FACTORS ON SOFTWARE EFFORT ESTIMATION

Simon WU Iok Kuan

Faculty of Business Administration, University of Macao, Macau, China

ABSTRACT

Software effort estimation is an important process of system development life cycle, as it may affect the success of software projects if project designers estimate the projects inaccurately. In the past of few decades, various effort prediction models have been proposed by academicians and practitioners. Traditional estimation techniques include Lines of Codes (LOC), Function Point Analysis (FPA) method and Mark II Function Points (Mark II FP) which have proven unsatisfactory for predicting effort of all types of software. In this study, the author proposed a regression model to predict the effort required to design small and medium scale application software. To develop such a model, the author used 60 completed software projects developed by a software company in Macau. From the projects, the author extracted factors and applied them to a regression model. A prediction of software effort with accuracy of MMRE = 8% was constructed.

KEYWORDS

Effort Estimation, Software Projects, Software Applications, System Development Life Cycle

1. INTRODUCTION

The problems faced by project designers in controlling and managing software projects are overrun of effort estimate. With inaccurate effort estimates, it surely affects project designers to make correct decisions and leading to the failure of the entire software project development [1]. Over the last ten couple of years, there is an increasing strong demand for software development team to build quality application software in the competitive global markets [2]. The total investment of application software development and maintenance has been overrun for the past ten years. As analyzed by [3] [4], the overestimated effort schedule is varied substantially from 41% to 258%, and the total investment over-prediction is from 97% to 151%. Besides, there are research conducted by US government agencies revealing that 60% of the software project development overrun the original scheduled completion time, 50% of these projects have been overrun the estimated costs while in the case of 46% of those completed projects were useless at all [5]. The results indicated that not only the importance of planning and controlling over software projects, but also the early effort estimation of software project development is important too.

Over the years, there are many discussions related to effort estimation of software project development [6] [7] [8] [9] [10] [11] [12]. However, some of those well-developed models cannot be used to make early estimation. Other models require more time spent by project developers to understand both user requirements and program specifications or until the programs are completely designed. For example, COCOMO measures program size with total LOC for development effort estimation. Unfortunately, LOC is unknown until the entire application software is developed. This makes the effort estimate unstable. Similarly, FPA method is used to make predication of program size using inputs, master files, logical files, interfaces and outputs. Although this method has been widely used by organizations, particularly in Europe, than LOC, it is still not known yet until the design phase is complete [13]. Due to problems of calibrating early effort estimate using both LOC and FPA, it is, therefore, needed to propose a general model for making effort estimates for small and medium application software.

2. Literature Review

Software effort estimation is one of the important activities in software development life cycle [10], as such, many estimation models have been proposed previously. These models are classified into non-algorithmic models and algorithmic models [11]. Non-algorithmic models are mainly based on comparatively methodologies such as Expert Judgment and Machine Learning Techniques [14] [15]. While algorithmic models are constructed using history numerical data [16]. For example, SLIM [17] [18]. COCOMO is a well-known estimation model which is procedurally complete and thoroughly documented for effort estimates [19]. Using COCOMO method, effort estimate proceeds through two steps. First, the basic level of COCOMO, a nominal effort estimate must be calculated by using a formula. While size measured in LOC is the only one independent variable in this level. Second, upon the calculation of nominal effort estimate, it is multiplied by a composite multiplier function, m(X), where X indicates the result of all 15 independent variables referred to cost drivers in the intermediate level of COCOMO. The functional form of the model is shown below:

$$E = a_i S^{b_i} \, m(X)$$

where S is the total program size measured in LOC and m(X) is the multiplier function.

By following a gradation, multipliers are to determine the rating of the cost drivers running from very low to very high. The coefficients a_i and b_i are calculated from a combination of the mode and the level. Software projects were categorized into three modes by Boehm [20], they are namely organic, semidetached and embedded. For those projects falling into organic type, they are relatively small in terms of program size, little innovation is required, and are designed by companies themselves. For those projects classifying into the type of embedded, they are large in terms of functions, they must be in operations within tight constraints and they require high innovation. Those projects of semi-detached class are falling somewhere between organic and embedded [19]. This makes the project designers finding difficulties to estimate the effort accurately required for application software [21].

In addition to COCOMO, Albrecht [22] [23] proposed another method named FPA for predicting application software effort using program requirements. This method is to measure the functional requirements of systems according to their complexities. In this method, the function types, such as external inputs, outputs, inquiries, external interfaces to other systems, and logical internal files, of systems are derived by classifying the system components perceived by system counters, they are then classified further and added together. According to the number of data elements of each component, it is further categorized into simple, average or complex classification. Each component is then assigned a number of calculation points according to its type and complexity. The total effort arrived for a system is calculated by multiplying the total sum of function points for all user function types by the technical complexity factors (TCF). The TCF is obtained by calibrating the level of influence (DI) of the 14 components [7].

Despite the research works for proposing the models, they have been proven unsatisfactory for predicting effort of different types of software applications [9]. With COCOMO model, it consists of three different sub-models, namely Basic, Intermediate, and Detailed COCOMO. According to evaluation suggested by Boehm [20], Intermediate model is proven to be better than the Basic model for effort prediction among the three levels. While the Detailed model is only slightly better than the Intermediate one [18] [24]. However, the 15 cost drivers of the Intermediate model are scalars ranging from 0.7 to 1.66 with a nominal value of one. This value is to represent the characteristics and the types of system projects as well as the environment where the projects were constructed. Therefore, the results of Intermediate model are significantly influenced by the values of those cost drivers, in turn detailed information of the software projects and the development environment must be identified and gathered. Since the values of the cost drivers were originally from the software projects constructed during the period of the 1960s and 1970s. This model is still at a high level of uncertainty particularly in today's complex software environment regarding its accuracy, reliability and validity [24]. Therefore, there is a high possibility of leading to inaccurate effort estimate in such a complex software environment.

For FPA method, several studies reported that project designers faced similar difficulties during system constructions. For example, due to the counting experience of the project counters who are varied from person to person. Because of this reason, the cost involved to reach the final number of function point data may be different [25] [26] [27]. In addition, when the function point data were counted by system analysts at a software house, the time spent by calculating system functions was mostly 40 hours for a medium-size system (800-2,400 FPs) and even more for large (>2,400 FPs) computer application systems. Furthermore, a study reported by [26], due to the growth of variability of function point counting, the differences in function point measures of the same system arisen by different system counters with an average of 12.2%. Jeffery [28] reported an even worse result with a 30% of variance within the same company and more than 30% among different companies. Due to the subjectivity in which function types are categorized into simple, average and complex, the potential for error variance may exist.

3. Objective Of The Study

The aim of this research is to find out how design factors, such as Development kit, Designer experience, Number of programmers, Complexity, and Education level, affect the effort required for system development. The reason of choosing these factors is because they are free from calculation of total lines of code and number of function points. Unlike LOC and FPA, the results of effort estimate are based on the information available after preliminary and detailed design. Instead, our model makes the effort estimate earlier than that of LOC and FPA, because we can determine these factors before the application software is fully developed.

4. Project Data

The author interviewed a local software company from which data were collected. The company designs small and medium systems for local customers and clients. The characteristics of the system projects include: small to medium size in terms of LOC ranged from 5,875 to 80,918; they were all developed using fourth generation programming language; they relate to shipment, import and export, inventory control and report generation activities; they were developed by a small team of two to five programmers and system analyst; the software company has a well-developed way to keep track the detailed information of the developed programs. All collected software projects were developed by following standard system development life cycle. In an attempt to confirm the reliability of both dependent and independent variables, all software projects were verified by a programmer and double checked by a system analyst. Minor mistakes were reported and corrected afterwards. The sample size consists of 60 completed software projects, from which 40 projects were drawn as a main sample and 20 projects as the holdout sample respectively. The main sample was used to develop a model for prediction purpose, and the holdout sample was used to measure the validity of the proposed model.

5. Research Model And Its Variables

Upon data collection, the following variables were proposed. Definitions of the variables are explained below. Effort is a dependent variable referring to total man-hour effort required to build a software project. Independent variables include Development_kit(Dev_kit), Designer_experience(Designer_exp), No_of_programmers(No_prog), Complexity(Comp) and Education_level(Edu_level).

5.1 Effort

This variable emphasizes the effort (man-hour) spent by project developers to design application software. Effort is measured either in man-hour or man-month depending on the size of software projects [23] [27]. In the study, we consider man-hour is because the software projects are small to medium. Some software projects didn’t last several months. For those software projects studied, only the time spent in analyzing and designing by project designers is counted. While the time spent to discuss with clients and end users are excluded. The measurement used to count the effort is the total number of man-hours for single software project. The software company has a very good practice to record detailed information, such as time spent for each project, the number of project designers assigned to a project and the development tool used, of each developed software project. Therefore, the data collection process was easy and straight forward.

5.2 Dev_kit

This variable is to measure the complexity of system development kit used by project designers. Usually, the complexity of a development kit correlate to the time required to develop software projects, as a good development kit can make programmers more productive during system development. When a suitable development kit is used, it can support the construction process by automating tasks executed at every stage of system development life cycle. It facilitates interaction among project designers by diagramming a dynamic, iterative process, rather than one in which changes are cumbersome [29]. It is also a useful tool to enable project designers to clarify end user’s requirements at the very early stage of system development life cycle [30]. CASE tool is the common development kit used to support development process in many companies. This factor is measured with a five-point Liker-like scale ranging from (1) very low productivity to (5) very high productivity.

5.3 Designer_exp

This variable is to measure the actual working experience of project designers designing application software in computer industry. The actual experience of project designers in developing software projects and the experience in a specific kind of programming language are key determinants. By common sense, an experienced project designer can reduce the number of errors to program codes if he has good mastering of that type of programming language and has a number of years in developing software projects. This leads to a minimum time in developing and maintaining programs in the future. Thus, the more the number of years of service that a designer serves in the industry, the higher the level of working experience the designer has gained. We take the average of years of experience among the team members if there is more than one participates in a project.

5.4 No_prog

This variable is to count the number of project designers working collaboratively as a team. In order to make sure a late project which can be completed on time, there are project designers who often add extra programmers [31]. Sometimes, this arrangement may not work well, especially when there is lack of proper communication among project designers and no training offered before the development. This could definitely slow down the development process and lead to many problems. However, the situation may not happen in our study, because the software projects developed by a team of project designers are small to medium in term of LOC. A project designer is relatively easy to make an accurate estimate before a software project starts. Therefore, there are no additional members who are invited to a late project. For this variable, according to the detailed information of the developed projects, we are in an easy position to collect the number of project developers responsible for each project being developed.

5.5 Comp

This variable refers to the degree of program complexity designed. A thorough understanding of the software development process improves the relationship between program complexity and maintenance effort. That is, high complexity of software projects increase the difficulty of project designers to quickly and accurately understand the programs before they are developed or repaired [32]. The higher the level of complexity of a program is, the greater the effort required by project designer [33]. Especially, when a program has highly interactive modules to communicate not only within itself, but also with modules from other programs. This will increase the time required by project designers in designing the software projects. In the study, this variable is to measure and examine system specifications and design specifications prepared by the company during analysis and design phases. Due to the characteristics of collected software projects, they all are business oriented programs. The determination process for program complexity is under the control of project designers. For this variable, the data is collected using a five-point Liker-like scale ranging from (1) very low complexity to (5) very high complexity.

5.6 Edu_level

This variable is to measure the level of education that a project designer has acquired in related field. Many companies prefer to recruit programmers who are equipped not only with extensive working experience in industry but also those who have well training with at least a bachelor degree or higher in related field. Project designers with higher level of education usually can solve programming problems more easily than those who don't. To measure the factor, we use a five-point Liker-like scale ranged from (1) very low level of education to (5) very high level of education.

A linear regression model is hypothesized following discussion of the variables and it is shown in the following equation.

$$\text{Effort} = \alpha + \beta_1\text{Dev\_kit} + \beta_2\text{Designer\_exp} + \beta_3\text{No\_pro} + \beta_4\text{Comp} + \beta_5\text{Edu\_level}$$

6. DATA ANALYSIS

The general descriptive statistics, mean and standard deviation, of the tested variables are presented in Table 1. In an attempt to find out the design factors influencing the effort estimate. We used SPSS to run a correlation test to identify any potentially useful relationships between dependent variable and independent variables. The results are shown in Table 2 after conducting multiple regression. There is an evidence revealing strong relationships between independent variables and the dependent variable. Particularly, there is a strong correlation reported between Dev_kit, No_pro and Comp and the dependent variable Effort. This is to imply the selected independent variables having potential predictive capabilities for the dependent variable.

Table 1 Summary statistics

Variable	Mean	Std. Deviation
Effort	662.19	252.21
Dev_kit	3.88	1.07
Designer_exp	5.69	2.11
No_pro	3.08	1.16
Comp	3.50	1.14
Edu_level	4.08	1.35

Table 2 Pearson correlation coefficients

Variable	Effort	Dev_kit	Designer_exp	No_pro	Comp
Dev_kit	.720	-	-	-	-
Designer_exp	-.251*	.115*	-	-	-
No_pro	.931	.550	-.061*	-	-
Comp	.645	.539	.037*	.556	-
Edu_level	.243*	.217*	-.223*	.305	-.064*

All are significant at 0.01 level, except for * significant at 0.05 level

From the results, implications have revealed design factors having strong influence on software effort estimate. One of the major factors influences the effort estimate is the No_pro, it is evidenced by the value of correlation (.93), indicating of 93% of the variance in effort estimate. This is to emphasize that when more team designers are involved in designing a software project, the overall accuracy of effort estimate can be improved substantially. Similarly, the value of Dev_kit (.72) shows significantly that it has high impact on productivity of project designers.

This is due to fact that a good development kit has a complete set of data, such as input and output formats, system data and database structures, maintained in data repository, so that designers can choose the data from the data repository. This enables designers contributing the minimum time to design the data items from the very beginning. As a result, the time spent by project designers in defining the data items is minimized significantly. Also, there is a close relationship between Effort and Comp (.64). This factor explains more time is required for development when internal complexity of a program is high. Thus, it is expected the effort estimate can be affected by the complexity of programs, especially when programs are the highly complex.

However, the correlation value of Edu_level (.-243) has low effect on the effort estimate. The possible explanation is that project designers can acquire extensive programming knowledge by themselves without attending a degree course in the modern days. Of course, some project designers can solve complex problems more easily if they have a higher level qualification but this is not true at all time. Similarly, the correlation value of Designer_exp (-.251) has totally no significant influence on effort estimate. It is because all of the system projects were developed using a single programming language. When a project designer is already familiar with a specific programming language, he may be very productive for the rest of other programs. Besides, all programs developed by the company are small to medium, extensive programming experience is not that important. Therefore, it is in line with previous studies developed by [34].

7. DISCUSSIONS

The model proposed in this study may not be an accurate predictor for future effort estimate since it is only suitable for the testing data from which it was constructed. In order to improve the accuracy and reliability of the proposed model for future effort estimate. We used an accuracy criterion for evaluation purpose. The accuracy and reliability of a model is highly important as it directly affects the success or the failure of system effort prediction. Project scheduling, controlling, coordination and staffing decisions are always depending on accurate estimation [13]. However, sometimes an inaccurate model can still be consistent if it uniformly misestimates effort for only a set of software project data. If a model is not consistent, the management may not rely much on the estimates. As a result, an inconsistent model may be of little use in practice. The accuracy of a given software project estimate is calibrated by the magnitude of relative error (MRE). The MRE is calculated for a given project by the following formula:

$$MRE = 100 \mid (\text{Actual Effort} - \text{Estimated Effort}) / \text{Actual Effort} \mid$$

Thus, the accuracy of prediction of a software project is proportional inversely to its MRE value. Besides, mean MRE (MMRE) is the mean value for the indicator over all observed data in the sample. A lower value of MMRE usually indicates a more accurate model [35]. The prediction at a given level usually explains an indication of overall fit for a set of software projects, based on the MRE values for each pair of data:

$$\text{Pred}(p) = \text{i}/n$$

where p is the selected threshold value for MRE, *i* refers to a set of software projects with a value of MRE which is smaller than or equal to *p*, while *n* is the total number of software project data. In an attempt to measure the accuracy and reliability of the proposed model, a holdout sample of

20 software projects was drawn from the main sample. Upon validation, the accuracy of predictive values for COCOMO model and the proposed model are shown in Table 3. By comparing the MMRE values of these two models, the proposed model has higher predictive capability than COCOMO model. In addition, the predicted results indicated explicitly that a simple model can generate less error than a complex one, because the design factors were directly extracted from the early stage of system analysis phase. Meanwhile, when comparing the PRED values of are models, our model has a higher predicted value than COCOMO. In general, it is in line with our expectation. Therefore, it is to conclude that the proposed model is more accurate and reliable than COCOMO model.

Table 3 Indicators of model accuracy

Model	MMRE	PRED(0.3)	PRED(0.2)	PRED(0.1)
COCOMO	.13	.76	.61	.22
New Model	.08	.82	.65	.31

8. Conclusion And Further Research Work

By drawing a conclusion, there are several points to note. The first point is to emphasize is that the results of the proposed model are promising and fruitful. It is because the possibility of developing a prediction model for application software using fourth generation programming language**.** However, although the results are pleasing, it must be addressed that we are not proposing a general model for all types of application software. The use of software projects are mainly from a small company as a basis for prediction purpose. The suggested results may not generalize to real environment.

The second point is that we avoided using COCOMO model or FPA method with two reasons. First, we don't know yet the total LOC of a program until it is fully developed. Second, the total LOC counted is different from programs to programs, it is also varied from programming language to programming language.

The third point is that it is possible to use design factors to estimate the total effort required for software project development. Such design factors are easily collected at an early stage of system development life cycle when the company has a good practice to manage system development activities. By using simple regression model, a prediction software project of effort estimate with accuracy of MMRE = 8% was constructed. And this level of accuracy was obtained relatively easy without using complex model such as COCOMO. It is to believe that our results may generalize to other software projects with similar characteristics.

9. Limitations

There are a couple of limitations: (1) the predicted results are mainly from those software projects constructed by a single company, the area of application may be only restricted to those software projects designed by that company and those software projects with similar characteristics; (2) the sample size is only 60 software projects which are mainly from a single company. To be more generalized, the sample size should be large enough and contain projects from more than one company. So that the new model can be applied to other areas to ensure its validity. (3) Further research is also needed to compare to FP Method or other methods.

REFERENCES

[1] Fu, Ya-fang, Liu, Xiao-dong, Yang, Ren-nong, Du, Yi-lin and Li Yan-jie (2010), “A Software Size Estimation Method Based on Improved FPA”, Second World Congress on Software Engineering, Vol. 2, pp228-233.

[2] Hastings, T. E. & Sajeev, A. S. M. (2001), “A Vector-Based Approach to Software Size Measurement and Effort Estimation”, IEEE Transactions on Software Engineering, Vol. 27, No. 4, pp.337-350.

[3] Norris, K. P. (1971), “The Accuracy of Project Cost and Duration Estimates in Industrial R&D”, R&D Management, Vol. 2, No. 1, pp.25-36.

[4] Murmann, Philipp A. (1994), “Expected Development Time Reductions in the German Mechanical Engineering Industry”, Journal of Product innovation Management, Vol. 11, pp.236-252.

[5] David Consulting Group (2012), “Project Estimating”, DCG Corporate Office, Paoli, 2007: http:davidconsultinggroup.com/training/estimation.aspx (January, 2017)

[6] Boehm, Barry (1976), “Software Engineering”, IEEE Transactions on Computers, Vol. C-25, Issue 12, pp1226-1241.

[7] Dreger, J. B. (1989), “Function Point Analysis”, Englewood Cliffs, NJ:Prentice-Hall.

[8] Smith, Randy K., Hale, Joanne E. & Parrish, Allen S. (2001), “An Empirical Study Using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation”, IEEE Transactions on Software Engineering, Vol. 27, No. 3, pp.264-271.

[9] Sataphthy, Shashank Mouli, Kumar, Mukesh & Rath, Santanu Kumar (2013), “Class Point Approach for Software Effort Estimation Using Soft Computing Techniques, International Conference on Advances in Computing, Communications and Informatics(ICACCI), p178-183.

[10] Tariq, Sidra, Usman, Muhammad, Wong, Raymond, Zhuang, Yan & Fong, Simon (2015), “On Learning Software Effort Estimation”, 3rd International Symposium and Business Intelligence, P79-84.

[11] Bhandari, Sangeeta (2016), “FCM Based Conceptual Framework for Software Effort Estimation”, International Conference on Computing for Sustainable Global Development, pp2585-2588.

[12] Moharreri, Kayhan, Sapre, Alhad Vinayak, Ramanathan, Jayashree & Ramnath, Rajiv (2016), “Cost-Effective Supervised Learning Models for Software Effort Estimation in Agile Environments”, IEEE 40th Annual Computer Software and Applications Conference, p135-140.

[13] Mukhopadhyay, Tridas & Kekre, Sunder. (1992), “Software Effort Models for Early Estimation of Process Control Applications”, IEEE Transactions on Software Engineering, Vol. 18, No. 10, pp.915-924.

[14] Boehm, Barry W. (1995), “Cost Models for Future Software Life Cycle Processes: COCOMO 2.0,” Anals of Software Engineering Special Volume on Software Process and Product Measurement, Science Publisher, Amsterdam, Netherlands, 1(3), p45-60.

[15] Srinivasan, Krishnamoorthy & Fisher, Douglas (1995), “Machine Learning Approaches to Estimating Software Development Effort”, IEEE Transactions on Software Engineering, Vol. 21, No. 2, pp126-137.

[16] Strike, Kevin, Emam, Khaled EI & Madhavji, Nazim (2001), “Software Cost Estimation with Incomplete Data”, IEEE Transactions on Software Engineering, Vol. 27, No. 10, pp215-223.

[17] Putnam, Lawrence H. (1978), “A General Empirical Solution to the Macro Software Sizing and Estimating Problem”, IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, pp345-361.

[18] Boehm, Barry W. (1981), “Software Engineering Economics”, Englewood Cliffs, NJ:Prentice-Hall.

[19] Subramanian, Girish H. & Breslawski, Steven (1995), “An Empirical Analysis of Software Effort Estimate Alternations”, Journal of Systems Software, Vol. 31, pp135-141.

[20] Boehm, Barry W. (1984), “Software Engineering Economics”, IEEE Transactions on Software Engineering”, Vol. 10, pp4-21.

[21] Agrawal, Priya & Kumar, Shraddha (2016), “Early Phase Software Effort Estimation Model”, Symposium on Colossal Data Analysis and Networking, pp1-8.

[22] Albrecht, Allen. J. (1979), “Measuring Application Development Productivity”, Proceedings of the IBM Applications Development Symposium, pp83-92.

[23] Albrecht, Allen J. & Gaffney. John E. (1983), “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE Transactions on Software Engineering, Vol. 9, No.6, pp639-648.
[24] Hu, Qing, Plant, Robert & Hertz, David (1998), “Software Cost Estimation Using Economic Production Models”, Journal of Management Information Systems, Vol. 15, No. 1, pp143-163.
[25] Bock D. B., & Klepper R. (1992). FP S: A Simplified Function Point Counting Method, “The Journal of Systems Software”, 18:245 254.
[26] Kemerer, Chris F. (1993). “Reliability of Function Points Measurement: A Field Experiment”, Communications of the ACM, 36(2):85 97.
[27] Lokan, Chris J. (2000). “An Empirical Analysis of Function Point Adjustment Factors”, Journal of Information and Software Technology, vol. 42, pp649-660.
[28] Jeffery, J., Low, G. & Barnes, C. (1993), “Comparison of Function Point Counting Techniques”, IEEE Transactions on Software Engineering, Vol. 19, No. 5, pp529-532.
[29] Misra, A. K. & Chaudhary, B. D. (1991), “An Interactive Structured Program Development Tool”, IEEE Region 10 International Conference on EC3-Energy, Computer, Communication and Control Systems, 3, 1-5.
[30] Kendall, K. E. & Kendall, J. E. (2005), “System Analysis and Design”, 6/e, Prentice-Hall.
[31] Brooks, F. (1975), “The Mythical Man-Month.” Addison-Wesley.
[32] Zhang, Xiaoni & Windsor, John (2003). “An Empirical Analysis of Software Volatility and Related Factors”, Industrial Management & Data Systems, Vol. 103, No. 4, pp275-281.
[33] Kemerer, C. F. & Slaughter, S. (1997). “Determinants of Software Maintenance Profiles: An Empirical Investigation”, Journal of Software Maintenance, Vol. 9, pp235-251.
[34] Krishnan, Mayuram S. (1998). “The Role of Team Factors in Software Cost and Quality”, Information Technology & People, Vol. 11(1), pp20-35.
[35] MacDonell, S. G., Shepperd, M. J. & Sallis, P. (1997), “Metrics for Database Systems: An Empirical Study”, Proceedings of the 4th International Software Metrics Symposium(Metrics 1997).
[36] MacDonell, S. G. (1994). “Comparative Review of Functional Complexity Assessment Methods for Effort Estimation”, Software Engineering Journal, pp107-116.