INFORMATICA, 2005, Vol. 16, No. 2, 295–312
© 2005 *Institute of Mathematics and Informatics, Vilnius*

The Impact of Details in the Class Diagram on Software Size Estimation

Aleš ŽIVKOVIČ, Marjan HERIČKO, Boštjan BRUMEN,
Simon BELOGLAVEC, Ivan ROZMAN
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova ulica 17, SI-2000 Maribor, Slovenia
e-mail: ales.zivkovic@uni-mb.si

Received: March 2004

Abstract. Software size is an important attribute in software project planning. Several methods for software size estimation are available; most of them are based on function points. Albrecht introduced function points as a technologically independent method with its own software abstraction layer. However, it is difficult to apply original abstraction elements to current technologies. Therefore researchers introduced additional rules and mappings for object-based solutions. In this paper several mapping strategies are discussed and compared. Based on the similarities in compared mappings, a common mapping strategy is then defined. This mapping is then tested on the reference application portfolio containing five applications. The aim of the test scenario is to evaluate the impact of the diverse detail levels in the class diagrams on software size measurement. Although the question of how to perform quality size measurements in object-oriented projects remains unanswered, the paper gives valuable information on the topic, supported by mathematics.

Key words: requirements engineering, function points, object modelling.

1. Introduction

The success of software projects depends on the decisions that project managers make in the early phases. Their decisions should be based on provable facts, not on intuition. One of the fundamental values for project planning is software size. Using software size, the effort and duration of the project can be calculated so that it has an influence on a project's plan and budget. Several methods for software size estimation are in use today. Most of them have their roots in the Function Point Analysis Method (FPA) introduced by Albrecht in 1979 (Albrecht, 1979). In this paper, the focus is placed on FPA method use with object-oriented (OO) projects. Although the FPA method is declared as technologically independent its usage is more difficult with OO projects. The method's independence is based on its own concepts describing a software system. Abstraction of the software system is gained with a standard separation in two parts: one part considers the data influence and another part takes into account the functionality of the system. Data functions are further divided into internal and external logical files assigning different weights to each data function type. The transactional functions describe functionality through three abstract types namely: external inputs, external outputs and external inquiries. 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. With OO design mapping is not that obvious. Therefore, several researchers proposed additional rules on how to use the FPA method with OO concepts. In the second section those rules are compared. Based on their similarities, a unified view on the rules is presented in mathematical form. The performance of mapping and its influence on size estimation accuracy is tested on the reference application portfolio containing five similar applications. In section three, the OOFP method results are compared with the values of four traditional methods for software size estimation. In addition, the impact of incomplete class diagrams on measurement is tested. To improve the results for OO systems, a slight change in the FPA complexity tables is proposed. Towards the end the difference in measurements on subsequent OO development artefacts is analysed in detail. In the conclusion, the results are summarised and discussed. The goals of the research are summarised as:

1. Comparison of OO-to-FPA mappings: to compare various authors' approaches and find common principles and mapping steps.
2. Formal model definition: in order to automate counting and provide transparency for the procedure, the mathematical model must be defined.
3. Estimates on early OO artefacts: to use the OOFP method as soon as possible and test its performance.
4. To measure the impact of incomplete data.

1.1. *Mappings of OO Concepts to FPA Concepts*

Basically the FPA method is technologically independent, and therefore equally applicable to projects where structured techniques are used as well as those characterised by the use of OO methods.

In practice, however, it is difficult to apply original rules to OO concepts and artefacts. The rules in the FPA manual (IFPUG, 1999) deal only with structured concepts. Consequently, some researchers have introduced additional rules and mappings of OO concepts to FPA elements. None of the proposed mappings have become a standard yet. Table 1 summarises different mapping approaches according to their principal author. Table 1 shows that various authors use similar rules to map OO concepts to FPA concepts. We will describe the main characteristics of each individual method. Deviations that exist between different mappings are discussed in Section 2.

1.2. *Method Proposed by Fetcke et al.*

Fetcke (Fetcke *et al.*, 1997) focused their research on a specific method, namely Object-Oriented Software Engineering (OOSE) (Jacobson and Christerson, 1992). The method is based on use cases. The authors proposed four groups of rules:

- identification of the counting boundary;

Table 1
Different approaches for mapping OO concepts to FPA elements

FPA element	Author			
	Fetke *et al.*	Uemura *et al.*[*]	Ram *et al.*[*]	Antoniol *et al.*[*]
Counting Boundary	Actor – UC boundary	Actor – UC boundary	Not identified	Not identified
Transactional Function (TF)	Mapping not uniform (it is based on UCs)	Identified according to patterns of communication in sequence diagrams	Method of the class	Method of the class
Data Function (DF)	Domain class, error message, help	Object with at least one attribute that exchanges data with another object	Data visible to every method of the class	Class or a set of classes (aggregation, inheritance)
Data Element Type (DET)	Attribute of the class	Attribute of the class, attribute of the super class	Attribute with a basic type	Attribute with a basic type, association with cardinality one
Record Element Type (RET)	Aggregation, abstract class, subclass (inheritance)	Always one	Reference to another object	Reference to another object, attribute (complex type), association (1:M or M:N), aggregation
File Type Referenced (FTR)	Object maintained or read by UC	Object in the message sequence	Does not distinguish between RET and FTR	Does not distinguish between RET and FTR

[*] Only applicable to analysis and design artefacts

- identification of items within the boundary (transactional and data functions);
- identification of the *item* type (DET, RET, FTR) for all items;
- prescribing the weight factors.

The application boundary is set in accordance with the definition of the boundary in the Use Case (UC) diagram, as defined in the UML standard. Actors are mapped into users of the system. Use Cases are mapped into transactional functions. This mapping is not always one-to-one. The number of transactional functions for the particular UC is usually defined by the UC description. The authors do not provide additional guidelines on the matter. The reference to the rules of the original method is noted. Since the OOSE method distinguishes between three types of objects (control, entity and boundary), only entity objects and objects with unknown types performing a count at the time are used as data functions. Aggregation and generalisation are treated in a specific way. Both concepts can have a significant impact on the number of data elements types, record element types and file types referenced, respectively.

1.3. *Method Proposed by Uemura et al.*

Uemura (Uemura *et al.*, 1999) use class and sequence diagrams as sources for OO-FPA mapping. The mapping is specified for diagrams developed in design that conduct design specification. The system boundary is identified according to the messages in the sequence diagrams. The messages sent by actors to non-actor objects represent the system boundary. Basically, the rule is the same as defined in the UML standard (OMG, 2001) the only difference is the diagram from which the boundary is identified. We can simplify this rule to become consistent with the rule used by Fetcke (Fetcke *et al.*, 1997). Under the presumption that each sequence diagram is directly related to one UC in the UC diagram, the boundary can be identified from the UC diagram. Objects with at least one attribute that have one, or more methods or call methods of other objects are mapped into data functions. The classes with methods that influence the state of other objects are mapped to external interface files (EIF). All others are considered as internal logical files (ILF). Transactional functions are identified according to five different communication patterns from the sequence diagram. The transactional function type is also identified from the communication pattern.

1.4. *Method Proposed by Antoniol et al.*

Antoniol (Antoniol *et al.*, 1999) proposed two methods for estimating size during the development process. In the early project phase, the use of the original method is proposed. After the design is conducted, the method developed by the authors, called Object-Oriented Function Points (OOFP), is considered to be more appropriate. The OOFP method is based on the deliverables of the design phase and takes advantage of the information available in the class diagrams. With this method, the gap between system abstraction, used by the FPA method and the abstraction, made with the class diagrams, is supposed to be resolved. The method provides additional rules with some freedom and the ability to choose the mapping algorithm in cases where class diagrams exhibit complex class hierarchies. With regard to the aggregation and generalisation/specialisation associations, the possible mapping choices are:

1. Single Class – each separate class is mapped to the internal logical file.
2. Aggregation – the entire aggregation structure is mapped to a single internal logical file.
3. Generalisation/Specialisation – classes comprised of the entire path from the root super-class to each leaf sub-class are mapped into the single internal logical file.
4. Mixed – a combination of the second and third option.

The number of attributes and associations for other classes are used to define internal logical file complexity. To distinguish between data element types and record element types, a simple but indistinct rule is used. The complex data types are classified as a record element types and simple or primitive data types as data element types.

Transactional functions are identified according to the methods in the class. The authors call them service requests. Abstract methods and inherited methods are ignored. The method complexity is determined according to the number of parameters and global variables referenced in the method. Similar as with attributes, the element data type is used for the classification of data element types, or the file types referenced.

1.5. *Method Proposed by Ram et al.*

The mapping is basically the same as in the OOFP approach. The function points for each class are calculated. The number of function points for the class is a sum of its logical file and transactional functions contribution. The transactional functions contribution is calculated from the methods; whereas for methods without parameters and with the void return type, complexity is considered as if it were for one DET. Ram *et al.* (Ram and Raju, 2000) define additional rules for class complexity classification. These rules are used in the second step. The class complexity is defined as being low if the class processes less then 50% of data visible to the class, average if 51 to 70% is processed, and high if the amount of processed data exceeds 70%. A numerical value is then assigned to the given complexity and multiplied with the number of function points calculated in the first step. The final amount is lowered for 10, 40 or 70%, according to its complexity. The term "*data elements which are visible to all methods of a class*" used by Ram *et al.* (Ram and Raju, 2000) is uncommon since such data elements are called attributes and are by definition visible in all methods. The term "process the data" also has a weak definition. Do the get and set methods process the data? If the answer is yes, then the second step concept fails; otherwise it is difficult to automate steps since a deep insight into the method's behaviour must be considered.

2. Comparison of Mappings

Comparing all four mappings we can conclude that Fetcke *et al.* (1997) and Uemura *et al.* (1999) define "boundary" in the same way. Fetcke *et al.* (1997) uses UC diagrams as the reference for boundary identification, Uemura *et al.* (1999) uses sequence diagrams. Conceptually, the mapping is the same. However, from our point of view Uemura's approach is less applicable for two reasons:

- Sequence diagrams are usually made in combination with the UC. It is uncommon to draw only sequence diagrams. However, it is not necessary for the sequence diagram to have an actor (e.g., associations "include" and "extend"). On the other hand, one UC can have many sequence diagrams that aggravate boundary identification. From the UC diagram, the boundary is directly visible and clearly defined by the UML standard.
- In the design time it is usually clear what is in the system and what is outside it. Classes represent the abstraction of the system to be built and are inside the fictive boundary of the system. In fact setting the boundary at the design time is unnecessary, thus it does not influence further steps. Antoniol (Antoniol *et al.*, 1999) and Ram (Ram and Raju, 2000) choose the same approach.

All approaches have a united view on data function mapping. The class is mapped to a logical file. However, the definitions for separation on internal or external logical files are weak in all four methods. Uemura (Uemura *et al.*, 1999) uses supplemental rules to distinguish between two data element types using class operations. Antoniol (Antoniol *et al.*, 1999) retain original guidelines regarding logical files division. In the OO systems, external classes encapsulate non-system components, such as other applications, external services and reused library classes. External classes correspond to external logical files, Antoniol (Antoniol *et al.*, 1999) says. However, a precise definition on how to count external classes is missing. Ram (Ram and Raju, 2000) uses Antoniol's definition. A mismatch in the element type can have a significant influence on the final result. With the use of statistical data it is possible to assess the magnitude of that impact and the corresponding error. Let us assume that the types of some elements were mixed. Eqs. E1 and E2 calculate the element mismatch error. In the Eq. E1, the error for data functions is calculated.

$$\begin{aligned}
&E_{DF} = \frac{\sum_i W_{ILF}(F_i)}{\sum_j W_{ILF}(F_{ILF_j}) + \sum_k W_{EIF}(F_{EIF_k})},\\
&N_{DF} = N_{ILF} + N_{EIF},\\
&E_{EIF} = \frac{N_{EIF}}{N_{DF}} * \frac{W_{EIF}(F_{ILF})}{W_{ILF}(F_{EIF})} \cong \frac{N_{EIF}}{N_{DF}} * \overline{e}_{EIF} = 0.185 * 0.3 = 0.0555,\\
&E_{ILF} = \frac{N_{ILF}}{N_{DF}} * \frac{W_{ILF}(F_{ILF})}{W_{EIF}(F_{ILF})} \cong \frac{N_{ILF}}{N_{DF}} * \overline{e}_{ILF} = 0.815 * 0.44 = 0.3586,
\end{aligned} \tag{E1}$$

where

N_X is the number of classes of type X,
W_X is the weight of type X,
F_X is the data element of type X,
E_X is the error made if all elements of type X are mismatched with another type,
$X = \{ILF, EIF\}$.

To get concrete numbers we need the ratio between data element types. The left graph in Fig. 1 shows the ratio between FPA elements from the ISBSG repository (ISBSG, 2001) (with 238 projects in the sample) and on the right is the graph for our sample of eight applications (e.g., see Zivkovic *et al.*, 2003). According to the industrial average, there are less then 20% of external interface files in the data functions for standalone

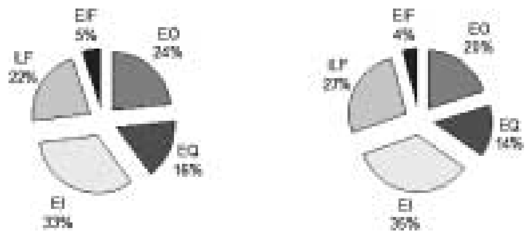


Fig. 1. The ratio between FPA elements.

applications, which have been developed from scratch. If we count all elements as internal logical files (ILF) the error rate is around 5%. In the opposite case, when we neglect ILFs, the error rate is 35%. The calculations indicate the use of internal logical files for all classes. The error rate is minimized. In the next section the suitability of original complexity tables is discussed in further detail.

In mappings for transactional functions, the gap between approaches is greater. Fetcke (Fetcke *et al.*, 1997) defines transactional functions according to the UC and their description. Uemura (Uemura *et al.*, 1999) defines five interaction patterns. Transactional functions are recognised from the sequence diagrams according to the type of the object starting and completing the interaction sequence. Patterns are used to identify transactional function complexity. Antoniol (Antoniol *et al.*, 1999) uses a term service request instead of a method, although the methods are actually counted and mapped into transactional functions. Abstract and inherited methods are not counted. In Antoniol's (Antoniol *et al.*, 1999) opinion it is impossible to determine the type of the transactional function from the class diagrams. The complexity table for external inputs and queries is used. Ram (Ram and Raju, 2000) also uses methods to determine the number and complexity of transactional functions. Comparing his approach with Antoniol's, Ram (Ram and Raju, 2000) is more careful with the inherited methods. If the inherited method overrides a method, its complexity is considered for that derived class alone. Ram (Ram and Raju, 2000) also points out that the abstract methods are defined in the derived classes and should be considered when calculating the complexity of each derived class. Antoniol (Antoniol *et al.*, 1999) indirectly uses the same rule, although he does not formulate it. Ram (Ram and Raju, 2000) calculates the complexity of transactional functions in two steps. For the first step, he uses Antoniol's (Antoniol *et al.*, 1999) approach of using method's signature to determine its complexity contribution. In the second step, (Ram and Raju, 2000) uses its unique class complexity classification (see Section 2). The Eq. E2 calculates the maximal error for transactional functions in the same manner E1 does for data functions. Since external inputs and external inquiries have the same weight, they are treated equally. If external inputs and external inquiries are both neglected and all transactional functions are treated as external outputs, error rate could be as high as 16%. If external outputs are mixed with external inputs or external inquiries, the rate of error is around 7%. The final numbers are specific for the case presented in the paper and must be recalculated for other types of applications.

$$\begin{aligned}
&E_{TF} = \frac{\sum_i W_{EI,EQ}(T_i)}{\sum_j W_{EI}(T_{EI_j}) + \sum_k W_{EO}(T_{EO_k}) + \sum_l W_{EQ}(T_{EQ_l})},\\
&N_{TF} = N_{EI} + N_{EO} + N_{EQ},\\
&E_{EI,EQ} = \frac{N_{EI,EQ}}{N_{TF}} * \frac{W_{EI,EQ}(T_{EI}/T_{EQ})}{W_{EO}(T_{EI}/T_{EQ})} \cong \frac{N_{EI,EQ}}{N_{TF}} * \overline{e}_{EI,EQ}\\
&\qquad = 0.67 * 0.25 = 0.1676,\\
&E_{EO} = \frac{N_{EO}}{N_{TF}} * \frac{W_{EO}(T_{EO})}{W_{EI,EQ}(T_{EO})} \cong \frac{N_{EO}}{N_{TF}} * \overline{e}_{EO} = 0.33 * 0.2 = 0.066,
\end{aligned} \tag{E2}$$

where

N_X is the number of classes of type X,

W_X is the weight of type X,

T_X is the transactional function of type X,

E_X is the error made if all transactional functions of type X are mismatched with other types,

$X = \{EI, EO, EQ\}$,

j, k and l can have values from 1 to the number of transactional functions of the belonging type.

There are slight deviations in the process of data elements identification. In all approaches, the attributes are the core candidates for data elements. If the attribute type is simple, the attributes map to a data element type; otherwise into a record element type for data functions and file type referenced for transactional functions. Antoniol (Antoniol *et al.*, 1999), Ram (Ram and Raju, 2000) and Fetcke (Fetcke *et al.*, 1997) pay regard to different kinds of associations that can map to either data element types or record element types/file types referenced. The rules are different (see Table 1). Only Uemura (Uemura *et al.*, 1999) set the number of record element types to one for all cases.

In this section, four different mapping approaches have been described and compared. Since the approaches are similar, in further research we decided to use Antoniol's (Antoniol *et al.*, 1999) approach. Ram (Ram and Raju, 2000) uses Antoniol's approach while making some changes, other approaches are less precise. The selected approach is simplified for the purposes of this research, using single class strategy for identifying logical files. Ram's idea to change the complexity tables is also reconsidered.

3. Test Case and Results

Let us assume that the requirements for a system are given. The OO analysis and design using UML are used to model the system. The software development process defines procedures to produce prescribed artefacts. For software size estimation only class diagrams are used. During analysis and design the class diagrams change dramatically. Table 2 illustrates that change. The table is composed from a well-known software process framework.

During the requirement engineering in the inception phase, class diagrams are not available. Use cases are identified and briefly described using natural language. Use Cases are prioritised and assigned into iterations. In the elaboration phase, initial class diagrams are conducted during the analysis and then refined in subsequent steps. The result is a domain class diagram containing operations, attributes, initial hierarchy and associations. In the design, an architectural style is selected, the domain class model is changed to reflect architectural properties, a database solution is designed and a presentation tier is developed. During the implementation, a design class diagrams are implemented and usually supplemented with additional methods and attributes.

Software size estimation from OO artefacts (UC diagrams) is hard to perform using the approaches described in Section 2. The first point where size estimation is possible

Table 2

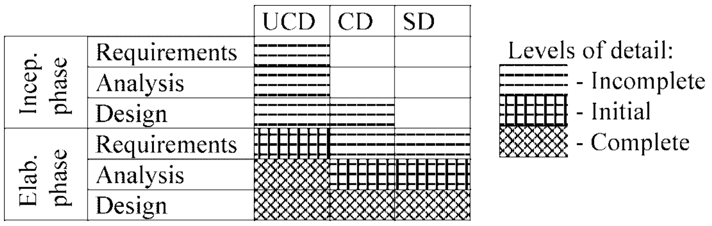
Selected UML diagrams through the development process

	Development process	
	Inception phase	Elaboration phase
Requirements	◊ Initial requirements are captured ◊ UCs are identified ◊ Priorities for UCs are set	◊ Initial UC description is formed
Analysis	◊ Basic domain concepts are identified	◊ Initial domain class diagram is conducted ◊ Interaction sequences are outlined
Design	◊ GUI type is set ◊ Architectural style is defined	◊ Complete class diagrams are conducted ◊ Detailed sequence diagrams are available
Implementation	◊ N/A	◊ Class diagrams are improved

is in the elaboration phase, when the initial domain class diagrams are available. Table 3 shows different levels of detail for selected UML diagrams through the inception and elaboration phase. Three levels of detail are defined. The incomplete level of detail means that some UML element is missing (e.g., class, attribute, method) due to incomplete requirements capture process or partial artefacts. The error resulting from incomplete diagrams is not measured. The initial level of details represents the complete artefacts with some data missing (e.g., data and return types, parameters). The complete level of details means that all the expected data are present. In our research, a reference test case defined by Fetcke (Fetcke, 1999b) was used to test the influence of the level of detail in the class diagrams to accuracy of the mapping. The chosen test case is referenced in the ISO/IEC TR 14143-3 Verification of functional size measurement methods (ISO/IEC TR 14143-3, 2003). Software size estimation using OOFP was applied twice. The first measurement was performed on initial class diagrams, with a complete set of classes, methods, attributes and associations. However, the method return type, parameters and their types, attribute types and the association's multiplicity were missing in this initial class diagram. The second measurement was performed on the complete domain class diagram. The measurement process was automated using the formal OOFP model. The calculation procedure can be formalised using a generalised structure notation (Fetcke, 1999a; Zivkovic *et al.*, 2003).

Table 3

Level of details for selected UML diagrams



		UCD	CD	SD
Incep. phase	Requirements	▤		
	Analysis	▤		
	Design	▤	▤	
Elab. phase	Requirements	▦	▤	▤
	Analysis	▩	▦	▦
	Design	▩	▩	▩

Levels of detail:
▤ - Incomplete
▦ - Initial
▩ - Complete

3.1. *Formal Model*

In general, the software system is composed of different data and transactional types. The number of data and transactional types, and their attributes, contribute to the size of the software system. The third component that has an influence on software size is the technical complexity of the solution. The universal function that maps application attributes into size is therefore:

$$FPC(a) = \left(\sum_i FPC_1(t_i) + \sum_j FPC_2(f_j) \right) * FPC_3(TC), \tag{E3}$$

where

$FPC(a)$ is the function that maps attributes of the application a into software size;

$FPC_1(t_i)$ is the function that maps transactional type t_i into size;

$FPC_2(f_j)$ is the function that maps data type f_j into size;

$FPC_3(TC)$ is the function that maps technical complexity of the anticipated solution for application a into a factor.

The total value for an application size is the sum of both parts multiplied by the factor of the solution's complexity. The factor can reduce or increase the overall size. However, it is not clear if the factor actually measures raw application size or is an attribute of the implementation and should be a part of the function that maps size to effort (Lokan, 2000). That is the reason why in this research, the function FPC$_3$ was not used in any calculations. The FPC functions for the OOFP method are presented in Eq. E4.

$$\begin{array}{ll}
FPC_1 = W_{EI}(N_d, N_r), & FPC_2 = W_{ILF}(N_d, N_g), \\
N_d = \sum_p S(p), & N_d = \sum_k S(k) + \sum_l \mathrm{O}(l), \\
N_r = \sum_p S^{-1}(p), & N_g = \sum_k S^{-1}(k) + \sum_l \mathrm{O}^{-1}(l), \\
S(p) = 1 \Rightarrow p = simple\_type, & S^{-1}(p) = 1 \Rightarrow p = complex\_type, \\
S(p) = 0 \Rightarrow p = complex\_type, & S^{-1}(p) = 0 \Rightarrow p = simple\_type, \\
 & \mathrm{O}(l) = 1 \Rightarrow l_{multiplicity} = 1, \\
 & \mathrm{O}(l) = 0 \Rightarrow l_{multiplicity} \neq 1. \\
\text{(a)} & \text{(b)}
\end{array} \tag{E4}$$

In the Eq. E4, the W_{EI} and W_{ILF} are functions that prescribe the number of function points according to the FPA element complexity tables. The function W has two parameters. For transactional functions, parameters are the number of data element types (N_d) and number of file types referenced (N_r). The OOFP method determines the value of N_d, N_r and N_g from the class diagram. The value is calculated differently for transactional functions (FPC_1) and data functions (FPC_2). For transactional functions, parameter N_d is calculated as the sum of values gathered from function S. The methods of the class represent the transactional functions, therefore i from Eq. E3 runs from 1 to the number of methods in the class. The function S returns 1 if the parameter p of the method i has a simple type and 0 in all other cases. The function S^{-1} is the opposite function, thus returns 0 if the parameter type is simple. For data functions, parameter N_g is used instead of N_r, representing the number of record element types. The value of N_d is determined from the attributes and associations of the class. In Eq. E4b, S is the function that returns 1 if the attribute k has a simple type and 0 otherwise. The function O evaluates association and returns 1 if the multiplicity of the relation is 1 and 0 otherwise. Functions W_{ILF} and W_{EI} are the step functions represented by discrete values with the following range:

$$W_{ILF} = \{7, 10, 15\},$$
$$W_{EI} = W_{EQ} = \{3, 4, 6\}.$$

The rules used for functions W are in accordance with the FPA method.

3.2. *Evaluation Process*

In the test case, an application portfolio (Fetcke, 1999b) conducted by five applications was used. All five applications were first converted into the class diagrams. The class diagram for application W is presented in Fig. 2.

The UML class diagrams were then converted into the XML Metadata Interchange (XMI) format. To automate the counting procedure, a Java tool was developed that parses the XMI file and uses parsed information to calculate the size of the software system. The functional size was measured twice, first on the initial class diagrams and secondly on the complete diagrams. The results are in Table 4. For the initial measurement (see first column), the results are zero for all cases. This is due to incomplete specifications, where none of the attributes or parameter data types are specified. Using the OOFP method the evaluation of the attributes without their types set is not defined. The possible solution

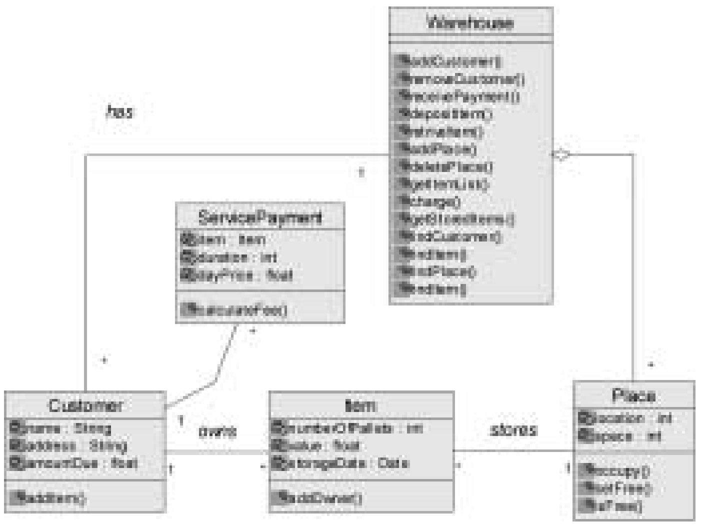


Fig. 2. Class diagram for application W.

is to treat unknown data types as simple data types. The results are in brackets. With considerably higher values from the values of the complete estimate we decided to reject this solution.

The anomaly in Table 4 forced us to reconsider Ram's (Ram and Raju, 2000) idea to include zero values in transactional complexity tables. For the OOFP2 test case all complexity tables were changed to include zero values. The complexity table for transactional functions was also changed to reflect the average number of parameters in the method. Lorenz (Lorenz and Kidd, 1994) reported measuring 0.7 parameters per method in Smalltalk projects. The value is specific for Smalltalk since in Smalltalk all the attributes are private. As a result, getter/setter methods influence the average values having zero or one parameter. We can argue that in modern programming languages such as Java and C# the same result is expected since C# introduces properties and Java recommends a JavaBean pattern. In the JavaBean pattern, all the attributes are accessed through getter/setter methods. Table 5 shows the changed values. Consequently the W functions are different. The results of the measurements are in Table 6.

In the column labelled OOFP, measurement values for the initial and complete specification is shown. Original FPA tables were used in the calculations. Results in the *complete* column are noticeably low. At least one data element type must be present for every method to rate it with low complexity and assign non-zero function points. The same is valid for logical files. Therefore, we introduce changes in the complexity tables to include a zero value and a lower complexity interval. The results are marked with OOFP2 and can be found in the second column. The remaining columns are shown for reference and are from the original document (Fetcke, 1999b).

Table 4
Application size in FP for the test case

Application	OOFP	
	Initial	Complete
W	0 (95)	33
M	0 (60)	16
C	0 (84)	27
LC	0 (83)	30
LS	0 (51)	16

Table 5
OOFP2 complexity table for transactional functions

	0–4 DET	5–10 DET	More than 10 DET
0 or 1 FTR	Low	Average	High
2 FTR	Average	Average	High
3 or more FTR	Average	High	High

Table 6

Measurement results

Application	OOFP		OOFP2		IFPUG FPA 4.1	Mk II FPA	FFP 1.0	COSMIC -FFP
	Initial	Complete	Initial	Complete				
W	0 (95)	33	51	83	77	72.96	102	81
M	0 (60)	16	35	46	40	32.40	52	38
C	0 (84)	27	35	65	49	46.72	65	51
LC	0 (83)	30	43	71	56	48.96	71	52
LS	0 (51)	16	27	37	31	24.00	41	29

3.3. *Statistical Evaluation of the Results*

In the test case with reference examples, the standard deviation (σ) between the results of measurements on incomplete and complete class diagrams using the OOFP2 method is 9.69 function points. Performing estimates on incomplete class diagrams may result in a 16% additional error. The maximum error was 29% and the minimum 7%. Student's t-Test helps us explain the results from Table 7. Table 7 shows the results of the Student's t-Test for the OOFP2 method. The first test, shown in the column labelled OOFP2-I, proves the hypothesis that initial estimates using incomplete class diagrams are significantly different from the estimates performed on complete class diagrams. The second test compares the performance of the original OOFP method using the FPA complexity tables with the OOFP2 method. The results in the column labelled OOFP-C again prove the hypothesis that tables with lowered values for the number of data types needed to reach the same number of function points, give significantly different results. However, it is difficult to conclude which result is better. Therefore, the results of the OOFP2 method are compared with the results of traditional methods. The results in columns three, four, five and six confirm the hypothesis, that OOFP2 gives a comparable result with all traditional FPA-based methods. None of the t-Test results show a significant difference in the values calculated with traditional methods in comparison with the values calculated with OOFP2 method.

With the use of Eqs. E3, E4 and the complexity tables, further analysis of the OOFP mapping is possible. The Eq. E5 expresses the number of function points for a class as a

Table 7

Student's t-Test results

	OOFP2-I	OOFP-C	IFPUG FPA 4.1	Mk II FPA	FFP 1.0	COSMIC-FFP
t-Test	0.044	0.004	0.417	0.230	0.674	0.425
stdev	15.1	14.4	18.1	18.8	21	19.2

function.

$$\begin{aligned}
&FPC_{class} = W_{ILF}(N_d, N_g) + \sum_{i=0}^{n} W_{EI}(N_d, N_r),\\
&MIN(FPC_{class}) = 7 + N_{methods} * 3 = 7 + N_{methods} * 3,\\
&MAX(FPC_{class}) = 15 + N_{methods} * 6.
\end{aligned} \tag{E5}$$

To evaluate the impact of the missing data in the requirements (e.g., data types, multiplicity, method parameters), the MIN and MAX functions are used. The function MIN returns a minimal value of function points for a class. The data functions contribution is simplified, thus it can have only three values. The value is set to 7 and is valid for cases where attributes do not have their data types set and association multiplicity is unknown. The sum of all attributes and associations do not exceed 19. The contribution of the transactional functions depends on the number of methods in the class. The average number of methods in the sample of 14 applications using different programming languages provided by Hericko (Hericko, 1998) is 11. Therefore, the minimum size for a class with the average number of methods is 40 FP. The MAX function returns the maximum possible value considering the number of methods. Using the average number of methods in a class, the size is 81 FP. With 50% the maximum difference is quite high. These results must be interpreted with caution, however, since values represent the complete interval. In the reference examples, the average difference is much lower, around 16%. In the ISBSG repository (ISBSG, 2001) containing data for 345 diverse software projects, the average project size is 535 FP. The 16% estimate error can be converted to the effort in person hours for different platforms and language combinations. In the future, the repository data will be used to improve early estimates and a principle of self-learning will be introduced into software-size estimations. Several class diagram abstraction levels will be defined and merged with the software development process to indicate points where a method that will result from the research can be used.

4. Conclusion

With new programming languages like Java and C# the need to provide the standard OO-to-FPA mapping becomes even more important. In this paper, important existing OO-to-FPA mappings were compared. The existing mappings are similar, each having some specifics. In our research we have defined a common mapping based on the existing OO-to-FPA mappings. The approach we proposed is based on class diagrams, thus not applicable during requirement gathering in the early project phases. It could be used after the initial class diagram is composed. Two distinct sources of inaccuracy in the data used for size estimations were identified. The data in the class diagrams may be incomplete, in that it does not show all the methods and attributes. This was treated as an incomplete requirement and was not measured in this research. The second situation arises when all the requirements are captured with different levels of detail. The research on the reference test case showed that a lack of data in the class diagrams significantly influences the estimation accuracy regardless of the complexity tables used in the measurement. Therefore the OOFP method is not appropriate for use during early development phases. In our research a new complexity table for transactional functions was proposed. According to our test sample, the changed complexity tables give more accurate results on detailed domain class models available during the analysis and design than original FPA complexity tables. Statistical tests also showed that these results are equivalent to the results of four different industry-leading FPC methods.

In future work, a new method will be created to attempt to get equally good results during the whole development cycle. To do so, more examples have to be tested to extend the validity of the new complexity tables across various applications. In addition, the source of information has to be broadened to include other UML diagrams organized in models for a specific development phase. Early estimates will be based on repository containing historical estimates data and a self-learning estimation model.

5. Appendix

Table 8

Hericko's data set

Programming language	No. of Classes	No. of Methods	Average No. of methods /class
Delphi (I)	50	521	10.42
Delphi (II)	45	476	10.57
Delphi (III)	38	252	6.63
Smalltalk (I)	167	2371	14.19
Smalltalk (II)	16	161	10.06
Smalltalk (III)	13	99	7.61
Smalltalk (IV)	38	426	11.21
Smalltalk (V)	34	793	23.32
C++ (I)	26	209	8.03
C++ (II)	64	1280	20
C++ (III)	69	483	7
C++ (IV)	78	546	7
C++ (V)	16	112	7
Java	146	1628	11.15

References

OMG (2001). *OMG Unified Modelling Language Specification*, Object Management Group, version 1.4, September 2001.

Albrecht, A. (1979). Measuring application development productivity. In *IBM Applications Development Symposium*, October 14–17. pp. 83–92.

Antoniol, G., C. Lokan, G. Caldiera and R. Fiutem (1999). A function point-like measure for object-oriented software. *Empirical Software Engineering*, **4**(3), 263–287.

Fetcke, T. (1999a). A generalized structure for function point analysis. In: *Proceedings of International Workshop on Software Measurement (IWSM'99)*. Mont-Tremblant, Canada. pp. 1–11.

Fetcke, T. (1999b). *The Warehouse Software Portfolio*: *A Case Study in Functional Size Measurement*. Technical Report Number 99-20, ISSN 1436-9915, TU Berlin.

Fetcke, T., A. Abran and T.-H. Nguyen (1997). Mapping the OO-Jacobson approach into function point analysis. In *Proceedings of IFPUG 1997 Spring Conference*. pp. 134–142.

Hericko, M. (1998). *Quality of Object-Oriented Software Development*. PhD thesis. University of Maribor, Maribor.

IFPUG (1999). *Function Point Counting Practices Manual*, 4.1 ed. International Function Point Users Group, Westerville, Ohio.

ISBSG (2001). *Practical Project Estimation, A Toolkit for Estimating Software Development Effort and Duration*. International Software Benchmarking Standards Group, March 2001.

ISO/IEC TR 14143-3 (2003). *Information Technology – Software Measurement – Functional Size Measurement, Part 3*: *Verification of Functional Size Measurement Methods*. ISO/IEC, First edition.

Jacobson, I., and M. Christerson (1992). *Object-Oriented Software Engineering*: *A Use Case Driven Approach*. Addison-Wesley.

Lokan, C.J. (2000). An empirical analysis of function point adjustment factors. *Information and Software Technology*, **42**(9), 649–659.

Lorenz, M., and J. Kidd (1994). *Object Oriented Software Metrics*. Prentice Hall.

Ram, D.J., and S.V.G.K. Raju (2000). Object oriented design function points. In *Proceedings of the First Asia-Pacific Conference on Quality Software*. Hong Kong. pp. 121–126.

Uemura, T., S. Kusumoto and K. Inoue (1999). Function point measurement tool for UML design specification. In *Proceedings of the Sixth International Symposium on Software Metrics*. Institute of Electrical and Electronics Engineers. pp. 62–69.

Zivkovic, A., M. Hericko and T. Kralj (2003). Empirical assessment of methods for software size estimation. *Informatica*, **27**(4), 425–432.

A. Živkovič is a teaching assistant at the University of Maribor where he is working on his PhD thesis. His research work covers different aspect of object technology with the emphasis on UML, software processes, Java platform and metrics. He gained his practical experiences in cooperation with industry on several projects. A. Živkovič received his diploma in 1996 and master degree in 2000 both from University of Maribor.

M. Heričko is an associate professor at the University of Maribor, Faculty of EE&CS, Institute of Informatics. He received his MSc (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 modeling.

B. Brumen received his PhD degree from University of Maribor in 2004. He is a teaching assistant at the University of Maribor. His research interests include databases and software security. He is author and co-author of articles published in different scientific journals.

S. Beloglavec is a teaching assistant at the University of Maribor where he is working on his PhD thesis. His research work focuses on design patterns, component oriented development. He gained his practical experiences on several research and industry projects. S. Beloglavec received his diploma 1997 from University of Maribor.

I. Rozman received the PhD degree from University of Maribor in 1983. He is a full professor of software engineering at the Faculty of EE and CS. Prof. Rozman is author and co-author of numerous articles published in different scientific journals and a member of several program committees at domestic and international conferences. He is currently rector of the University of Maribor.

Klasių diagramos detalumo įtaka programinės įrangos apimties vertinimui

Aleš ŽIVKOVIČ, Marjan HERIČKO, Boštjan BRUMEN,
Simon BELOGLAVEC, Ivan ROZMAN

Programinės įrangos apimtis – svarbus veiksnys planuojant projektus programų sistemoms sukurti. Žinomi keli programinės įrangos apimties vertinimo metodai; daugumas iš jų apimtį vertina panaudodami funkcinius taškus. Funkcinių taškų metodas, kaip metodas, nepriklausantis nuo konkrečios technologijos ir išreiškiamas savarankiško abstrakcijos lygmens terminais, buvo pasiūlytas Albrechto. Tačiau šį abstraktų aparatą sunku pritaikyti šiuolaikinėms technologijoms. Todėl daugelis mokslininkų siūlo papildomas taisykles ir atvaizdžius, pritaikytus objektinei paradigmai.

Šiame straipsnyje aptariamos ir lyginamos kelios atvaizdavimo strategijos. Panaudojant atskleistus šių strategijų panašumus, pasiūlyta bendresnė atvaizdavimo strategija. Ši strategija buvo testuota, panaudojant penkis modelinius pavyzdžius. Testavimo scenarijaus tikslas buvo įvertinti, kokį poveikį programinės įrangos apimties vertinimui turi klasių diagramos detalumo laipsnis. Nors objektinės programų sistemos apimties vertinimo problema ir nebuvo galutinai išspręsta, gauti rezultatai yra vertingas įnašas į jos sprendimą.