 수학과(학사)

1980년 중앙대학교 전산학과(석사)

1988년 중앙대학교 전산학과(박사)

1983년~현재 경상대학교 컴퓨터과학과 교수

관심분야 : 소프트웨어 신뢰성, 시험도구 자동화