A Method for Calculating Acknowledged Project Effort Using a Quality Index

Marjan Heričko and Aleš Živkovič
University of Maribor, Faculty of Electrical Engineering and Computer Science
Smetanova 17, SI-2000 Maribor, Slovenia
E-mail: {marjan.hericko, ales.zivkovic}@uni-mb.si

Zoltán Porkoláb
Eötvös Loránd University, Faculty of Informatics
Pázmány Péter sétány 1/C , H - 1117 Budapest, Hungary
E-mail: gsd@elte.hu

Keywords: project effort, product metrics, software size, function points, object-oriented metrics

Received: August 11, 2007

Software size is the fundamental metric for project planning. Effort and duration are calculated based on the size estimate. However, for a given software size, the actual development effort could be significantly different. The question is whether the increase in effort is due to the low productivity of the development team or higher product quality. While higher product quality is highly desirable and usually worth investing in, the reasons for additional effort might be elsewhere. In the research presented in this paper, the focus is on the correlation between code quality and productivity. Code quality is only one aspect of product quality. This paper presents a method for calculating a new type of project effort named "acknowledged effort". Acknowledged effort is calculated based on the actual effort and code quality. This new type of effort reflects not only the project's size and the productivity of the development team, but also the quality aspect of the delivered software system.

Povzetek: V prispevku je analizrana korelacija med kakovostjo programske kode in produktivnostjo.

1 Introduction

Software size is an elementary measure often used to calculate project effort, costs, productivity and duration. In practice, the actual effort measured during the project development time could be significantly different although the estimated project size is the same. The effort is influenced by several factors like the complexity of the solution, development team size, development platform, etc. In this research, the focus is on the code quality that could as well influence the project total effort. The quality is defined as the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. [10]. In terms of measures, it is a collection of metrics that cover categories like functional correctness, maintainability, efficiency, portability, usability and dependability [5,7,8,12]. The number of metrics used to determine product quality is not well defined and it could range from just a few to a hundred or more [2,12]. In contrast, there is the idea of a single number that express quality - the quality index (QI). The quality index is based on the 20/80 rule. According to some findings, 20% of variables can capture 80% of the intrinsic quality. The second principle behind the quality index is consistency and repeatability. If we perform the same procedure over and over again it will provide us with insight into product quality, regardless of its absolute accuracy in general applications. However, the standard deviation of the accuracy should be low in order to get valuable results. In this research, the code quality is measured in order to justify the deviations in project effort. Besides productivity, also code quality should be evaluated when comparing the performance of the development teams. The general functional relation between the productivity and software size is [1]:

$$P = \frac{E}{S} \qquad \text{(Eq. 1)}$$

where E is the actual effort spent developing some functionality and S is the total size of the functionality in question. For software size estimation, different methods are used [1,9,14,15], all of which have their roots in the Function Point Analysis (FPA) method.

The main contribution of the research presented in this paper could be summarized as:

1. The identification of the minimal object-oriented source code metrics set that might compose the quality index (QI) used together with software size in function points.
2. The definition of acknowledged effort that combines a team's productivity with the code quality. The actual project effort is then compared to the acknowledged effort in order to evaluate project results.

The research is based on the following assumptions and restrictions:

- the delivered code is complete and fully functional
- financial results, restrictions and influences are not considered in the proposed method
- the mean value for the productivity of some data set is valid for average code quality e.g. the projects are of different quality, however if the number of projects in the data set is big enough the data set would represent the average code quality.

This paper is divided into six sections. In the next section, the FPA method is briefly presented. The product metrics are introduced in section three. The main idea and the proposed method can be found in section four. The last section summarizes the findings related to a set of object-oriented projects and discusses the potential direction for future work.

2 Size Estimation for Object-Oriented Projects

Albrecht [1, 9] introduced the Function Point Analysis (FPA) method in 1979. Since then, it has become the most important method for software size estimation. The method introduced a specific way of representing a software system and distinguished between data functions and transactional functions. Data functions (DF) are further divided into internal and external logical files (ILF and EIF) assigning different weights to each data function type. The transactional functions (TF) describe functionality through three abstract types, namely: external inputs (EI), external outputs (EO) and external inquiries (EQ). To be able to determine the contribution of the FPA element (ILF, EIF, EI, EO or EQ) to the final estimated size value, the complexity is assigned to each element. The complexity is determined by the number of simple data elements named Data Element Type (DET) or structured elements named Record Element Types (RET). To get actual values in Function Points (FP), the tables defined in the method are used. The FPA abstraction concept is easily applied to structured analysis and design artefacts. The mapping of entities attributes and processes to FPA elements is straightforward.

The method was intended for all domains, although in practice, its accuracy is different within different domains. From a practical standpoint, it can be concluded that the FPA method application is more difficult with object-oriented projects. The elements and constructs of the FPA method are not directly applicable to object-oriented concepts also used within the Java and .NET development platforms. Therefore, a mapping of object-oriented concepts into FPA elements is needed. The mapping is not defined within the FPA method itself and is consequently not uniform. Different authors have proposed different mapping functions [3, 14, 15, 17], mostly in the form of additional rules. Information is gathered from different diagrams (e.g. Use Case diagrams, class diagrams, sequence diagrams)[14, 17] which are then considered separately. In one of our previous research, the OO-to-FPA mapping was defined and automated [17]. More detailed research of the FPA transformation tables has shown that the weight factors of the standard FPA method have to be calibrated for use in object-oriented projects [3, 4, 15].

3 Product Metrics for Object-Oriented Systems

In the software engineering community the term metric has been used in many distinct ways. For the purpose of this research, metrics are defined as a function, whose value is derived from a product, process or resource. It is important to distinguish between objective and subjective metrics. An objective metric is a function whose input is software data (elements) and whose output is a single numerical value. Subjective metrics, on the other hand, attempt to track less quantifiable data and usually depend on subjective judgment. When speaking about quality metrics the obtained metric value indicates the degree to which software possesses a given quality attribute. Therefore quality metrics are an indirect measure of software quality. We need validated metrics, metrics whose values have been proven to be statistically associated with corresponding software attributes. For object-oriented software the following metrics are often used [2, 13]:

- Weighted Methods per Class (WMC) - the sum of the complexities of the methods of a class (if all method's static complexities are considered to be unity, the number of methods). The number of methods and the complexity of methods involved are indicators of how much time and effort is required to develop and maintain the class. A large number of methods might limit the possibility of reuse since the class becomes too application specific.
- Depth of Inheritance Tree (DIT) - depth of the inheritance of the class. Inheritance through classes increases its efficiency by reducing the redundancy. However, the deeper inheritance hierarchy makes the behavior more difficult to predict and understand. There is no general threshold value for this metric. The threshold must be determined within the development team.
- Number Of Children (NOC) - the number of immediate sub-classes subordinated to a class in the class hierarchy. The greater the number of children in the inheritance hierarchy the greater the reuse. Then again a large number of children of a class might indicate improper abstraction for a parent class. In general the high DIT value and low NOC means better reusability but worse maintainability. It also has a negative impact on understandability and is more difficult to modify. Since there are no empirical or theoretical boundary values, the

developers should find the proper threshold value for the system under development.

- Response For a Class (RFC) - the sum of the number of its methods and the total of all other methods that they directly invoke. If the number of methods invoked in response to a message received by an object is large, the maintenance and testing are more demanding. Again there is no specific threshold value for the metrics.
- Coupling Between Objects (CBO) - the number of non-inheritance related couples with other classes (class is coupled with another if its methods use the attributes of the other class). The reusability of classes and/or subsystems is low when coupling between them is high, the system is also harder to understand. Normally a class should have a low coupling with the rest of the classes. A high coupling between different parts of a system has a negative impact on the modularity of the system and is usually a sign of poor design.
- Lack of Cohesion in Methods (LCOM) - the number of disjoint sets produced from the intersection of the set of attributes that are used by the methods reduced by the number of method pairs acting on at least one shared attribute. The LCOM metric is a value of the dissimilarity of the methods in the class. A high LCOM value in a class indicates that it might be a good idea to split the class into two or more sub classes. The metrics help identify flaws in the design of a program structure. The high LCOM values are associated with lower productivity, greater design and rework effort. The LCOM could be used as a predictor for maintenance effort.
- Method Hiding Factor (MHF) - sum of the invisibilities of all methods defined in all classes / total number of methods.
- Attribute Hiding Factor (AHF) - sum of the invisibilities of all attributes defined in all classes / total number of attributes
- Method Inheritance Factor (MIF) - sum of inherited methods / total number of available methods
- Attribute Inheritance Factor (AIF) - sum of inherited attributes / total number of available attributes
- Polymorphism Factor (POF) - actual number of possible different polymorphic situation / maximum number of possible distinct polymorphic situation
- Coupling Factor (COF) - actual number of couplings not imputable to inheritance / maximum possible number of couplings.
- Cyclomatic Complexity (CC) - in object-oriented design, the metrics represents the complexity of a method and indirectly also complexity of a class. The value should be as low as possible. The values between 10 and 20 are considered as an upper limit for metrics.
- Maintainability Index (MI) - predict the maintainability of the software combining several elementary metrics. Two versions are in use. The first version uses three elementary metrics to calculate the index and the second uses four metrics. The fourth metrics evaluates the average number of comments per class. However, it is not clear if the greater number of comments actually increases the ease of code maintenance. In this research the first version of the metrics will be used. The threshold values for MI are: MI<65 indicate poor maintainability, $65 \leq$ MI ≤ 85 fair maintainability and MI > 85 promises excellent maintainability [2].

In addition to the described metrics, some size-related metrics should also be considered. Table 1 summarizes the candidate metrics classified according to the class/method level.

Table 1: Size Related Metrics

Class Level	**Method Level**
number of methods	number of parameters
number of properties	number of local variables
number of constructors	number of exception blocks
number of nested classes	max stack size
number of data fields	number of instructions
number of events	number of all operators in the method
number of attributes	number of distinct operators
the number of all instructions	number of operands

In order to collect and analyze metric data of object-oriented projects, the tool in the Microsoft .NET framework 2.0 was developed. The input is arbitrary executable format for the Microsoft platform. The parser that is a part of the tool performs an analysis directly on the common intermediate language code as defined in the .NET framework. In addition to the metrics presented in section three (not all metrics are supported in this version), the tool collects the data presented in Table 1. The tool is also described in [18].

Based on the collected data and the code quality metrics presented in section three, the correlation between different metrics were investigated as well as their potential impact on project effort and code quality. Based on the findings, the subset of code metrics was selected. The selected metrics that are used for calculating quality index (QI) and acknowledged effort (E_{ACK}) are listed in the next section.

4 The Proposed Method

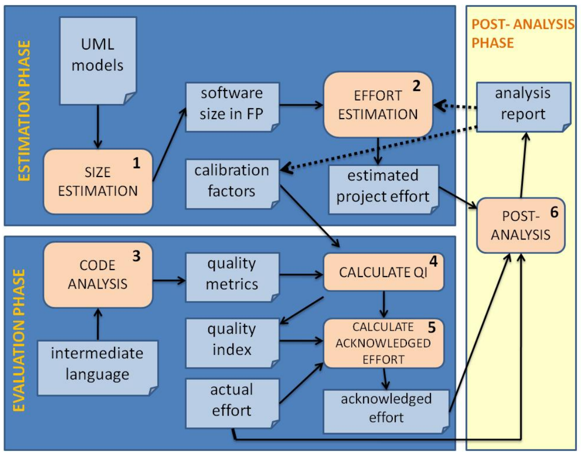

Figure 1: The Schematic View on the Proposed Method

Figure 1 shows the proposed method for calculating the acknowledged effort. In the analysis phase the project size and effort are calculated based on the UML models and projects characteristics. After the implementation phase the code analysis is performed. Based on the code analysis and actual effort the acknowledged effort is calculated. The acknowledged effort is defined as:

$$E_{ACK} = E_A * (RE_1 + (QI * RE_2)) \qquad \text{(Eq. 2)}$$

where E_{ACK} is the acknowledged effort expressed in hours, E_A is actual effort in hours, RE_1 and RE_2 are reward factors and QI is the quality index that has no unit. In our research the value for RE_1 is 0,7 and RE_2 is 0,1 which influences the actual effort for ±20 %.

The QI is defined as:

$$QI = \frac{\sum_{i=1}^{n} PMQR_i}{n} \qquad \text{(Eq. 3)}$$
$$PMQR \in \{1..5\}$$
$$PMQR_i = f_i(mv_i)$$

where *PMQR* is the product metric quality rating, *n* is the number of code metrics used in the calculation, *mv* is a code metric value and *f* is the function that transforms metric value for metric *i* to the product metric quality rating. Code metrics that were considered in this research are described in section two. The quality rating transformation function is defined for each metric individualy. For the purpose of this research *f* is a step function that is defined based on the individual metric threshold values. An example for the maintability index (MI) metric is shown in Figure 2.

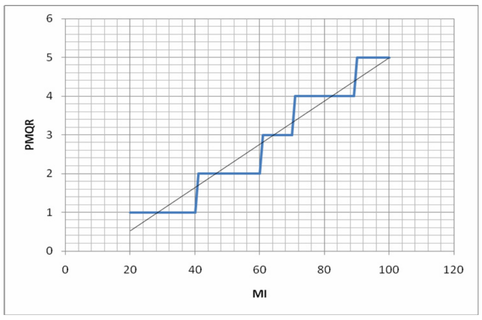

Figure 2: An example of the function f for the MI metric

QI is composed of *n* product metrics. The number of metrics and its type should be defined according to the project and environment characteristics. Each product metric has its threshold values. The threshold values are project specific and should be calibrated considering the folowing attributes:

- development team (experience level, team size, number of roles involved, etc.),
- development environment (platform, technology, process model, tools, etc.),
- domain (telecommunications, insurance, banking, etc.),
- customer (long term agreements, inhouse development),
- development type (off the shelf, research projects, critical systems, new development, reengineering, etc.).

In this research, the metrics described in section three were considered for selection. The narrowed list includes the following metrics:

- Depth of Inheritance Tree (DIT),
- Number of Children (NOC),
- Weighted Methods per Class (WMC),
- Coupling Between Objects (CBO),
- Response fo Classes (RFC),
- Lack of Cohesion in Methods (LCOM),
- Cyclomatic Complexity (CC) and
- Maintainability Index (MI)

The DIT and NOC make a complementary pair and should be considered as a pair [6, 7]. In this research only the DIT was used. WMC, CBO and RFC are highly correlated [2, 6]. The CBO was selected for the final set. The extended cyclomatic complexity (ECC) is included in the calculation of the MI which makes CC highly correlated to the MI values [6, 7]. Therefore the CC metric is also excluded from our metrics set. Thus the final metrics set used for calculating QI consists of DIT, CBO, LCOM and MI.

For the product metrics (PM) in the final metrics set, four functions *f* that transform metric values to the product metric quality ratings (PMQR) were defined. The PMQR range is one to five and the range of the PM is metric specific. Figure 3 presents the transformation function for all four metrics (DIT, CBO, LCOM and MI). Please note that these step functions should be calibrated before their use in a different environment.

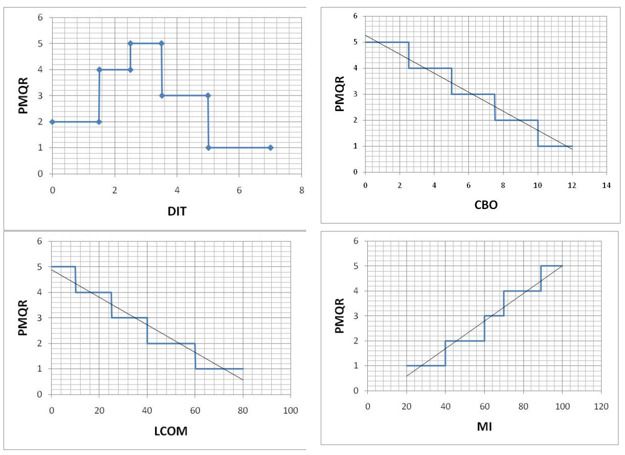

Figure 3: Step function f for the DIT, CBO, LCOM and MI metrics

Calculating acknowledged effort is only one possibility for applying the quality index (QI) defined in this paper. Another possibility is to calculate the corrected productivity of the development team. The corrected productivity is defined as the normalized productivity for the delivered results. In case of bad design and low quality code, the productivity calculated from the effort and size -- sometimes called actual productivity -- is higher then the corrected productivity defined here. If the delivered code is of outstanding quality the actual productivity is lower than the corrected productivity calculated using the quality index.

$$P_A = \frac{E_A}{S} \qquad \text{(Eq. 4)}$$

$$P_C = P_A * (RE_1 + (QI * RE_2)) = \frac{E_{ACK}}{S}$$

where P_A is actual productivity for the current project calculated from the actual effort E_A (sometimes also called recorded effort). P_A is calculated at the end of the project. P_C is corrected productivity, RE_1 and RE_2 are reward factors, S is software size and QI is quality index.

The proposed method was used on a set of OO projects in order to explore the acknowledged effort on real projects. Table 2 summarizes the metrics values for three groups of projects. In the first group are smaller student projects written in Java. The students were from the last grade of the computer science study program. Since Java is already introduced in the first year and used throughout the study program for individual and/or group projects at different subjects it can safely be assumed that in the last year, students have good programming skills and sufficient development experience. In the second group are industry projects developed on the Microsoft .NET platform. The development team was experienced and used sophisticated development approaches like custom code generators and design patterns. The third group is a control group. The projects are for well known products, developed by highly experienced development teams. The metrics data is from the master thesis prepared at the Uppsala University in Sweden [2]. The DIT, NOC, CBO and LCOM metrics were collected on the class level. Therefore two values are provided in the table. The first value is the mean and the second is the standard deviation. MI is calculated at the project level, thus only one value is in the table.

Table 2: The Values for Selected Code Quality Metrics

	DIT	*CBO*	*LCOM*	*MI*
GROUP 1				
Project 1.1	3,35 2,47	11,82 11,23	95,56 13,33	65,19
Project 1.2	1,07 0,38	4,37 5,50	65,19 34,25	49,64
Project 1.3	2,91 2,47	12,55 12,74	68,14 34,54	34,48
Project 1.4	1,82 1,85	5,26 8,49	71,18 28,95	30,31
Project 1.5	2,57 2,18	11,07 12,45	73,17 28,52	34,29
Project 1.6	1,54 1,50	5,7 9,62	72,71 35,41	48,36
Project 1.7	1,00 0	4,20 3,52	51,25 43,18	29,86
Project 1.8	1,21 1,26	2,79 5,36	67,20 35,25	45,37
Project 1.9	4,7 2,13	17,48 12,16	73,91 26,85	72,34
Project 1.10	2,85 2,03	9,00 12,23	66,20 25,08	68,57
Project 1.11	0,36 0,48	1,31 2,57	85,00 10,68	87,49
Project 1.12	0,75 1,68	2,25 5,10	15,63 30,15	49,53
Project 1.13	2,00 2,03	8,58 8,28	74,78 30,47	80,34
Project 1.14	3,64 2,51	14,49 12,42	85,65 7,10	77,04
GROUP 2				
Project 2.1	2,47 1,01	0,79 4,08	4,43 18,04	87,53
Project 2.2	2,23 1,11	5,96 8,81	31,06 37,42	79,98
Project 2.3	3,28 1,71	9,00 8,90	23,57 40,25	89,59
Project 2.4	1,20 0,44	5,00 11,18	12,40 27,72	77,41
GROUP 3				
Project 3.1	0,58 0,75	8,36 5,87	0,33 0,33	129
Project 3.2	0,21 0,41	9,47 6,18	0,41 0,36	164
Project 3.3	1,23 0,83	6,53 4,56	0,38 0,19	182

Table 3 presents data for size, effort and *QI*. Project size is expressed in function points as well as in lines of code (LOC). In column four is the actual effort E_A in hours followed by the acknowledged effort E_{ACK}. In the last column are the values for the quality index calculated following the proposed method. For most of the student projects the *QI* is less than three. Consequently the acknowledged effort is smaller than the actual effort reported by the students. The projects in the second group demonstrate better code quality then is normally expected (*QI*>3,0), the acknowledged effort is higher.

The projects from the third group are not included in the table since the actual effort for them is unknown.

Table 3: Size, Effort and QI Results for Test Projects

	Size (FP)	Size (LOC)	E_A (h)	E_{ACK} (h)	QI
GROUP 1					
Project 1.1	72	4.216	95	90	2,50
Project 1.2	65	4.176	105	97	2,25
Project 1.3	163	1.760	191	172	2,00
Project 1.4	37	2.006	88	81	2,25
Project 1.5	88	2.777	192	173	2,00
Project 1.6	157	3.642	57	54	2,50
Project 1.7	71	1.782	171	158	2,25
Project 1.8	173	2.159	54	50	2,25
Project 1.9	110	3.400	143	132	2,25
Project 1.10	35	1.686	45	44	2,75
Project 1.11	43	2.576	210	210	3,00
Project 1.12	35	623	39	40	3,25
Project 1.13	60	985	125	122	2,75
Project 1.14	70	4.189	165	153	2,25
GROUP 2					
Project 2.1	2.122	93.978	8.800	10.120	4,5
Project 2.2	440	32.532	1.067	1.120	3,5
Project 2.3	1.987	156.122	2.133	2.347	4,0
Project 2.4	13	771	56	59	3,5

5 Conclusion

The typical evaluation of completed software projects includes costs, effort and completeness of the delivered functionality. In this research the focus was only on project effort. From the management point of view, the recorded effort is not necessarily the acceptable project effort when taking into consideration the quality of the delivered code. In this paper, the idea of acknowledged effort was presented. Acknowledged effort combines actual effort with a quality index. The quality index is a single value that represents the quality of the delivered code. The management could then reward or penalize the development team for arbitrary percentages in accordance with the code quality. The formula provided in this paper should be calibrated accordingly. The idea presented in the paper was tested on the sample data set, including 18 projects. The results demonstrate when the effort should be smaller than the actual effort as well as when the quality of code is better than average and the developers should be additionally rewarded for their work.

In the future, the method will be tested with different metrics sets and additional project in order to validate the sensitivity of the proposed method.

Acknowledgement

This work was realised as a part of the Slovenian-Hungarian bilateral research project BI-HU/06-07-01 Application of OO Metrics to estimate .NET project software size.

References

[1] Albrecht,A., (1979). Measuring Application Development Productivity, *IBM Applications Development Symposium*, pp. 83-92.

[2] Andersson, M., Vestergren, P., (2004). *Object-Oriented Design Quality Metrics* (Master Theses), Uppsala University.

[3] Antoniol,G., Lokan,C., Caldiera,G., and Fiutem,R., (1999). A Function Point-Like Measure for Object-Oriented Software, *Empirical Software Engineering*, Springer, pp. 263-287.

[4] Antoniol,G., Fiutem,R., and Lokan,C., (2003). Object-oriented function points: An empirical validation, *Empirical Software Engineering*, Springer, pp. 225-254.

[5] Arisholm, E., (2006). Empirical assesment of the impact of structural properties on the changeability of object-oriented software, Information and Software Technology, Elsevier. pp. 1046-1055.

[6] Chidamber, S.R., Darcy, D. P., Kemerer, C.F. (1998). Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis, *IEEE Transaction on Software Engineering*, IEEE, pp. 629-639.

[7] Fioravanti, F., Nesi, P., Stortoni, F., (1999). Metrics for Controlling Effort During Adaptive Maintenance of Object Oriented Systems, *Proceedings of the IEEE International Conference on Software Maintenance*, IEEE, pp. 483-493.

[8] Gyimóthy, T, Ferenc, R., Siket, I., (2005). Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction, *IEEE Transaction on Software Engineering*, IEEE, pp.897-910

[9] IFPUG, (2004). *Function Point Counting Practices Manual, Release 4.2*, International Function Point Users Group.

[10] ISO, (1986). *ISO 8402 - Quality management and quality assurance*, ISO.

[11] ISBSG, (2001). *Practical Project Estimation, A toolkit for estimating software development effort and duration*. International Software Benchmarking Standards Group.

[12] Marinescu, R., Ratiu, D., (2004). Quantifying the Quality of Object-Oriented Design: the Factor-Strategy Model, *Proceedings on the 11th Working Conference on Reverse Engineering (WCRE'04)*, IEEE.

[13] NASA, (1995). Software Quality Metrics for Object Oriented System Environments, Mational Aeronautics and Space Administration.

[14] Uemura,T., Kusumoto,S., and Inoue,K., 2001. Function-point analysis using design specifications based on the Unified Modelling Language. *Journal of Software Maintenance and Evolution-Research and Practice*, Interscience, pp. 223-243.

[15] Živkovič,A., Heričko,M., and Kralj,T., (2003). Empirical assessment of methods for software size estimation. *Informatica (Ljubljana)*, Slovenian Society Informatika, pp. 425-432.

[16] Živkovič,A., Heričko,M., Brumen B., Beloglavec S., Rozman I., (2005a). The Impact of Details in the Class Diagram on Software Size Estimation, *Informatica (Lithuania)*, Institute of Mathematics and Informatics, pp. 295-312.

[17] Živkovič, A., Rozman, I., Heričko, M., (2005b). Automated Software Size Estimation based on Function Points using UML Models, *Information & Software Technology*, Elsevier, 881 - 890

[18] Živkovič, A., Heričko, M., Porkoláb, Z., (2007). Evaluating the Correlation Between Code Quality, Software Size and Effort, *Proceedings of the 10th International Multiconference Information Society IS 2007 Volume A*, IJS.