An Empirical Study of Effort Estimation Using Simplified Functional Point Analysis

Bingchiang Jeng[a]*, Dowming Yeh[b], Deron Wang[c], Shu-Lan Chu[b]
[a]*Department of Information Management, National Sun Yat-sen University,*
70 Lien-hai Rd., Kaohsiung 804, Taiwan
Email: jeng@mail.nsysu.edu.tw
[b]*Information and Computer Education Institute, National Kaohsiung Normal University,*
116 Hou-Ping 1st Road, Kaohsiung 802, Taiwan
Email: dmyeh@nknucc.nknu.edu.tw, csl@icemail.nknu.edu.tw
[c]*Department of Information System, China Steel Corporation,*
1 Chung Kang Road, Kaohsiung 812, Taiwan
Email: deronwan@mail.csc.com.tw

ABSTRACT

An important task in software development is to estimate the effort needed for developing software, which is the major cost of a software project. The success of a project greatly depends on the accuracy of estimation. Although a variety of estimation models have been proposed, practitioners still experience difficulties in applying these methods to modern applications because most models attempt to cover all possible factors related to size, and therefore effort. As a result, these generic models are too complicated or difficult to be implemented, which becomes an obstacle for their acceptance by practitioners. We present a different approach in this paper: instead of developing a generic model that is more comprehensive and complete, we simplify and tailor a popular estimation model, Function Point Analysis (FPA), so that it is specific and incorporate more of an application's characteristics. The empirical study shows that the accuracy of the simplified model is comparable with that of the full-fledged FPA.

Keywords: Empirical study, Function point analysis, Size estimation, Effort estimation, project planning

1: INTRODUCTIONS

Size and effort estimation are major inputs to most project estimation tasks within a software development process. They serve as the reference mark for cost estimation, schedule planning, and so on [32][38][39][41]. Until recently, however, most approaches to size or effort estimation still rely on senior domain experts who conduct estimation based on experiences of similar projects in the past [22]. The problem with such an approach is that the quality of estimation depends heavily on personal experiences and subjective judgments. This usually results in poor estimation accuracy. The consequence is widespread cost and schedule overruns as is commonly seen.

Size and effort are strongly related. Once the size of a project is determined, the effort to build the software is calculated either by a straightforward conversion through the average productivity of an organization or an empirical model [4][5][9][10][11][20][45]. These empirical models try to formalize as many influencial factors as possible and calculate the effort estimation in a pre-defined equation based on them. But practitioners still experience difficulty in applications of these models due to the existence of many factors (e.g., team experience, software architecture, implementation tools, application types, etc.) that may affect the development effort differently but are not necessarily fully reflected in the models [14][17][19][29][30][43][44].

On the other hand, most existing models are also too complicated or difficult to be implemented, which further hinders their acceptance to practitioners [2][3]. To deal with this problem, it is not a good idea to invent yet another sophisticated estimation model to make it more complete but to seek for an approach that could directly reduce the effect of application related hidden factors. The reason is simply that most of these factors are inherent to an organization or a project and cannot be generalized; thus a generic model can do no help. Interested readers are referred to Kitchenham's paper for further comments [31].

In this paper, we will present an approach to develop a simplified estimation model using historical data of an organization and some metrics related to application types, system architecture, and so on. The proposed estimation model would produce effort estimation directly without the intermediate size estimation step since what really matters to a software development organization is the effort needed for a project. Size estimation is necessary for generic models since they need to be applied across various organizations and the size can represent an organization-independent measure; whereas effort estimation are closely related to a specific organization.

The simplified estimation model is inspired by Function Point Analysis (FPA), which is a very popular generic estimation method [4][5][13][23]. FPA is a size estimation method that decomposes a software system, according to its functional requirements, into various subcomponents repeatedly until they are suitable for analysis. Each component then receives a function point value based on its complexity and function type, and the summation of them is multiplied with an influence adjustment factor to generate the final function points. The resulting function points then serve as an input to some regression model to produce effort estimation. The usage of FPA is advocated by many experts [16]. FPA has advantages over approaches based on lines of code since it is independent of technology and implementation. However, its application relies upon well-trained experienced experts to judge and estimate the various aspects of a software application [2][3]. Thus, it is difficult for a common project team to adopted FPA .

The remainder of the paper is organized as follows. Section 2 briefly introduces the history of function point analysis. Section 3 presents the simplified definition of function points and the regression model. The regression model is established in Section 4. An empirical study that applies this method to a real application and compares the result with FPA is shown in section 5. The final section is a discussion on what we found in this study as well as its future research directions.

2: EFFORT ESTIMATION

Size estimation is a critical factor to the success of a software project [26]. Different techniques have been presented in literature [14][17][20][45] and the most popular approaches classified are: algorithmic and parametric models, expert judgment, and reasoning by analogy [7]. Other less formal method include price-to-win, top-down, bottom-up, rules of thumb, and available capacity. According to Heemstra's survey, a total of twenty-nine software cost models have been proposed since 1966 [20].

In addition, machine learning models for size estimation are also introduced recently [37][42]. The approaches include case-based reasoning [8][15][40], fuzzy logic [33][35], neural network [18][27], and many others [21][36]. Most of the approaches report comparable results to other techniques [17]. However, due to the lack of formal data in the early stages of the software development process, most of these models apply to only latter stages of the software life cycle.

Function point analysis belongs to the algorithmic approach of size estimation, which is sometimes called a parametric or statistic approach. This type of methods evaluates two classes of attributes of a software project, i.e., size factors and influence factors, separately. FPA, invented by Albrecht at IBM in 1979, is probably the first of such a model [4]. In his work, a new metric for software size rather than lines of code is proposed. According to the definition of International Function Point User Group (IFPUG) [12], a function point is to "...measure software size by quantifying the functionality provided to the user based solely on logical designs and functionality specifications." [12] Since the functionality of a software system from a user's perspective is usually given in the early days of the project, FPA has a unique advantage of being applicable in the early stage when other approaches to size measurement are not appropriate. This measurement result then facilitates evaluation, planning, management, and control of projects.

In the FPA model, functions of a software system are classified into five types: External Inputs (EI) – unique user data or control input that adds or changes data, External Outputs (EO) – unique user data or control output that leaves the boundaries of the system, External Inquiries (EQ) - unique input that generates immediate output, Internal Logical Files (ILF) – internally maintained logical group of data, and External Interface Files (EIF) – file passed or shared between applications. IFPUG further groups these function types into two categories: data function type and transactional function type. The former includes internal logical files and external interfaces file, while the latter includes external inputs, external outputs, and external inquiries. Within each category, different function elements are identified.

The IFPUG suggested steps for function point analysis are the following: 1) identifying the function elements existing in each prescribed function, 2) assigning each function a ranking level of simple, medium or complex, based on the number of function elements found, 3) calculating an initial function point count, 4) determining the total degree of influence on the general system characteristics, and 5) calculating the final function points based on the initial point count and the influence adjustment factor.

The function elements are those described above (i.e., DET, RET, and FTR), but it takes effort to identify and count their occurrences in each classified function category in the first step. In the next step, the number of found function elements is used to determine the rank of a function to be simple, medium, or complex in the classified category according to a complexity ranking table. FPA has also pre-defined the complexity weighting factor table that associates each level of complexity with a numerical constant, ranging from 3 to 15, based on their relative weight and complexity over other function types. With this table, a rank value is determined for each individual function category and multiplied by the function point in that category. The results are summed up to generate the initial unadjusted function point count as is required in the third step of the process.

Since FPA is designed to be generic and across organizations, the analysis continues by considering general environmental factors. Currently, fourteen general system characteristics are identified, which are: data communications, distributed functions, performance objectives, heavily used configuration, transaction rate, on-line data entry, end-user efficiency, on-line update, complex processing, reusability, installation ease, operational ease, multiple sites, and facilitate change. Depending on the degree of influence, each characteristic receives a rank from zero to five. The total rank summation of the fourteen factors is multiplied by 0.01 and added with a constant 0.65, which determines the value adjustment factor (VAF). The previous function point value is then multiplied with this VAF to get the final value.

In addition to the original model, a variety of other FPA models exist. Since the first revision of FPA in 1984, a number of variations and extensions have sprung from the original model [24]. Symons introduces Mark II Function Point in 1991 [44]; Whitmire introduces 3D Function Point in 1992 [46]; Abran et al. introduces Full Function Point in 1997 [1]. Other related models include Object Point Analysis and Feature Point Model [6][25]. The current version of FPA is published by the International Function Point User Group (IFPUG) with version 4.1 in 1994.

Almost all effort estimation models are derived empirically, that is, by using regression analysis on historical data from past projects. These models predict effort with the size as the major input parameter. The inputted size measure is based on LOC for some models, for example, COCOMO II by Bohem [11]. Most LOC-based models are non-linear, that is, effort is exponentially related to size. Other models are based on FP, including one proposed by Albrecht and Gaffney [5]. In most FP-based models, however, effort is linearly related to size. The result of FPA can still be used by the LOC-based models by converting FP to LOC, a technique called backfiring. IFPUG provides a table for mapping FP to LOC for various programming languages.

3: SIMPLIFIED FPA MODEL

Although FPA is popular and advantageous over many methods, it is difficult to adopt FPA in modern development environment (for example, real-time, OO, business, scientific, and so on) and that is why various extension models were proposed. In particular, Symons has noted that “the concept of (Albrecht) logical file is almost impossible to define unambiguously, particularly in a database environment …” [44] The coefficients in the complexity weighting factor table has to be justified too before implementation, as noted by Kitchenham [31]: “there are no standard conversion factors to equate … In addition, it is unclear that such weights are appropriate for new systems as well as system enhancements.” The third problem is the determination of the influence adjustment factor. Empirical studies show that each estimator has different considerations on the influence factor of the general system characteristics, and thus the final function points from different estimator may vary at worst 30% within an organization and more when across organizations [23][28][29].

In a specific application where historical project data exist, we think a simpler approach can avoid or alleviate the above problems, instead of adopting the full-fledged FPA models. Our idea is to cultivate the model so that it is more specific to directly reduce the effect of unrelated factors related to development effort in an application domain and more distinct so that the variance of subjective judgments in influence adjustment factors may be mitigated. The approach is manifested in three steps. The first step is to reduce the gap between the FPA’s classification of function types and a real application. The function complexity of an application is usually not fully represented by the five basic function types and function elements. In addition, counting the function elements is too low level to be implemented and may need certain level of expertise. So the first step is to re-evaluate the basic function types with respect to the given application domain to enhance the classification with a higher level of meaningful complexity indicators.

The second step is to incorporate the general system characteristics into the function point counting in order to reduce possible subjective bias. Since our attempt is not to develop a generic model as other FPA models do, there is no need to separate the consideration of function complexity measurement into two parts. Such rationale is supported by Kemerer[27], Low and Jeffery [34]. Their studies show that effort predictions based on raw function points are just as accurate as predictions based on adjusted function points. The abandonment of adjustment is further suggested by Kitchenham [31]. So we simply set the value of the influence adjustment factor equal to 1 and ignore it later.

This instead means that the complexity weighting factor table has to be dynamically adjusted according to system characteristics of different applications. So the final step is a regression analysis that will re-create the complexity weighting factor table and determine its coefficients based on the given historical data. In the following paragraphs, we will use an application example to explain these steps in detail.

There are five categories in the original function types in FPA. There is nothing wrong with the basic classification, but within a specific application domain, other factors may exhibit stronger influence over the effort required for developing software than these five generic dimensions. The study that we choose for practice is an OLTP (On Line Transaction Processing) type of applications within a large ERP system running by a local steel company; it is implemented with COBOL on a database system. After discussion with the senior engineers in the management of information system department, six most influential factors that may affect the functional complexity in their daily programming jobs are identified. The factors and their correspondences with the original FPA are discussed as follows:

1. Inter-communication between programs: A common feature of the company’s OLTP programs is that many programs call each other in order to complete a specific task. These programs usually exchange a certain form of complicated parameters during communication. Thus the number of inter-communication parameter sets reflects the complexity of the task a program performs. This

function category corresponds to the concept of external interface files and internal logical files in FPA.

2. Connected subsystems: Depending on its business purpose, a program can be individually connected to either the sales management system, the production management system, the logistic and data warehouse system, the device management system, the administration system, or the financial management system, respectively. It is clear without saying that the more subsystems a program is involved with, the higher functional complexity it carries. At the least, it will take software engineers more time to study the connected systems and their internal data elements. This function category is a little similar to the first one, but it puts more emphasis on the aspect of general system characteristics covered by VAFs in FPA, which includes data communications, distributed functions, complex processing, and multiple sites.

3. Updated files: A good practice for business application programs in physical implementation is to keep a log for inserted, deleted, or modified data for the purpose of failure recovery or auditing trails. This increases the load and complexity of a program. This function category corresponds to the concept of internal logical files and external interface files in FPA.

4. User departments involved: A special characteristic of the systems in our study is that most of the applications have similar user interface and the more user departments are involved, the more complicated is a user screen. In addition, more users also implicitly imply more complicated function requirements. This function category corresponds to the concept of external inputs, external outputs, and external inquiries in FPA.

5. Utility programs: The utilities are those programs implementing code tables, data checking or validation modules, or certain business rules, etc. So the more utilities are invoked, the more a program executes the data transformation functions. This function category corresponds to the concept of external interface files in FPA.

We can see that the spirit of the new function categories still follows the FPA's original classification. But the levels of emphasis are quite different between the two classifications. Moreover, our categories carry more functional complexity information related to the particular application domain than the original basic function types. In addition, it is more natural and easier to the project team since these function categories are what they are familiar with. The new function category is more direct and does not suffer from such problems. In addition, the counting of function elements is easier, as shown in a later example. Since the new function types are already the most basic elements with meaning, there is no need to further divide them unambiguously.

In the FPA model, the complexity weighting factor table is important in order to count the function points of each individual function. Since we have redefined the basic function types according to the given application domain, we need to re-build the table and determine its coefficients. The method is by regression analysis of the historical data.

4: ESTABLISHMENT OF THE MODEL

We choose a software team whose daily jobs are to develop new functions for existing systems under requests from other departments. In order to make data analysis easier and clearer, we decide to apply the effort estimation in only two phases of the software development process, which are the implementation and the testing phases. The reason is, at that moment, the function requirements and the overall design of programs are well known so that the counting of function points is easier and more certain. The analysis, however, can be extended to the whole life cycle since the cost of these two phases over the total development cost in that department is roughly a constant ratio with a little variance.

The training data for the regression analysis consist of 41 records of small programming or testing tasks, which were collected from projects for two application systems: the first one (System B) has twenty-one records and the other (System C) twenty records. Each of the records is evaluated with respect to the classified function categories and assigned with a rank of low, medium, or high functionality. If the complexity of a function type (i.e., the number of functions) falls below 30% in its corresponding complexity spectrum (calculated from the whole training data set), it is classified as low complexity; if it is greater than the first 30% but less than 70%, it has medium complexity; otherwise, it has high complexity.

In the regression analysis, the Durbin-Watson (DW) test is employed to examine that the errors of independent variables should not be self-related. From our analysis, the DW value of the model is 2.19. This reveals that there is no significant self-relation in the model and the prediction of the models can be trustworthy. F test examines the overall regression model, also called as Analysis of Variance (ANOVA). The F value of the analysis result is 505.45, and the P-Value is less than 0.005. Therefore, the linear relationships of our models are well established. The coefficient of multiple determinations measures the proportion that independent variables are able to explicate the dependent variable. Another related measure is the adjusted coefficient of determination, which is considered more representative than the coefficient of determination. Applying these measures, the coefficient of multiple determination and adjusted coefficient of determination for the model are 0.986 and 0.984, respectively. This indicates that the independent variables in the model can account for around 98 % of the dependent variable.

We iterate the regression analysis process to determine the best values of W_i that will minimize the total error summation. After the iterative process, it converges and the final values are given in Table 1.

Table 1 Convergence of weight control factors

Category	W_i	A_i
Inter-communication parameter sets	0.133	7/10/14
Connected subsystems	0.1219	13/17/26
User departments	0.125	10/14/20
Connected utility programs	0.1153	3/4/6
Updated files	0.1066	6/9/13

5: RESULTS AND COMPARISON

Since our estimation model is based on man-day data, it is essential to convert IFPUG function point to such metrics. Again, we use System B and System C as training data to find the average man-days for a IFPUG function point. The computed function points are shown in Table 2. The average man-days for a function point are therefore taken to be the average of the data from two systems, 0.2796.

Table 2. Function points and man-days conversion

	System B	System C
Unadjusted function points	404	397
Value adjustment factors	34	33
Function points	400	389
Actual man-days	126	95
Man-days per function point	0.315	0.2442

Finally, the average man-days for a function point is used to estimate the man-days for system A and the result is compared that of the regression model. The function point value of System A is 111, and the estimated man-days are 31.0356 (111 multiplied by 0.2796). The actual man-days for System A development are 32. The accuracy of the IFPUG estimation is therefore 96.99%. The estimation of our regression model for the System A, on the other hand, is 33.6485. This results in an accuracy of 94.85%. The accuracies of both methods are about the same.

Backfiring is a mathematical conversion between source lines of code (SLOC) and function point counts. Although SLOC is not considered an rigid metrics, it is still commonly used in the industry, particularly for maintenance and reengineering projects. To compute SLOC from our model, the average SLOC corresponding to a man-day need to determined.

Once again, the average SLOC per man-day are determined by data from system B and C, and the value is 379.48. If we use this value to estimate the size of System A, the estimated SLOC are 12769; while the actual SLOC are 10230, making the accuracy of the backfiring around 75.18%.

6: CONCLUSIONS

Although effort estimation is a critical step to the success of a software project, many projects are still relying on human experts to conduct this task. Two reasons explain this phenomenon. The first is that most of the current algorithmic estimation models are too hard or too complicated to be accepted by common users. The second is these models usually fail to reflect the most essential factors in a specific domain by considering all possible factors that may affect the software complexity and thus estimation accuracy varies.

This paper presents a different approach by trading “generic” with “specific”. Instead of using an even more generic model that is more comprehensive and complete, we simplify and tailor the estimation model so that it is more specific and fits the application domain. We believe that only when the estimation model is tightly rooted in a given domain, the problem of too many factors issues can be removed or mitigated.

Our approach first re-evaluates the function type category defined in the FPA model based on the application domain and system architecture. This gains two benefits. One is to make the function types more suitable for the application domain and more accurately reflect its function complexity; the other is to make the counting of function points easy for software engineers themselves instead of by a certified FPA expert since the classification and evaluation required here are familiar to them.

The complexity weighting factor table is constructed based on the newly defined function categories. By the process of regression analysis, we have shown how to construct a model and test its convergence. In addition, we have also shown how to use the sensitivity test to identify an ill-defined function type and then combine or remove it.

The study of this approach turns out to be quite successful as compared to the results from FPA. This strengthens our belief that a generic model trying to satisfy every situation (that is, applications for all types of software and environments) may not be necessary for a specific application. And we believe the key of this success in our study is the better classified function categories with easily comprehensible meaning and less ambiguity.

REFERENCES

[1] Abran, A., Maya, M., Desharnais, J. M., St-Pierre, D., 1997. Adapting Function Points to Real-Time Software. American Programmer 10 (11), 32-43.

[2] Abran, A., Robillard, P. N., 1994. Function points: a study of their measurement processes and scale transformations. Journal of Systems and Software 25, 171-184.

[3] Abran, A., Robillard, P. N., 1996. Function points analysis: an empirical study of its measurement processes. IEEE Trans. on Software Eng. 22, 895-909.

[4] Albrecht, A. J., 1979. Measuring application development productivity, Proceedings. In: IBM GUIDE/SHARE Applications Development Symposium, California.

[5] Albrecht, A. J., Gaffney Jr., J. E., 1983. Software function, source lines of code, and development effort prediction: a software science validation. IEEE Trans. on Software Eng. 9(6), 639–648.

[6] Antoniol, G., Fiutem, R., Lokan, C., 2003. Object-oriented function points: An empirical validation. Empirical Software Eng. 8 (3), 225-254.
[7] Barry, B., Abts, C., Chulani, S., 2000. Software development cost estimation approaches — a survey. Annals of Software Engineering 10 : 177-205.
[8] Bisio, R., Malabocchia, F., 1995. Cost estimation of software projects through case-base reasoning. In: Case-Based Reasoning Research and Development. First International Conference, ICCBR-95 Proceedings 11-22.
[9] Boehm, B. W., 1981. Software Engineering Economics., Prentice-Hall, New Jersey.
[10] Boehm, B. W., Papaccio, P. N., 1988. Understanding and controlling software costs. IEEE Trans. on Software Eng. 14(10), 1462–1477.
[11] Bohem, B. W., 1996. Anchoring the software process. IEEE Software, 13(4), pp. 73-82.
[12] Boehm, R., 2002. Frequently Asked Questions. www.ifpug.org, 2002.
[13] Dreger, J. B., 1989. Function Point Analysis. Prentice-Hall, New Jersey.
[14] Fenton, N. E., Neil, M., 1997. Software metrics: success, failures and new directions. Journal of Systems and Software 47, 149-157.
[15] Finnie, G. R., Wittig, G. E., Desharnais, J. M., 1997. Estimating software development effort with case-based reasoning. In: Proceedings of International Conference on Case-Based Reasoning, D. Leake, E. Plaza, (Eds) 13-22.
[16] Furey, S., 1997. Why we should use function points. IEEE Software 14 (2), 28, 30.
[17] Gray, A. R., MacDonell, S. G., 1997. A comparison of techniques for developing predictive models of software metrics. Information and Software Technology 39, 425-437.
[18] Hakkarainen, J., Laamamen, P., Rask, R., 1993. Neural networks in specification level software size estimation. In: P.K. Simpson (Ed.), Neural Network Applications, IEEE Technology Update Series, 887–895.
[19] Hakuta, M., Tone, F., Ohminami, M., 1997. A software size estimation model and its evaluation. Journal of Systems and Software 37, 253-263.
[20] Heemstra, F., 1992. Software cost estimation. Information and Software Technology, October, 627-639.
[21] Heiat, A., 2002. Comparison of artificial neural network and regression models for estimating software development effort. Information and Software Technology 44, 911-922.
[22] Hughes, R. T., 1996. Expert Judgment as an Estimating Method. Information and Software Technology, 67-75.
[23] Jeffery, D. R., Low, G. C., Barnes, M., 1993. Comparison of function point counting techniques. IEEE Trans. Software Eng. 19 (5), 529-532.
[24] Jones, C., 1988. A short history of function points and features points. Software Productivity Research Inc.
[25] Jones, C., 1991. Applied Software Measurement: Assuring Productivity and Quality, McGraw-Hill, New York.
[26] Jorgensen, M., Sjoberg, D. I. K., 2001. Impact of effort estimates on software project work. Information and Software Technology 43, 939-948.
[27] Karunanithi, N., Whitley, D., Malaiya,Y. K., 1992. Using neural networks in reliability prediction. IEEE Software 9 (4), 53-59.
[28] Kemerer, C. F., 1987. An Empirical Validation of Software Cost Estimation Models. Comm. ACM 30 (5), 416-429.
[29] Kemerer, C.F., 1993. Reliability of function point measurement: a field experiment. Comm. ACM 36(2), 85-97.
[30] Kitchenham, B., Pfleeger, S. L., Fenton, N. E., 1995. Towards a Framework for Software Measurement Validation. IEEE Trans. Software Eng. 21 (12), 929-944.
[31] Kitchenham, B., 1997. The Problem with Function Points. IEEE Software 14 (2), 29-31.
[32] Lederer, A. L., Prasad, J., 1992. Nine Management Guidelines for Better Cost Estimating. Comm. ACM 35(2), 51-59.
[33] Lima, O. D., Farias, P. M., Belchior, A. D., 2003. Fuzzy modeling for function points analysis. Software Quality Journal 11 (2), 149-166.
[34] Low, G. C., Jeffery, D. R., 1990. Function Points in the Estimation and Evaluation of the Software Process. IEEE Trans. Software Eng. 16 (1), 64-71.
[35] MacDonell, S. G., 2003. Software source code sizing using fuzzy logic modeling. Information and Software Technology 45, 389-404.
[36] MacDonell, S. G., Shepperd, M. J., 2003. Combining techniques to optimize effort predictions in software project management. Journal of Systems and Software 66, 91-98.
[37] Mair, C., Kododa, G., Lefley, M., Phalp, K., Schofield, C., Shepperd, M., Webster, S., 2000. An investigation of machine learning based prediction systems. Journal of Systems and Software 53, 23-29.
[38] Osmundson, J. S., Micheal, J. B., Machniak, M. J., Grossman, M. A., 2003. Quality management metrics for software development. Information and Management 40, 799-812.
[39] Pressman, R. S., 2001. Software Engineering, 5/e. McGraw-hill, New York.
[40] Prietula, M., Vicinanza, S., Mukhopadhyay, T., 1996. Software effort estimation with a case-based reasoner. Journal of Experimental and Theoretical Artificial Intelligence 8 (3-4), 341-363.
[41] Sommerville, I., 2000. Software Engineering, 6/e. upplagan , Addison-Wesley.
[42] Srinivasan, K., Fisher, D., 1995. Machine learning approaches to estimating software development effort. IEEE Trans. on Software Eng. 21 (2), 126–137.
[43] Symons, C. R., 1988. Function point analysis: difficulties and improvements. IEEE Trans. on Software Eng. 14 (1), 2–10.
[44] Symons, C. R., 1991. Software Sizing and Estimating -- MkII FPA (Function Point Analysis), John Wiley & Sons, Chichester, UK.
[45] Tate, G., Verner, J. M., 1990. Software sizing and costing models: a survey of empirical validation and comparison studies. Journal of Information Technology 5, 12–26.
[46] Whitmire, S. A., 1992. 3-D function points: scientific and real-time extensions to function points. In: Proc. of the 1992 Pacific Northwest Software Quality Conference.