How to Estimate Software Size and Effort in Iterative Development[1]

Aleš Živkovič, Marjan Heričko

University of Maribor, Faculty of Electrical Engineering and Computer Science,
Smetanova 17, SI-2000 Maribor, Slovenia
e-mail: ales.zivkovic@uni-mb.si

Abstract
Iterative development has become the predominant development approach. While providing several benefits for developers, iterative development makes size and effort estimates more difficult. The main problem is incomplete artifacts between iterations. In this paper an approach that enables early size estimation using Unified Modeling Language (UML) artifacts is presented. The approach incorporates self-improvement steps that increase estimation accuracy in subsequent iterations. A demonstration of its applicability and research results are also presented. The results anticipate the possibility of a significant improvement in size and effort estimates by applying the approach presented here.

Keywords: iterative development, Function Point Analysis method, object-oriented size estimation, effort estimates

1 Introduction

For accurate project planning, effort estimate is one of the most important values. Effort is calculated using software size as [8,19]:

$$E = n * (Size)^m$$

where E is Effort in man-months, n and m are empirically calculated factors and *Size* is software size expressed in some suitable metric (i.e. Lines Of Code, Function Points, Use Case Points, Object Points, Number of Screens). This paper uses Function Points (FP) to express software size. In general, the estimation methods can be categorized as: (1) expert opinion, (2) using benchmark data, (3) analogy, (4) proxy points, (5) custom models and (6) algorithmic models [8, 19]. Algorithmic size estimation methods like Function Point Analysis (FPA) [1, 10], Mk II FPA [28] and COSMIC-FFP [6] can provide an estimator with a simple and powerful approach to obtain accurate information on a project's size, which can then be used to calculate the necessary development effort according to the statistical data taken from industry-specific repositories. Current algorithmic methods and approaches demand information about the software system at some level of abstraction, which should be equal for all parts of the system being built. However, iterative development produces artifacts at different abstraction levels that are valid within the same iteration. Subsequently, the size estimator can not apply size estimation methods early enough in the development process to make sense for project planning activities. The artifacts in iterative development become a suitable input for algorithmic size estimation methods in the last iteration, when the software size, no matter how accurate it is, has little value for the duration, or cost predictions, of the current project.

[1] This paper is an abridged version of the scientific paper "The Size and Effort Estimates in Iterative Development" published in Information and Software Technology, vol. 50, issue. 7-8, pp. 772-781. For mathematical models and empirical data please see the original paper.

The solution presented in this paper employs early estimates based on use cases and combines them with more accurate class-diagram-based estimates. Therefore, different abstraction levels for the software system are defined. Each abstraction level uses an algorithm that transforms abstraction elements into software size. The accuracy of the transformation depends on the quality of the abstraction itself and the relevance of historical data that fulfills the abstraction gap. The abstraction gap is the gap between the model and the actual system or code.

Most size-estimation methods are function-based and can be applied during the analysis development phase. The early estimation methods are mainly heuristic. The focus of this research is on algorithmic approaches and therefore excludes comparison with heuristic approaches. None of the function-based methods were constructed for size estimation in iterative development since they do not deal with incomplete artifacts. There are only a few object-oriented methods available. Object Points [8] and Object-Oriented Function Points (OOFP) [2] are based on information in the analysis or design class diagram which becomes available late in the analysis phase. Predictive Object Points [8] combine information in the class diagram with selected product metrics in order to predict effort. Beside heuristic methods only two algorithmic methods could be applied very early in the project life cycle - Use Case Points (UCP) [18] and an improved Object-Oriented Function Point method named OOFP2 [30]. The latter is a combination of algorithmic and heuristic approaches since it uses historical and benchmark data in its early estimates.

2 Abstraction levels

Abstraction is an important concept when presenting and reasoning about software systems. It helps us present a problem without the details that are not critical for the purpose of the discussion in question. Using abstraction, the same issue can be presented in different ways, highlighting only those details in the solution space that are important at the present level of dealing with the problem. During object-oriented development, different models of a software system are produced. Models do not represent the actual system running on the machines; they are abstractions of the code. Going through different project development phases, models become less abstract and closer to the code. Let us illustrate this with an example.

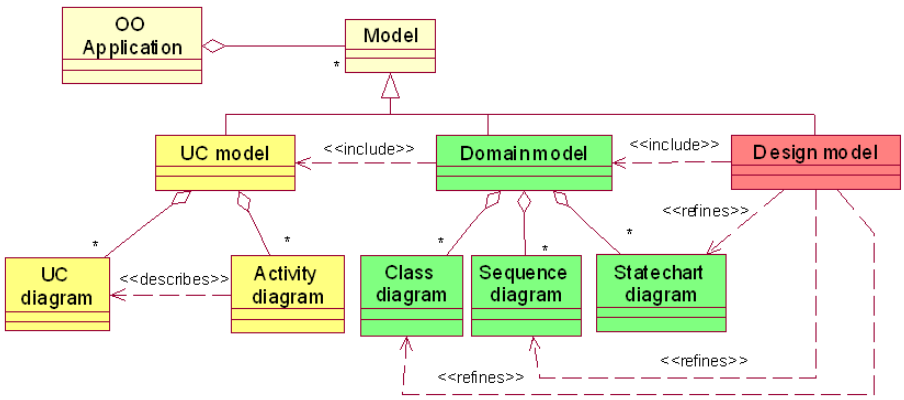

Figure 1: The Software Models and UML diagrams used with Different Abstractions

Figure 1 shows different software models and the corresponding Unified Modeling Language (UML) diagrams [23, 24] used with different abstractions. In this example, the Use Case (UC) model represents the first and, therefore, less-detailed abstraction of the object-oriented application. The domain model is a second abstraction in this example and the design model represents the third -- the most-detailed abstraction that is also closest to the actual code. The

abstractions in Figure 1 are only one example that is commonly used in object-oriented development.
The abstraction level limits the amount of information about the software system available in the individual steps of the development process. The information quantity, as well as quality, further limits the ability to perform software size estimation. Information Quality usually defines the Functional Size Measurement (FSM) method [13, 14], or its steps, and defines the expected accuracy, reliability and costs of the size estimation process. As an example, Table 1 summarizes the levels of function point count as defined in [26]. The six levels defined are: Size Approximation (Level 6), Rough Count (Level 5), Default Complexity Count (Level 4), Detailed Count (Level 3), Detailed Linked Count (Level 2) and Detailed Linked and Flagged Count (Level 1). Although each level has its characteristics and applicability in practice, this rough overview shows two accuracy classes with similar documentation reliability and cost characteristics. The accuracy class defines the average estimation error that characterizes the level. Levels one, two and three are grouped into the first 10% accuracy class. Levels four, five and six are in the second, 20% class.

Characteristic	Level of FP Count					
	Level 1	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
accuracy	very good ~10 %	very good ~10 %	very good ~10 %	good ~15 %	acceptable 20 - 25 %	acceptable around 20 %
reliability	high	high	high	average	average	low
costs	very high	high	average	low	low	very low
input data details	very high	high	high	average	low	very low
documentation level	well documented	documented	documented	documented	partially documented	not documented
repeatability	repeatable	repeatable	repeatable	repeatable	partially repeatable	not repeatable
maintainability	small	small	small	medium	high	very high

Table 1: Total Metrics' Function Point Count Levels

To make the idea more understandable, grouped abstraction levels are used. The first abstraction level uses use case diagrams. For each use case from the use-case diagram, a rough estimate for the number of activities/steps needed to fully accomplish the functional task of the use case is defined with an activity diagram. This estimate is labeled as *Early*. Another abstraction level used in this research uses a final version of the domain class diagram. This estimate is labeled as *Comparative*. The limit of two abstraction levels was set in order to make traceability through iteration possible and to enable the reader to follow the main idea of the work. In practice, the number of abstraction levels can be different. In case of more abstraction levels, the method could be expanded to incorporate one estimate for each abstraction level in each iteration.

2.1 Size estimation method

Size estimation is performed with the OOFP2 method. This section summarizes that approach. For a more detailed description please see [29, 30]. The early estimates are based on use case and activity diagrams [23, 24]. The use case diagrams are available early in the project's life cycle. Use cases are usually briefly described using natural language. The description is needed to be able to divide use cases into iterations and to build an iteration plan. When a more detailed use case description becomes available, re-assessment is automatically made using a tool.

In the OOFP2 approach, size estimation is done in three stages. What estimate to apply depends on what information is available. In iterative estimates presented in this paper only two estimation stages are used. The two estimation stages are:

- **Early estimation** (UCD&AD) - with additional information available, the statistically founded basic estimate can be replaced with an estimate based on actual project data. The activity diagrams are used in this estimation; results from sequence diagrams are also considered.
- **Final estimation** (CD) - this estimation is based on the domain class diagram using OO-to-FPA mapping, as described in [30]. This estimate is the final one in the prediction cycle. The estimates that follow are performed for purposes of comparison and repository fulfillment only.

3 Iterative development

An iterative process makes it possible to easily accommodate changes, to obtain feedback and factor it into the project, to reduce risk early, and to adjust the process dynamically [9]. Iterative development dramatically changed the software development process that was previously in use in the 1990s. The processes in use at that time focused on each developmental step until all the artifacts were produced, when the transition to the next step was made. The incorrectness from previous steps was amplified in subsequent steps without the possibility of verifying the correctness of the results. In iterative development, on the other hand, the development process has been divided into several iterations where all the steps are performed in each iteration, producing only partial results for each step. For example, if the project has three iterations, the detailed description of all use cases is produced at the beginning of the third iteration. However, the results of the previous two iterations will also include the full implementation of functionality assigned to these iterations. The advantages are obvious. The likelihood of a successful project completion is higher beginning with each new iteration. The development cycles are shorter and therefore give more opportunities for user feedback. Consequently, the probability of delivering the right functionality with satisfactory usability is also greater in comparison with traditional development.

Despite all of the positive aspects, iterative development has several side effects. One of them is with regard to software size estimation. To estimate software size accurately in object-oriented development, the final version of the domain class model has to be available [2, 7, 29, 30]. As already described in this section, the final versions of object-oriented artifacts are developed in the last iteration. However, the outline of the complete project plan must be conducted at the beginning of the project in the first iteration. This initial project plan is then updated at the end of each iteration to reflect deviations between the planned activities and performed activities. Now, if we do not have appropriate inputs for the existing size estimation methods, the methods can not be applied. The second best solution is the use of methods for early estimates. These methods usually rely on statistical data, repositories, and industry average values [12, 18]. The size and effort problem is reduced since some initial estimates are available for project planning. We can take this improvement a step forward by joining early estimate methods with the methods for final estimates in object-oriented development. The basic idea comes from the eXtreme Programming (XP) [4] community where members of the team help their coach estimate the size of user stories. The team then declares how much effort capacity they think they will average per iteration. That figure is called *velocity* [4]. The customer creates a release plan grouped into iterations according to the project effort estimate and the velocity. In XP, the iterations have a fixed duration. The

number of finished story points in the last iteration is used as an estimate for how many points will be done in the next iteration. This principle is called "Yesterday's Weather" and its guiding principle is: the best predictor for today's weather is yesterday's weather. Measuring the time needed for implementing user stories (or use cases), the coach gets an idea of a team's performance and is able to correct the velocity estimates for subsequent iterations. The velocity may bounce around the actual value, but it usually stabilizes quickly. This is due to the use of relative estimates instead of absolute ones. In this research, the process of the velocity estimate is applied to size estimates. If the process is repeated long enough, the prediction can come as close to the actual value as the estimator wants (Cauchy condition of limit existence). Figure 2 illustrates this process for size estimates.

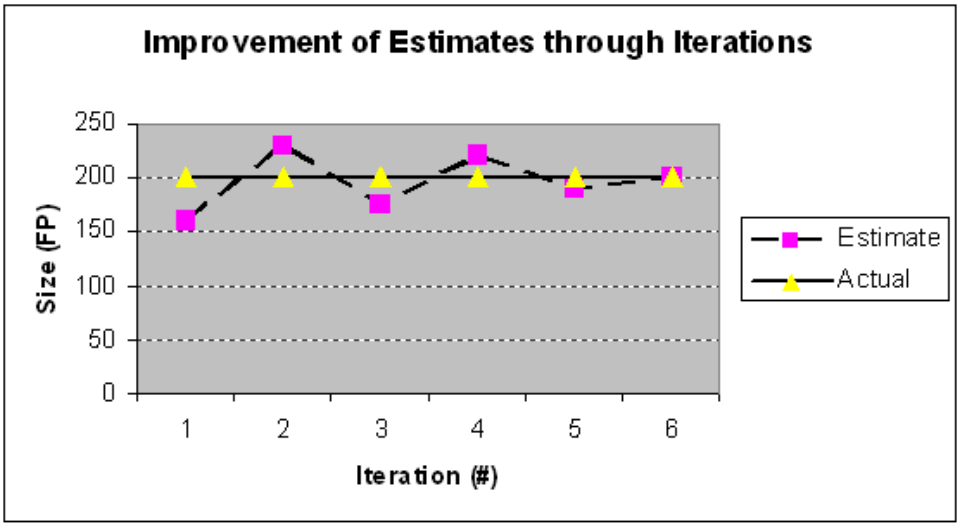

Figure 2: Improvement of Estimates through Iterations

Assume that a project size is 200 function points (FP). At the beginning of the project, the estimate is 160 FPs. After the use cases assigned for the first iteration are developed, the underestimation of the first estimate becomes obvious. The second estimate in our example is 230 FPs. Again, after the use cases assigned for the second iteration are developed, the difference between the actual iteration size and the iteration estimate is calculated. The difference between the actual size and the estimate should decrease. Please keep in mind that during the project, the actual size has not yet been acknowledged. Repeating the process, the estimate error is smaller from iteration to iteration and hopefully the exact value is then predicted. In the next section, this idea is applied to object-oriented size estimation. Please bear in mind that only the main idea of velocity is used in our iterative size estimation process. There is an important difference between the XP estimation process using velocity and our process. In the XP estimation process, the velocity value representing the development capability of the team is changed from iteration to iteration. The estimates of the user stories are not changed [4]. In our size estimation process, the difference between the estimated and actual value for the iteration is used to calculate the correction factor. The correction factor is then applied to the original size estimates.

4 Iterative size estimation

In this section, the process of size estimation in iterative development is described using a simplified example. The process is built on two size estimation techniques. The early estimate employs use cases and activity diagrams to foretell the size of the completed project. The project size is expressed in function points (FP). Then, a new estimate is made after a domain

class model is conducted in the first iteration. The difference between both estimates is used to improve size estimates for the use cases assigned to future iterations. Figure 3 shows an example of a use case diagram with four use cases (UC_1-UC_4). The first two use cases (UC_1 and UC_2) are assigned to the first iteration, UC_3 to the second and UC_4 to the last iteration.

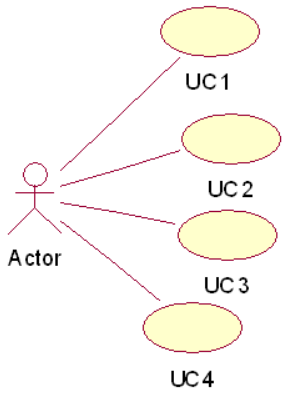

Figure 3: Sample Use Case Diagram

	Project Size in Function Points (FP)					
	Iteration 1		Iteration 2		Iteration 3	
	Early Estimate	Final Size	Early Estimate	Final Size	Early Estimate	Final Size
UC_1	x_{E1}	$x_1=x_{E1}+d_1$				
UC_2	x_{E2}	$x_2=x_{E2}+d_2$				
UC_3	x_{E3}	N/A	$x_{EC3}=x_{E3}*(1+r_1)$	$x_3=x_{EC3}+d_3$		
UC_4	x_{E4}	N/A	$x_{EC4}=x_{E4}*(1+r_1)$	N/A	$x_{EC4}=(x_{E4}*r_1)*r_2$	$x_4=x_{EC4}+d_4$
Iteration Total Size (FP)	$sum(x_{E1}:x_{E2})$	$sum(x_1:x_2)$	x_{EC3}	x_3	x_{EC4}	x_4
Project Total Size (FP)	$sum(x_{E1}:x_{E4})$	$sum(x_1:x_2)+sum(x_{E3}:x_{4E4})$	$sum(x_1:x_2)+sum(x_{EC3}:x_{EC4})$	$sum(x_1:x_3)+x_{EC4}$	$sum(x_1:x_3)+x_{EC4}$	$sum(x_1:x_4)$

Table 2: The Process of Size Estimates in Iterative Development

	Project Size in Function Points (FP)					
	Iteration 1		Iteration 2		Iteration 3	
	Early Estimate	Final Size	Early Estimate	Final Size	Early Estimate	Final Size
UC_1	x_{E1}	$x_1=x_{E1}+d_1$				
UC_2	x_{E2}	$x_2=x_{E2}+d_2$				
UC_3	x_{E3}	N/A	$x_{EC3}=x_{E3}*(1+r_1)$	$x_3=x_{EC3}+d_3$		
UC_4	x_{E4}	N/A	$x_{EC4}=x_{E4}*(1+r_1)$	N/A	$x_{EC4}=(x_{E4}*r_1)*r_2$	$x_4=x_{EC4}+d_4$
Iteration Total Size (FP)	$sum(x_{E1}:x_{E2})$	$sum(x_1:x_2)$	x_{EC3}	x_3	x_{EC4}	x_4
Project Total Size (FP)	$sum(x_{E1}:x_{E4})$	$sum(x_1:x_2)+sum(x_{E3}:x_{4E4})$	$sum(x_1:x_2)+sum(x_{EC3}:x_{EC4})$	$sum(x_1:x_3)+x_{EC4}$	$sum(x_1:x_3)+x_{EC4}$	$sum(x_1:x_4)$

Table 2 illustrates the iterative estimation procedure. The table is divided into three iterations (columns: Iteration 1 - Iteration 3). For each iteration, two values for size are provided. The

Early Estimate is a size estimate that calculates the size from use cases and activity diagrams; the *Final Size* is the actual size. For the first iteration, labeled Iteration 1, the early estimate produced size estimates x_{E1}, x_{E2}, x_{E3} and x_{E4} for use cases UC_1, UC_2, UC_3 and UC_4. Since only UC_1 and UC_2 are assigned to the first iteration, the iteration total is the sum of size estimates for all use cases in this iteration (in our case UC_1 and UC_2). The project total is the sum of all four values. In the second column of the first iteration, the values for finished use cases are updated and the use cases scheduled for the subsequent iterations are marked with N/A. The early estimates x_{E1} and x_{E2} are updated with the values d_1 and d_2 that represent the difference between the early estimates and the actual size, in our example: x_1 and x_2. Following this change, the iteration and project totals are also updated. Before continuing to the next iteration we have to determine the value (r) that should be used to correct early estimates from the first iteration for the use cases that were not yet implemented (UC_3 and UC_4). The equation for this calculation is:

$$r_1 = \frac{d_1 + d_2}{x_1 + x_2}$$

After the first iteration is completed, the estimated values for use cases assigned to the first iteration are compared to the actual values. In the general form, the final total estimated project size for iteration K is defined as the estimated total project size for iteration K plus the difference with the actual size.

Finally, the corrected estimated size for iteration K+1 can be calculated and used in calculating the estimated total project size in iteration K+1. The corrected estimate for iteration K+1 is calculated as the estimate in iteration K multiplied by the correction factor. The total project size estimate is composed of two parts: the first part is the sum of actual values from previous iterations and the second part is the sum of corrected values for future iterations.
Now, we return to the example in Table 3. The estimate of the project total is the sum over the early estimates (x_1, x_4), the actual difference for the already implemented use cases annotated with *d* and estimates corrected with correction factors *r* that were calculated using lessons learned in the previous iteration. The same procedure is now applied once again for the second iteration. The outcome of the second iteration is a new correction factor (r_2) that is then used for the last iteration.

5 Conclusion

Size and effort estimations are important metrics that, if accurate, have a positive impact on project planning and management. Although several methods, procedures, and principles for size and effort estimation exist, their applicability is limited when iterative development is used. In this paper the principles of iterative size and effort estimates were presented. The process is based on the use of velocity from extreme programming. It uses the difference between the estimated and actual values of a previous iteration to forecast the size of the next iteration as the sum of the size estimate and calculated correction value. The process is adapted to iterative development. The iterative estimation process was tested in a controlled environment. The results are very promising since the corrected values significantly outperformed original values in two out of three test projects.

6 References

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

[2.] G. Antoniol, C. Lokan, G. Caldiera and R. Fiutem, A Function Point-Like Measure for Object-Oriented Software, Empirical Software Engineering, 4 (1999), pp. 263-287.

[3.] R. Asshman, Project Estimation: A Simple Use-Case Based Model, IT Pro, July/Avgust 2004, pp. 40-44.

[4.] K. Beck, M. Fowler, Planning Extreme Programming, Adison-Wesley, 2001.

[5.] J. Bielak, Improving Size Estimates Using Historical Data, IEEE Software, November/December, 2000, pp. 27-35.

[6.] COSMIC. COSMIC-FFP Measurement Manual - The COSMIC Implementation Guide for ISO/IEC 19761:2003, version 2.2., Common Software Measurement International Consortium (COSMIC), 2003.

[7.] H. Diab, M. Frappier, and R. St Denis, A formal definition of function points for automated measurement of B specifications. Formal Methods and Software Engineering, Proceedings, pp. 483-494, 2002.

[8.] D. D. Galorath, M. W. Evans, Software Sizing, Estimation, and Risk management, Auerbach Publications, 2006.

[9.] IBM, Rational Method Composer - Rational Unified Process, version 7.1, 2006

[10.] IFPUG, Function Point Counting Practices Manual, Release 4.2, International Function Point Users Group (IFPUG), Princeton Junction, USA, January 2004.

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

[12.] ISBSG, ISBSG Estimating, Benchmarking and Research Suite R10 CD ROM. International Software Benchmarking Standards Group, http://www.isbsg.org, 2006.

[13.] ISO, ISO/IEC TR 14143-1. Information technology - Software measurement - Functional size measurement, Part 1: Definition of concepts, First edition, ISO/IEC, 1998.

[14.] ISO, ISO/IEC TR 14143-2. Information technology - Software measurement - Functional size measurement, Part 2: Conformity evaluation of software size measurement methods to ISO/IEC 14143-1:1998, First edition, ISO/IEC, 2002.

[15.] Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, Second printing, April 1999.

[16.] M. Jørgensen, U. Indahl, D. Sjøberg, Software effort estimation by analogy and "regression toward the mean", The Journal of Systems and Software, 68 (2003), p.p. 253-262.

[17.] J. Kaczmarek, M. Kucharski, Size and Effort estimation for applications written in Java, Information and Software Technology, 46 (2004), pp. 589-601.

[18.] G. Karner, Use Case Points - Resource Estimation for Objectory Projects, Master Thesis, Linkøping University, Sweden, 1993

[19.] L. M. Laird, M. C. Brennan, Software Measurement and Estimation: A Practical Approach, John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

[20.] K. D. Maxwell, P. Forselius, Benchmarking Software Development Productivity, IEEE Software, January/February 2000, p.p. 80-88.

[21.] P. Mohagheghi, B. Anda, R. Conradi, Effort Estimation of Use Cases for Incremental Large-Scale Software Development, ICSE'05, May 15-21, 2005, St. Louis, Missouri, USA, pp. 303-311.

[22.] S. Moser, B. Henderson-Sellers, V. B. Mišoć, Measuring Object-Oriented Business Models, Proceedings of the Technology of Object-Oriented Languages and Systems - Tools-25, 1997, pp. 340
[23.] OMG, Unified Modeling Language Specification, version 1.4., Object Management Group, 2001.
[24.] OMG, Unified Modeling Language: Superstructure Specification, version 2.0, Object Management Group (OMG), 2006.
[25.] V. Rajlich, P. Gosavi, Incremental Change in Object-Oriented Programming, IEEE Software July/Avgust 2004, pp. 62-69
[26.] Total Metrics, Total Metrics - Levels of Counting. http://www.totalmetrics.com, pp. 1-8. 2001. Total Metrics, 2001.
[27.] Trendowicz, J. Heidrich, J. Münch, Y. Ishigai, K. Yokoyama, N. Kikuchi, Development of a Hybrid Cost Estimation Model in an Iterative Manner, ICSE'06, May 20-28, 2006, Shanghai, China, pp. 331 - 340
[28.] UKSMA, UKSMA. Mk II Function Point Analysis, Counting Practices Manual. version 1.31. United Kingdom Software Metrics Association (UKSMA), 1998.
[29.] Zivkovic, M. Hericko, B. Brumen, S. Beloglavec, I. Rozman, The Impact of Details in the Class Diagram on Software Size Estimation, Informatica (Lithuania), Volume 16, Number 2, 2005
[30.] Zivkovic, I. Rozman, M. Heričko, Automated Software Size Estimation based on Function Points using UML Models, Information & Software Technology, Volume 47, Issue 13, 1 October 2005, Pages 881-890

Aleš Živkovič is an Assistant Professor at the University of Maribor. His research work covers different aspect of object technology with the emphasis on UML, software processes, Java platform and metrics. He has more then ten years of experience with OT and has prepared and carried out numerous seminars and workshops on Java SE and EE, UML, RUP, and XML. He gained his practical experiences in cooperation with industry on several projects. Aleš received his master degree in 2000 and PhD degree in 2005 both from University of Maribor. He is Certified Information System Auditor (CISA) and hold professional certificate for Object Oriented Analysis and Design with UML and PRINCE 2 Practitioner certificate.

Marjan Heričko is an Associate Professor at the University of Maribor, Faculty of EE&CS, Institute of Informatics. He received his M.Sc. (1993) and PhD (1998) in computer science from the University of Maribor. His research interests include all aspects of IS development with emphasis on metrics, software patterns, process models and modelling. Marjan is the deputy head of the Institute of informatics and scientific coordinator of the Slovenian national platform for software and services.