A COMPARISION OF
OBJECT ORIENTED SIZE EVALUATION TECHNIQUES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

HATİCE SİNEM SIRAKAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JANUARY 2003

Approval of the Graduate School of Informatics Institute.

Prof.Dr. Neşe YALABIK

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof.Dr. Semih BİLGEN

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Onur DEMİRÖRS

Supervisor

Examining Committee Members

Prof.Dr. Semih BİLGEN

Assoc.Prof.Dr. Onur DEMİRÖRS

Assist..Prof.Dr. Ali DOĞRU

Assist.Prof.Dr. Erkan MUMCUOĞLU

Dr. Altan KOÇYİĞİT

ABSTRACT

A COMPARISION OF

OBJECT ORIENTED SIZE EVALUATION TECHNIQUES

Sırakaya, Hatice Sinem

M.Sc., Department of Information Systems

Supervisor: Assoc. Prof. Dr. Onur DEMİRÖRS

January 2003, 126 pages

Popular Object Oriented size metrics and estimation methods are examined. A case study is conducted. Five of the methods ("LOC", "OOPS", "Use Case Points Method", "J.Kammelar's Sizing Approach" and "Mark II FP") are applied to a project whose requirements are defined by means of use cases. Size and effort estimations are made and compared with the actual results of the project.

Keywords: Object Oriented Project Size Estimation, Use Case, LOC, Point Value, Use Case Points, Component Object Points (COPs), MK II FP.

ÖZ

NESNE TABANLI YAZILIM BOYUTU DEĞERLENDİRME

TEKNİKLERİNİN KARŞILAŞTIRILMASI

Sırakaya, Hatice Sinem

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç.Dr. Onur DEMİRÖRS

Ocak 2003, 126 sayfa

Yaygın nesne tabanlı yazılım boyutu değerlendirme tekniklerinden bir kısmı incelenmiştir. Bir alan çalışması yapılmıştır. Tekniklerden beş tanesi (“LOC”, “OOPS”, “Use Case Points Methodu”, “J.Kammelar’ın Boyut Değerlendirme Yaklaşımı” ve “Mark II FP”) gereksinimleri use case’lerle tanımlanmış bir projeye uygulanmıştır. Boyut ve efor değerlendirmeleri yapılmış ve projenin gerçek sonuçlarıyla karşılaştırılmıştır.

Anahtar Kelimeler: Nesne Tabanlı Proje Boyutu Değerlendirmesi, Use Case, LOC, Point Value, Use Case Points, Component Object Points (COPs), MK II FP.

ACKNOWLEDGEMENTS

I would like to thank my advisor Assoc. Prof. Onur DEMİRÖRS for his support, guidance and insight throughout the development of this thesis. I also thank Alpay KARAGÖZ for his help in providing the project information. I want to thank my mother for her support all the way. Finally, I gratefully thank to my husband for his support and encouragement in completing this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

3D	Three Dimensional
ADTs	Abstract Data Types
AI	Artificial Intelligence
ANGEL	Analogy Software Tool
ASL	Algebraic Specification Language
AVC	Average Number of LOC
CAD	Computer Aided Design
CASE	Computer Aided Software Engineering
CLOC	Commented Lines of Code
COCOMO	Constructive Cost Model
COPs	Component Object Points
COSMIC	Common Software Measurement International Consortium
DET	Data Element Type
DIT	Average Depth of Inheritance Tree
DSI	Delivered Source Instructions
EI	External Inputs
EIF	External Interface Files
ES	Executable Statements

ECE	External Control Entry
ECX	External Control Exit
EO	External Outputs
EQ	External Inquiries
FFP	Full Function Points
FP	Function Point
FPA	Function Point Analysis
FPI	Function Point Index
FUR	Functional User Requirements
GUI	Graphical User Interface
IFPUG	International Function Point Users Group
ICASE	Integrated Computer-aided Software Engineering
ICR	Internal Control Read
ICW	Internal Control Write
ILF	Internal Logical Files
ISO	International Organization for Standardization
LOC	Lines of Code
MIS	Management Information Systems
MK II FPA	Mark II Function Point Analysis
NCLOC	Non-commented Lines of Code
NOC	Average Number of Children per Base Class

OO	Object Oriented
OOPS	Object-Oriented Project Size Estimation
POPs	Predictive Object Points
RCG	Read-Only Control Group
SELAM	Software Engineering Laboratory in Applied Metrics
SOM	Statistical Object Model
SPR	Software Productivity Research
SSM	Software Sizing Model
TCA	Technical Complexity Adjustment
TCFs	Technical Complexity Factors
TDI	Total Degrees of Influence
TLC	Number of Top-Level Classes
UAW	Unadjusted Actor Weight
UFP	Unadjusted Function Points
UCG	Update Control Group
UCP	Adjusted Use Case Points
UUCW	Unadjusted Use Case Weights
UUPC	Unadjusted Use Case Points
VPM	Vector Prediction Model
VSM	Vector Size Measure
WMC	Weighted Methods per Class

CHAPTER 1

1. INTRODUCTION

In recent years, software has become the dominant component in systems and, its size and complexity has increased. With this increasing size and complexity, management has become a difficult issue: increasing cost overruns, schedule delays and poor-quality projects have resulted in a software crisis [25]. To solve these problems, software developers and managers have started to look for new software development techniques to understand and control their projects better.

By the mid 80's, object oriented (OO) software development has come-out to become one of the solutions to this crisis [8], [13], [20], [45], [77]. This was a new development technique in which the real world was represented by objects. In some respects, this technique was similar to the traditional procedural programming e.g. usage of abstract data types (called classes in OO software development) and procedures (called methods in OO software development). On the other hand, many new concepts such as classes, methods, message passing, inheritance, polymorphism, and encapsulation have appeared with OO software development [4], [8], [12], [13], [19], [76]. However, it was not enough to find out such a new technique. The need to successfully manage such OO systems has resulted in many new metrics suitable for OO structure or the application of traditional ones to these systems.

In fact, metric usage is very important in OO software development to better control its complex structure [13] and to meet its promise to solve the crisis [19]. Today there are many metrics being used in the project management and control of OO systems. *"Size"* metric is one of the most important of them. All other metrics (Cost, effort, complexity, productivity, quality etc.), which are used for management activities, project control and resource allocation during the development process, are based on size.

There are two approaches for the measurement of OO software: Whereas some practitioners say that traditional metrics are not suitable for OO software and new ones are needed [13], the others believe that traditional metrics can be applied to OO software, may be with some modifications and additions [45], [66], [76].

Whatever the approach is, today's all existing metrics have their criticisms, in part because of the general difficulty of the estimation process [27], [72], [29] and the immaturity of the measurement science for the software engineering [38], [40], [72]. First of all, good theories are necessary for correct measurements. However, most metrics lack this theoretical basis or depend on unclear ones [19], [20], [28], [43], [45], [72], [78]. In fact, the mappings from the real world domain to the metric models are usually not well defined, there is a lack of good empirical relational systems and there is a general misunderstanding of software attributes and scientific measurement in software engineering [28], [37], [29]. Secondly, there are no global standards on procedures and methods for metric definitions and usage [8], [40]. This results in metrics having various definitions. For example, today there are many different counting rules for LOC and FP. Thirdly; many metrics are never validated or validated in different ways [38], [40], [29]. Because of all these reasons, most existing metrics lack necessary measurement properties and the rigor, which is available in other engineering disciplines [19], [20], [43], [79], [29]. Also because most OO metrics are composite ones and/or using weights for adjustment, they are difficult to calculate on interval ratio scales and they lack sensitivity [13].

Besides these general problems, there are also problems on the application of traditional metrics to OO software. With OO software development, many new concepts such as classes, methods, massage passing, inheritance, polymorphism and encapsulation have appeared. On the other hand, traditional metrics were designed in a way suitable for the structured programming in which none of the above concepts exists [4], [8], [13], [19], [20], [38], [66], [68], [76] and most of these metrics accept code consisting of only text, so it is not clear how to apply them to OO structure [29]. Also, as mentioned earlier, most traditional metrics lack theoretical basis. So, it is difficult to understand and apply them to OO software [13]. Finally, there is one criticism specific to OO size metrics: Some practitioners say that OO size has more than one dimension i.e. functionality, length, reuse and complexity. However, traditional metrics measure only the functionality dimension of OO software size [57], [77].

So far many size estimation researches have been done on OO software. Some of these are based on applying the existing traditional size metrics to OO software whereas others are new ones just designed for OO software. However, still there is no completely rational and satisfactory model to measure the size of OO software.

1.1 Scope and Outline of the Thesis

This thesis takes into account the previous attempts and studies made in the field of OO size estimation and by a case study try to find the answers to the questions of *"What are the current state of available OO size metrics?"* and *"If the requirements of a project are defined in terms of a use case model, which OO size metrics can better estimate the project size?"*

A completed industrial project is considered as a case study for the size estimations. From the size metrics and methods mentioned in this thesis, five of them are chosen to estimate the project size. These are “Lines of Code”, “Object Oriented Project Size Estimation”, “Use Case Points Method”, “J.Kammelar’s Sizing Approach” and “Mark II Function Points”. The calculations are made manually by applying guidelines defined in Section 2.2 of the related sizing methods. Then effort estimations are made and compared with the actual results of the project.

In the scope of this thesis, the projects, whose requirements are defined in terms of a use case model, are taken into consideration. Thus the comparison results of this thesis can be applicable only to the projects defined by such models.

In Chapter 2, a survey on related subjects i.e. OO size metrics and methods is presented. Discussions on the related work are given.

In Chapter 3, application of the selected size metrics and methods to a project are given. Also, results are evaluated and compared with each other.

In Chapter 4, general discussions and concluding comments are presented. Also, some future work is suggested.

CHAPTER 2

2. SOFTWARE SIZE ESTIMATION

This chapter is a survey on software size measurement: Software Size and OO Size Metrics and Methods.

2.1 Software Size Measurement

Size is one of the most important measures for early project estimation and better project control. Following gives a brief overview on software size measurement concepts.

2.1.1 Software Size

Project estimations begin with estimating the size of software to be produced. It is critical to accurately estimate the size early in a project in order to get accurate effort, schedule and cost estimations for monitoring and improving the project progress, productivity and quality. We also need to know the size of software to predict future maintenance requirements. Besides these, software size estimation has an important role in the normalization of other metrics such as measuring the defect density in terms of defects per LOC [4], [29].

Software size measures are divided into two types [51]:

- *Technical measures:* size software products from the developer's point of view and they are used for efficiency analysis. Lines of Code (LOC) is an example of such a measure.
- *Functional measures:* size software products from the user's point of view and they are used for productivity analysis and building estimation models. Also, as being technology and implementation independent, they can be used in productivity comparisons for different techniques and technologies [54]. Function Point Analysis (FPA) is an example of such a measure.

Software size estimation is especially important for Project Planning. In fact, project planning begins with software size estimation. Many methods, usually LOC and FPA, are used for this purpose.

Once the software size is estimated, it is possible to derive the necessary cost for the project. The project costs are divided into three main categories [63]:

- Hardware costs,
- Travel and training costs
- Effort costs.

Since effort costs form the largest part of the total project cost, usually "cost estimation" means "effort estimation" [65], [29]. Effort estimations are used to determine the amount of work necessary for a project measured in labor hours. In most cost estimation models size is taken as the main input parameter and some other cost drivers are used to calculate the final effort and cost.

The general formula for effort is:

$\text{Effort} = A + B*\text{Size}^C$

where A, B and C are environmental depended constants.

This effort estimation is used to determine the project schedule. For example, the Basic Model of COCOMO estimates the required effort (measured in Staff-Months SM) based on size (measured in thousands of Delivered Source Instructions KDSI) as [26]:

$SM = a * (KDSI)^b$

Then the development schedule (Time of Develop TDEV) of the project in months is calculated as:

$TDEV= c * (SM)^d$

Usually Gantt or PERT charts are used to report such scheduling activities.

When the effort and cost are estimated, the productivity can be measured. Generally productivity estimations base on models that divide size by effort (for example productivity measured in KLOC / person-month). Input to these models also includes other factors such as expertise of the development organization, complexity of the problem, analysis and design techniques, programming language, reliability of the computer system, and availability of hardware and software tools [17].

Another use of size is in quality estimations. Software quality is related with correctness, maintainability, integrity and usability of software [17]. To measure these dimensions of quality, some predictors such as size and complexity are combined with some outcomes such as defects. The most common quality metric is defects per size. Size is also related with quality by means of productivity. In fact, some productivity models are used with quality models. Advanced COCOMO is such an example [29].

2.2 OO Size Metrics and Methods and Related Work

With today's new development techniques, understanding of software size has become a four dimensional concept: functionality, length, reuse and complexity [57], [77]. With this changing concept, software size estimation process has involved a wide range of metrics and methods from the traditional to the new ones. Following gives a brief overview on some of these software size metrics and methods in the context of OO (see Appendix B for a summary table).

2.2.1 Expert Estimations

Expert estimations are the subjective methods to estimate the size of a software project. In spite of their subjectivity, these methods are still the most widely used ones then the more objective methods because of [53]:

- The lack of necessary information at the beginning of the project,
- The specificity of the domain addressed,
- The effort and time required,
- The need to introduce a vocabulary foreign to stakeholders without a software background.

The most known of these methods is the *"expert judgment"* method in which experts make predictions on the size of a project based on their past experiences from local or industry-wide observations [27], [29].

Although it is so common, expert judgment have serious problems. The estimates are based on the quality and experience of the experts [29]. The psychological and personal factors and the level of necessary system information of the experts can affect the judgments [46]. Using out-of-date data or wrong memories from the past projects can result in inaccurate estimates [72]. Also, because estimates are implicit, they cannot be repeated [65]. And finally, all these inaccurate size estimations inevitably cause doubtful schedules and budgets [53].

A more formal form of expert judgment is the *"estimation by analogy"* In this method, first the important characteristics of the project is defined. Then by using these characteristics, the project is compared with one or more past projects of known sizes. The similarities and differences are found and some adjustments are made to estimate the current project size [65], [29]. In the comparisons, the similarities are determined by the closeness of the characteristics that is calculated as distance in n-dimensional space [18]. Estimation by analogy can also be used to estimate cost, effort and schedule.

The method has some advantages. First of all, it is very easy to understand. It can be used even when no statistical relationship exists. No calibration for the new development area is needed. It can be automated (e.g. ANGEL tool) [65]. Also, the comparison analyses are usually documented so they can be used in later estimates and reviews [29]. However, the method has also some problems. Since it is difficult to find the analogies and determine the similarities, application of the method needs a large amount of time. In addition, there are some unclear subjects about the method such as using old data points and effect of different variables on different data sets [65].

To say, although estimation by analogy is a better method then expert judgment, still more study is needed in this area.

In addition to the above methods, there are also some statistical sizing methods for the subjective judgments. *"PERT"* is the most popular one. For PERT estimation, first during the planning and requirements phase, the project is divided into its major functions. Secondly, the size of each function is estimated. Finally, these size estimations are summed to get the total project size [46].

Since the estimates are based on expert judgment or analogies, PERT is a subjective method [46]. To decrease its subjectivity, Putman and Fizsimmons [58] defined an adaptation of PERT. In their adopted method, the size is estimated not as a single point value, but as a statistical distribution [27]. Here, for each function, the smallest, the most likely and the largest sizes are estimated and by the below formula, the total project size is found [46], [58]:

$\text{Size}_i = (s_i + 4m_i + l_i) / 6$

$\text{Size}_s = \sum \text{Size}_i$

The deviations for each function and for the total project are calculated as [1], [58]:

$D_i = (s_i - l_i) / 6$

$D_s = (\sum D_i^2)^{1/2}$

This statistical distribution can also be used for effort and schedule estimates [27]. On the other hand, since the estimates are based on expert judgment or analogies, the previously mentioned problems of expert estimates also exist for this method.

Another statistical sizing method is the *"software sizing model (SSM)"*. SSM was developed in 1980. It is a method based on expert judgment. SSM is an automated method and since 1985, it is being automated by the cost estimation tool PRICE [15]. First the user(s) decompose the system into modules. Then, the user(s) provides the module and project data to SSM. From this information, SSM generates customized input screens for the project. After that, by using the screens, the user(s) provides the required input data sets (i.e. pairwise rankings, PERT estimation for each module, sorting and relative size ranking data). At least one reference module of known size must be available for the ranked modules. To calculate other modules' sizes, these reference values are used [15], [27]. The inconsistencies between the input data sets are resolved by statistical relations [27]. The execution of SSM results in size estimates and standard deviations for each module and for the total project. Also, for the total project size, confidence intervals are provided [15], [27].

There are some pluses of SSM. First of all, SSM is an automated, interactive, self-documented method. It is not database dependent. In any development phase in which modularization is possible, the method can be used. SSM's final size estimation can be in any form independent of how the reference modules sizes' are defined. When compared with PERT, SSM performs better results [15]. However, since it is an expert judgment based method, rankings and selection of reference modules and as a result the accuracy of the final estimation can be affected by the quality of the estimators' judgments.

A later study similar to the concepts of SSM is the *"paired comparison"* method for software sizing. This is a method used when a measurement instrument or an acceptable measurement scale does not exist [52], [53]. In paired comparison, one or more experts estimate the relative largeness of n entities (requirements, use cases, modules, objects etc.) with respect to each other. First the entities are arranged from largest to smallest. Then a judgment matrix is established in which relative sizes of each entity are defined. From these a ratio scale is derived. By using this ratio scale and at least one reference value (i.e. the size of an entity which is available from the past projects), the absolute values of the entities are calculated [52], [53].

Paired comparison is a method, which can be used at the early stages of a project where little information is available. In addition, since each entity is compared with others, the errors and inconsistencies can be easily determined.

Automation is also possible (e.g.. MinimumTime from Ericsson) [52]. On the other hand, some care must be taken when using paired comparison:

- The functional and technical aspects of the system must be understood very well to make judgments more accurate [52], [53],
- For large number of entities, work must be divided into smaller judges to increase the accuracy [52],
- In choosing a reference value, care must be taken. Too large or too small values can result in under or overestimation [52],
- Establishing a verbal scale can be practical in calculations [53].

Like many other methods, for paired comparison, more study is needed to validate the method's usefulness.

2.2.2 Lines of Code

"Lines of Code (LOC)" is the oldest and most widely used traditional size metric. Since it is objective, easy to understand and measure, LOC has been used for measuring length, normalization of other metrics and as an input to cost/effort, productivity and quality estimations [30], [50], [29].

Today there are many definitions of LOC being used for different purposes [29]. One of them is *"noncommented lines of code (NCLOC)"*. In NCLOC, the blank lines and comments are not counted. Therefore, this method is not suitable for purposes such as determining the computer storage requirements.

Another method is to count not only NCLOC but also the *"comment lines (CLOC)"*. The total size is calculated as:

LOC = NCLOC + CLOC

Counting number of *"executable statements (ES)"* is also possible. In this method, each statement on the same physical line is counted distinctly. Comment lines, data declarations and headings are ignored.

To measure the amount of delivered code rather than the written code, *"delivered source instructions (DSI)"* can be used. DSI counts each statement on the same physical line distinctly. It excludes comment lines and includes data declarations and headings.

Rather than using LOC, length of a program can be measured by using the *"bytes of computer storage required for the text"*, or by using the *"number of characters in the program text"*. Both of these methods are easy to understand and collect. Also, since LOC, bytes of computer storage required for the text and number of characters in the program text are on the ratio scale, they can be easily converted to each other.

With so many definitions, LOC has some problems:

- Since length is an important input for many estimation methods, it must be obtained accurately at the early stages of a project. However, accurate measurement of LOC is possible only at the later stages of a project when the code is written [30], [44], [50], [76].
- There is no standard definition of what a LOC is [40], [50], [76], [29] and existing ones are conflicting with each other. Therefore, it is difficult to compare such measures and confusion can appear in other estimates using LOC as an input.
- LOC depends on the programming language. Therefore, programs written in different languages cannot be directly compared [30], [44], [50], [29].

Especially in effort, functionality and complexity comparisons using LOC, some conversion factors must be used [29].

- When using LOC, two programs of the same size are always accepted to be equally complex [76].
- LOC does not count the different levels of complexity of different lines [76].
- Estimation of software size in LOC in the early phases of a project when no code is available can only be done by expert estimation. Such a LOC can differ from expert to expert.

The application of LOC to OO systems has not only the above problems, but also many other ambiguities. Since LOC is a size measure introduced for the structured programming, it does not take into account the OO concepts such as reuse, inheritance, polymorphism and usage of class libraries. Therefore, it is not clear how to count LOC when considering such concepts [40], [76].

One application of LOC to OO systems is based on use cases. J. Smith [67] in his article defines a framework to estimate the system size (in terms of LOC) and corresponding effort depending on the number of use cases for that system. The steps are as follows:

1. The structural hierarchy of the whole system is defined. 5 levels are proposed:

Level 4 – System of Systems

Level 3 – System

Level 2 – Subsystem Group

Level 1- Subsystem

Level 0 – Class

where Class and Subsystem are defined in UML.

2. For each level, the number of use cases is determined.

3. The total system size is roughly calculated by using the following formula:

Total Size = [(N1/10)*7+(N2/10)*56+(N3/10)*448+(N4/10)*3584] ksloc

where N1, N2, N3 and N4 are the number of use cases of the corresponding levels.

4. Table 2.1 is used to determine effort per use case for the corresponding level. This table was formed by using the Estimate Professional^{TM} toll based on COCOMO 2.0 and Putnam's SLIM models where C++ was taken as the base language.

Table 2.1 Efforts per Use Case

Size (slocs)	**Effort hrs/use case Simple business system**	**Effort hrs/use case Scientific system**	**Effort hrs/use case Complex command and control system**
7000 (Level 1)	55 (range 40-75)	120 (range 90-160)	260 (range 190-350)
56000 (Level 2)	820 (range 710-950)	1700 (range 1500-2000)	3300 (range 2900-3900)
448000 (Level 3)	12000	21000	38000
3584000 (Level 4)	148000	252000	432000

By using the size to effort relationship defined in COCOMO 2.0 i.e. Effort = $A*(Size)^{1.11}$, and the above Total Size formula, for each level the effort multipliers are calculated as:

For Level 1, $EN1= (0.1*N1+0.8*N2+6.4*N3+51.2*N4)^{0.11}$

For Level 2, $EN2 = (0.0125*N1+0.1*N2+0.8*N3+6.4*N4)^{0.11}$

For Level 3, $EN3 = (0.00156*N1+0.0125*N2+0.1*N3+0.8*N4)^{0.11}$

For Level 4, $EN4 = (0.00002*N1+0.00156*N2+0.0125*N3+0.1*N4)^{0.11}$

5. For each level the effort per use case is calculated by multiplying ENi by the corresponding effort/use case value defined in Table 2.2.2.1. For example, for a simple business system consisting of only Level 1 use cases with an effort multiplier of 1.2, the effort per use case for this level is 1.2*55=66 hrs/use case.

6. Finally, the total effort for each level is calculated as:

Total Effort = Effort per Use Case for Level i*Ni hrs

This method makes the size and effort estimations possible for any level of use cases [67]. It is an easily understandable method. Some automated tools can be used for calculations. However, it is limited to C++ and equivalent level of languages. And like many other methods, more testing and reestimation of parameters are necessary.

2.2.3 Software Science

"Software Science" is a scientific model developed by Maurice H. Halstead [34] during 1970s to measure software size and complexity.

In Software Science, a program is defined as a collection of tokens known as operators and operands [64], [29]. Operators are the symbols or keywords that show the algorithmic actions, and operands are the symbols used in data representations.

The basic metrics used are:

μ_1 = number of unique operators.

μ_2 = number of unique operands.

N_1 = total occurrences of operators.

N_2 = total occurrences of operands.

The program length is calculated as:

$N = N_1 + N_2$

And the program vocabulary is calculated as:

$\mu = \mu_1 + \mu_2$

From these metrics, other additional measures are calculated. These are:

- Volume of a program (V) = $N * \log_2\mu$
- Program level (L) = V* / V

  where V* is the potential volume i.e. minimal size of the program.
- Difficulty of a program (D) = 1 / L
- Effort (E) = V / L = D * V
- Time for effort (T) = E / 18

For Software Science, all the four internal attributes (μ_1, μ_2, N_1, N_2) are on an absolute scale. Also, the three views of size i.e. length, vocabulary and volume are valid measures from the measurement theory perspective [28], [29].

Although Halstead's model seems to base on valid scientific theories, and got great interest from the software community, it has also some serious problems:

- Since there is no detailed design in the early phases of a project, it is difficult to calculate the Halstead length [72].
- In fact, software science metrics are defined to count the operators and operands of algorithms rather than programs, and are based on studies written in Fortran and Algol. Therefore, it is difficult to determine how to count and even discriminate between the operators and operands of programs written in other languages. However, this problem can be solved by using automated tools [22], [64].
- There are criticisms on the validity of the experimental data such as using too small sample sizes and single subjects in the experiments [64].
- The program level depends on not only the program language but also the experience and style of the programmer. So, it is difficult to test the validity of this formula [64].
- In the derivation of some formulas, there is a lack of theoretical justification [64]. Especially the effort and time metrics are crude prediction systems [29].

The application of Software Science metrics to OO systems is also possible. However, such an application has its own difficulties because of the different structure of such systems [76]. The most important difficulty is to count the operators and operands, and to find a reference for each use of a method. In OO approaches, methods are accepted as operators and variables as operands. From these definitions, it is unclear if an object is an operator or an operand. Also subjects specific to OO systems such as inheritance, polymorphism and existence of some dynamic values make the above problems more difficult to solve.

Because of its limitations, today this model is not being widely used. However, it has some good ideas about size measurement that can be used in future studies.

2.2.4 Function Point Analysis

"Function Point Analysis (FPA)" is developed by Allan J.Albrecht in 1979 while in IBM to size business information systems [5]. It is based on the idea of measuring the amount of functionality delivered to the user in terms of *"function points (FPs)"*.

In FP calculation, the system components are classified from the end-users view as external inputs, external outputs, external inquiries, external interface files and logical internal files and they are counted. Then weights are assigned for each of these counts. Finally, some complexity factors are used for adjustment to get the final FP.

The detailed steps for these calculations are [6], [73], [29]:

1. The system components are classified as:

External Inputs (EI): items provided by the user that describe distinct application-oriented data.

External Outputs (EO): items provided to the user that generate distinct application-oriented data.

External Inquiries (EQ): each unique input/output combination, where an input causes and generates an immediate output.

External Interface Files (EIF): files passed or shared between applications.

Internal Logical Files (ILF): each major logical group of user data or control information in the application.

2. The system components are also classified as *"simple"*, *"average"* and *"complex"* depending on the number of data elements and records they contain and their some other properties. For each component, a weight is assigned and *Unadjusted Function Point (UFP)* is calculated by using Table 2.2.

Table 2.2 Weight Factors for FPA

Component Description	**Simple**	**Average**	**Complex**	**Total**
External Inputs	…* 3	…* 4	…* 6	………
External Outputs	…* 4	…* 5	…* 7	………
External Inquiries	…* 3	…* 4	…* 6	………
External Interface Files	…* 7	…* 10	…* 15	………
Logical Internal Files	…* 5	…* 7	…* 10	………
			UFP	………

3. 14 *Technical Complexity Factors (TCFs)* (Table 2.3) are rated from 0 to 5 based on their degree of influence to the system based on Table 2.4 and summed to get *Total Degrees of Influence (TDI).*

Then TCF is calculated as:

TCF = 0.65 + 0.01*TDI

4. The final FP is:

FP = UFC * TCF

where FP is a dimensionless number on an arbitrary scale [73].

Although FPA aims to solve the problems associated with LOC, it has also many criticisms. Here is a list of its advantages and criticisms on it:

- Unlike LOC, FPA is independent of languages, tools and methodologies for implementation [4], [6], [39], [40], [50], [70].
- Early size estimation is easier with FPA, because the necessary FPA information can be obtained from user requirements, design specifications, source listings, initial proposals or even from live systems regardless of the level of detail available [6], [32], [39], [40], [44], [49], [71], [73]. Such requirements also provide an early validation for the method [6].
- Since FPA depends on user requirements, it can be easily understood and accepted by non-technical users [44], [71], [73].
- As the requirements change, reestimation is possible, however, there is a 400% to 2000% increase between the early estimates and later ones [50], [29].
- FPA has the largest statistical support [32].
- Today there are many cost estimation tools using FPA in their calculations [40].

Table 2.3 Technical Complexity Factors for FPA

Description
Reliable back-up and recovery
Distributed functions
Heavily used configuration
Operational ease
Complex interface
Reusability
Multiple sites
Data communications
Performance
Online data entry
Online update
Complex processing
Installation ease
Facilitate change

Table 2.4 Complexity Rates for FPA

Description	Complexity Rate	Description	Complexity Rate
Not present or no influence if present	0	Average influence	3
Insignificant influence	1	Significant influence	4
Moderate influence	2	Strong influence, throughout	5

- FPA can also be used with LOC to estimate system size [49], [70]. For example, for a particular language, the average number of LOC (AVC) to implement a FP can be found by using historical data. Then the size of the new system being developed in this language can be calculated as:

Size = AVC * number of FPs

On the other hand,

- FPA was designed to size business information systems, not to cope with complex mathematical algorithms [73]. Therefore, it is not suitable for real time, scientific and embedded systems [1], [29].
- There are difficulties when FPA concepts are applied to database oriented, transaction-processing systems [27].
- For large systems, application of FPA requires much time and effort [27].
- FPA underweights large and complex systems but have better results in small ones [73].
- Training and experience is needed for reliable FP counting [40], [49].
- Although Albrecht claims that FPA is technology independent [6], it is not [73]. All the counting rules are based on the documents of structured design techniques [31].
- The component classification of simple, average and complex is oversimplified [73].
- The choice of weights was determined by debate and trail [44] and technical complexity factor rating is a subjective process [29].
- There is a low interrater reliability in the counts [44], [70]. Different interpretations of the system by different analysts would result in different FP counts for the same system.

- There is a lack of intermethod reliability. Although there is a consensus on the basic components, there are many variations in the way of FP counting [44], [47], [49].
- The 14 technical complexity factors may not be necessary or satisfactory all the time and contributes little to the performance [39], [73].
- The counts of technical complexity factors do not satisfy the measurement theory because the factors are rated on an ordinal scale, but they are used as if they were on a ratio scale [29].
- It is possible to double count the internal complexity while in the UFP count and in the TCF count [73].
- Because of the subjectivity of the method, there is no fully automated FP counting tool [44].

With the increasing use of OO systems, the software practitioners have started to look for ways to apply FPA to size such systems. So far, many studies have been conducted in this context. Some practitioners, who believe that the functionality provided by FPA to both the OO and traditional procedural systems are the same, have found out some new measures similar to FPA such as [42] and Karner (See also Sections 2.2.9, 2.2.12, 2.2.13). The others have tried to map OO concepts into the FPA model such as [31]. However, the efficiency of such applications is still being discussed and more studies are needed in this field.

For example, in their study, T.Fetcke, A.Abran and T.Nguyen defined some mapping rules between the use case driven OO-Jacobson approach and Function Point model [31]. This mapping was aimed to estimate size and effort from the use case model. Since Function Point method is a widely known one, this mapping supplies an easily understandable and applicable estimation process. Also, measurement results can be compared between different development methods [31]. On the other hand, the most important drawback of the method is its limitation to the OO-Jacobson approach.

One way of defining the functional requirements of a system is in terms of use case models. Such modeling is especially suitable for OO systems. D.Longstreet [48] defines a natural relationship between FPA and use cases: “Function Pints is a method to size software from a requirements perspective and Use Cases is a method to develop requirements”. He defines many similarities between the concepts of FPA and use cases such as boundary and user (actor in use cases) definitions. He also explains how function points can be counted from use case descriptions. For Longstreet, such an application is very easy. Moreover, he says that this adaptation of use cases and FPA results in better use case descriptions, and estimates.

Today FPs are not just used for size measurement but also for many other purposes such as cost estimation [50], productivity measurement [10], [11], productivity analysis in software maintenance [33], quality evaluation, effort estimation [6], [29], and as a normalizing factor (e.g. defects per FP) [29].

Although Albrecht’s FPA with its many applications, is the most widely used metric in the industry, it has been loosing its popularity. Many variations of FPA have appeared to overcome its weaknesses against the concepts of new software technologies such as real-time and scientific software [75]. Original FPA method has also changed since its introduction. In 1984, a modification is made to the method. In the old version, for each component empirical weights were used whereas now each component is classified and assigned complexity rates depending on some rules. In the late eighties, the International Function Point Users Group (IFPUG) was founded and has produced the new versions of FPA. Since 1984, the basic FPA standards have not been changed and now the last version is known as IFPUG 4.1.

Today there are many alternatives of the FPA method. Some of the most popular ones are given in the following subsections.

2.2.4.1 Feature Points:

"Feature Points" method is an adaptation of FPA introduced by Software Productivity Research, Inc. (SPR) in 1996 [69].

Besides FPA's five component types, it has an additional sixth type called *algorithms* with a default weight of 3. Here an algorithm is defined as the set of rules, which must be completely expressed to solve a significant computational problem [69]. Another difference of Feature Points method and FPA is that the weights of external files are 7 instead of 10. Also in this method a single complexity weight is used for each component type (Table 2.5).

Table 2.5 Weight Factors for Feature Points

Component Description	Weight
Number of algorithms	3
Number of inputs	4
Number of outputs	5
Number of inquiries	4
Number of external files	7
Number of interfaces	7

Since it takes the algorithmic complexity into account in the calculations, the method can be applied to many kinds of systems such as real time systems, embedded systems, CAD, AI and even MIS [69].

When selecting between Feature Points and FPA, number and definition of algorithms in the system should be considered. If algorithms are countable and algorithmic factors are significant, Feature Points is a better solution, otherwise FPA should be selected [69].

Today Feature Points is one of the most tested and accepted alternatives of FPA [35]. There are many automated tools that support the method (e.g. Checkpoint and knowledgePLAR® from SPR). However, because of the difficulty of algorithmic calculations for large and highly complex projects, the method has been loosing its popularity [75].

2.2.4.2 Mark II Function Points:

"Mark II Function Points" method was developed by Charles Symons [73] to solve the problems of FPA especially about the calculation of internal processing complexity. The method is published in 1991. Now the design authority of the method is the Metrics Practices Committee (MPC) of the UK Software Metrics Association. Since its introduction, Mark II FP has been increasingly used in many places like UK, Hong Kong, India, Singapore and Canada [40], [55].

Mark II FP aims to measure the information processing. This method views the system as a set of logical transactions and calculates the Functional Size of software.

Mark II FP is independent of the project management and the development methods being used. On the other hand, it was deigned to measure the business information systems. Therefore, application of the method to other domains such as scientific and real-time software can be possible, but may require some modifications of the method [81].

The main differences between Mark II FP and the original FPA are:

- In Mark II FP, the concept of *entities* rather than *logical files* is used [73]. This reduces the subjectivity of measurements that result from using files [32], [72].
- Mark II FP is based on the effort required to produce the functionality, whereas FPA aims to find the value of functionality delivered to the end-user. Therefore, FPA is more difficult to verify or calibrate [73].
- Mark II FP is a continuous measure. On the other hand, FPA limits component size once a threshold is reached [81].
- For Mark II FP, calibration and recalibration is easy [73]. However, when there is little or no history, calibration to the new system may be difficult [35].
- In maintenance activities, FPA can show only the total size of the changed components. On the other hand, it is possible to calculate the size of each changed component in Mark II FP [73].
- Mark II FP requires more effort than FPA [73].
- Although it is said that Mark II FP is technology independent [81], it is also technology dependent like FPA when Technical Complexity Adjustment is added to the calculations [73].
- Both Mark II FP and FPA tends to give similar sizes up to software of 400 function points, but for larger software Mark II FP gives large size measures than FPA [73], [81].

For the Mark II FP calculations [81], first the viewpoint, purpose and count type of the software are determined. There are three viewpoints: *the Project Viewpoint, the Application Manager Viewpoint* and *the Business Enterprise Viewpoint*. Then the application boundary is drawn.

As a third step, logical transactions are identified. In the Counting Practices Manual [81], a logical transaction is defined as:

"Each logical transaction is triggered by a unique event of interest in the external world, or a request for information and, when wholly complete, leaves the application in a self consistent state in relation to the unique event."

Logical transactions consist of logical inputs, processes and outputs. In the Counting Practices Manual [81], functional sizes of these elements are defined as:

- The size of the input element is proportional to the number of uniquely processed Data Element Types (DTE's) composing the input side of the transaction.
- The size of the processing element is proportional to the number of uniquely processed Data Entity Types (entities) referenced during the course of the logical transaction.
- The size of the output element is proportional to the number of uniquely processed Data Element Types (DTE's) composing the output side of the transaction.

For each logical transaction, Input Data Element Types, Data Entity Types Referenced and Output Data Element Types are determined. Here entity type means something in the real world about which the user wants to hold information. One important point is that, only the primary entities not the non-primary ones should be taken into account.

Depending on these Input Data Element Types, Data Entity Types Referenced and Output Data Element Types, the *Functional Size* i.e. *Function Point Index* is calculated as:

$$FPI = W_I * \Sigma N_I + W_E * \Sigma N_E + W_O * \Sigma N_O$$

where

N_I = Input Data Element Types.

W_I = weight for Input Data Element Types.

N_E = Data Entity Types Referenced.

W_E = weight for Data Entity Types Referenced.

N_O = Output Data Element Types.

W_O = weight for Output Data Element Types.

and N_I, N_E and N_O are each summed over all Logical Transactions. Also the accepted industry average weights are 0.58, 1.66 and 0.26 for W_I, W_E and W_O respectively.

The Function Point Index defined above is based on the information processing size. As being optional, size of non-functional requirements i.e. technical complexity and certain quality requirements can be added to the above calculations. In fact, in the previous releases of the Counting Practices Manual, Technical Complexity Adjustment was a mandatory part of the method. However, because this part is not suitable for new software and to comply with the current International Standard on 'Functional Size Measurement', this part became an optional part.

For Mark II FP, there are additional six TCFs to the 14 general TCFs of the original FPA. These factors are:

- Interfaces with other applications.
- Special security features.
- Direct use by third parties.

- Documentation requirements.
- Special user training facilities.
- User defined characteristics.

In the sizing process, for each of the above factors a score of 0 to 5 is given. Then all this values are summed to get the Total Degrees of Influence (TDI). Technical Complexity Adjustment (TCA) is calculated as:

TCA = (TDI * C) + 0.65

where 0.005 is the current industry average for C.

Finally, the Adjusted Function Point Index (AFPI) is calculated:

AFPI = FPI * TCA

The productivity is defined as:

Productivity = Output / Input

where output is the Function Point Index (FPI) and input is the related effort i.e.:

Productivity = FPI / Effort

Here it is recommended that the unit of effort be defined in terms of work-hours. So, productivity is calculated in Function Points per work-hour.

It is also possible to covert Mark II FP to FPA and vice versa [74]. Both Mark II FP and FPA tends to give similar sizes for software of 200 to 400 FPA function points. For larger systems up to 1500 FPA function points, Symons gives the following conversion formula:

$M = 0.9 * I + 0.0005 * I^2$

where M = Mark II FP's and I = FPA UFP's.

For software above 1500 FPA function points, other formulas are suggested [74].

The application of MARK II FP in OO systems can be done by means use cases. For this purpose, P. G. Rule [61] recommends a model that combines Mark II FP and use cases. In this model, use cases are described as measurable logical transactions each having their input fields, response fields and referenced object classes. Then this information is used to calculate the functional size. Also Rule says that by this modeling the following problems related with use cases can be solved:

- Lack of rigor in the application of the technique. There are many interpretations of understanding use cases.
- Lack of a consistent level of granularity.
- UML practitioners freely apply the concept of abstract use cases i.e. 'uses/used by' and 'extends', that are more concerned with identifying opportunities for reuse than with analyzing the problem and describing the requirements.

Too early 'optimization' of the solution and a tendency to jump into 'the first solution that is thought of' rather than do a considered evaluation of a number of solution options.

In another work, Rule [60] gives some insight into the problems of use case description level and discussion of 'scenario' term. In spite of these difficulties, he says that logical transaction of Mark II FP is very similar to the detailed level use case concept. Therefore, they can be used together and this makes Mark II FP an easy and cheap method for OO software.

2.2.4.3 3-D Function Points:

"3-D Function Points" was introduced by S.A.Whitmire [80] of Boeing in 1991. The method is similar to Albrecht's FPA, however it has also *control*

components in addition to the *functional* and *data* components [32], [75]. The data components are calculated like in FPA. For the functional components, number and complexity of functions and the set of semantic statements, and for the control components, system states and transitions are taken into account [32]. By this way, the method brings two new concepts to FPA: transformations and transitions.

For the application of 3-D FP to OO software Table 2.6 is used [18].

Table 2.6 Weights for 3-D FP

Component Types	**Low Complexity**	**Average Complexity**	**High Complexity**	**Total**
Internal Data	………* 7	………* 10	………* 15	………
External Data	………* 5	………* 7	………* 10	………
Inputs	………* 3	………* 4	………* 6	………
Outputs	………* 4	………* .5	………-* 7	………
Inquiries	………* 3	……….* 4	………* 6	………
Transformations	………* 7	………* 10	………* 15	………
Transactions	N/A	………* 3	N/A	………
Total 3-D FP Value	………	………	………	………

Also for the reuse concept of OO software, percentage of 3-D FPs for each imported class can be used.

3-D FP is a technology independent method especially suitable for real time and scientific systems. However, 3-D FP counting is difficult in the early phases of a project since it requires a detailed system knowledge [32]. Also its application to OO software requires well documentation of imported software [18]. Since the method has not been published outside Boeing, too little is known about its validity and success.

2.2.4.4 FP by Matson, Barret and Mellichamp:

The FP method suggested by Matson, Barret and Mellichamp [50] is an alteration of Albrecht's FPA.

In this method, system components are classified as inputs, outputs, master files and inquiries where interfaces are not counted separately, however, as part of master files. Unlike FPA, only one complexity level is used and the adjustment factors have a range of +-25% rather than +-35%.

By taking the coefficients as the average complexity values, the FP for the kth observation is calculated as:

$$FP_k = (4IN + 5OUT + 4INQ + 10FILE) * c_k$$

where

IN = inputs

OUT = outputs

INQ = inquiries

FILE = master files

c_k = adjustment factor

From this information, effort can be calculated by the formula:

$$E = \beta_0 + \beta_1 * (c_k * IN)^2 + \beta_2 * (c_k * OUT)^2 + \beta_3 * (c_k * INQ)^2 + \beta_4 * (c_k * FILE)^2 + \varepsilon$$

Although the method is based on a small set of data, it gives good results and can be taken as a sample for future studies.

2.2.4.5 Full Function Points:

In a 1997 research project by the University of Quebec in cooperation with the Software Engineering Laboratory in Applied Metrics (SELAM), "Full Function Points (FFP)" method was developed [51].

The aim of FFP is to solve the problems associated with FPA when applied to real-time and embedded systems. The method is an extension of FPA, but it also takes into account the specific transactional and data characteristics [i.e. counting sub processes and single-occurrence data) of real-time systems [51], [54]. Therefore, FFP can be applicable to a large set of systems from real-time to MIS [56].

FFP takes FPA method as a base where it uses the five components of FPA (i.e. EI, EO, EQ, ILF and EIF) to measure the management function types and defines additional six components to measure control function types specific to real-time systems [1], [51]. These new components are classified as [56]:

- Data Function Types:

  Update Control Group (UCG): similar to ILF.

  Read-only control Group (RCG): similar to EIF.

- Transactional Function Types:

  External Control Entry (ECE): similar to a simpler subset of EI.

  External Control Exit (ECX): similar to a simpler subset of EO/EQ.

  Internal Control Read (ICR): similar to a simpler subset of EI/EO/EQ.

  Internal Control Write (ICW): similar to a simpler subset of EI.

Also there are many other FFP concepts similar to FPA's such as definition of boundaries and users. But unlike FPA, FFP excludes technical and implementation considerations in its calculations [56].

In the FFP measurement process [51], [54], first the management function types are identified and counted depending on the FPA rules. Then to measure the control function types, the sub processes and their function types are determined and assigned points according to the FFP counting rules. The unadjusted count can be expresses as follows [1]:

FFP = Management FP + Control FP

= (FPA Count – Control information) + Control FP

For FFP, so far many field tests have been conducted. The results from different organizations are:

- FFP measures real-time systems more adequately than FPA [14], [51].
- The FFP concepts and rules are defined clearly so it is easy to understand and there is no need for a FFP specialist [1]. Even inexperienced teams can make estimations as successfully as the experienced ones [14].
- There are more function types that need to be counted in FFP than FPA, however because of the method's simplicity, the required effort is similar to FPA [1], [51].
- The weights are empirically determined ones in FFP and they are chosen in a way to supply the compatibility with FPA method. Therefore some calibration may be needed for future applications [51].
- The difference between FFP and FPA results depends on the number of sub processes in each process of the system. If there are only a few sub processes, then FFP and FPA give close size results [54].

- The quality of functional requirements affects the FFP results. In fact, if these requirements are well documented, FFP can give better size estimations in the early phases of a project [1].
- For various design methods, the needed effort can be determined with FFP [14].

Although the method has been applied successfully in many organizations, since 1999, COSMIC FFP has taken the place of FFP with many new improvements.

2.2.4.6 COSMIC FFP:

The Common Software Measurement International Consortium (COSMIC) developed a new functional size measure called *"COSMIC FFP"* in November 1999 [3].

COSMIC FFP is a model that provides reliable size measures for real-time, MIS and hybrid (containing both real-time and MIS properties) systems. However, the model is not suitable for complex mathematical algorithms and process continuous variables [23].

COSMIC FFP takes User Functional Requirements as a base and does not attempt to take into account the Technical and Quality Requirements [24], [75].

To size the system functionality, first the Functional User Requirements are mapped to a COSMIC FFP FUR (Functional User Requirements) model [23]. This FUR model consists of two layers: Transaction Types (or Functional Processes) and Data Movement Types (or Functional Sub Processes). There are four Data Movement Types: *entry, exit, read* and *write*.

The mapping process compose of the following steps:

- Identifying software layer and/or peer items.

- Identifying software boundaries.
- Identifying functional processes.
- Identifying data groups.
- Identifying data attributes.

When the mapping process is finished, the measurement process begins with identifying each sub processes and their types. Then for each of the sub processes, a numerical size of 1 Cfsu (Cosmic Functional Size Unit) is given. Here 1 Cfsu is equal to 1 Data Movement Type. Finally for each identified layer of sub processes, size is calculated as:

$$\text{Size}_{\text{Cfsu}}(\text{layer}_i) = \Sigma\text{size}(\text{entries}_i) + \Sigma\text{size}(\text{exits}_i) + \Sigma\text{size}(\text{reads}_i) + \Sigma\text{size}(\text{writes}_i)$$

For each identified layer of changes to the requirements is calculates as:

$$\text{Size}_{\text{Cfsu}}(\text{change}(\text{layer}_i)) = \Sigma\text{size}(\text{added sub processes}) + \Sigma\text{size}(\text{modifies sub processes}) + \Sigma\text{size}(\text{deleted sub processes})$$

These calculations can also be done by using the metric defined in [23].

So, by adding sub processes, it is possible to measure the size of any higher-level processes [3]. In fact, COSMIC FFP provides a size measurement in any layer or peer-item [2], [3], [24].

The most important pluses of the model are its theoretical base [3] and compliance with ISO 14143-1 standard [23].

Many field tests were done on COSMIC FFP and the following results are obtained [3]:

- Experience on COSMIC FFP and the domain are necessary to get better results.
- Specifying the measurement procedures of COSMIC FFP is necessary to meet the organizational needs.
- It is an easily applicable model.
- The effort needed for the model is similar to the other FP methods.

During the field tests, also the relation between size and effort was found for different phases of a project (i.e. specify, build and test phases).

Specify effort = $4.0342 * (\text{size})^{0.9903}$

Build effort = $12.313 * (\text{size})^{1.015}$

Test effort = $5.2124 * (\text{size})^{1.024}$

However, the formulas are based on small and non-homogeneous data from various types of organizations. So, calibration may be needed when applied to other organizations.

2.2.5 Statistical Object Model

One study done to measure software size for OO systems is Laranjeira's *"Statistical Object Model (SOM)"* [46]. The model attempts to solve the following software sizing problems:

- The need to have specific knowledge of a system in its early stages,
- The need to be able to relate, as accurately as possible, this knowledge to the physical size of the program.
- The need to find a way to cope with the limited information about a system in the early stages of its development.

SOM is a statistical approach to estimate the size of software within a specified confidence interval. Its logic comes from Boehm’s previous cost estimation studies. Statistical object model is based on graphs called “learning curves” on which the estimations converge to the actual size with the increasing details of object decomposition as illustrated in Figure 2.1.

Here, since functional specifications are represented by objects, the model is especially suitable for OO systems.

SOM is a model, which provides the estimators more accurate size estimates by using statistical theory. Nonfunctional requirements and low biasing are taken into consideration in the model. Also, results of SOM can be used as an input to the available cost estimation models such as COCOMO. On the other hand, it is a subjective model because; B is not a calculated value. It depends on the organizational characteristics and experiences. However, some statistical techniques can be used to increase the accuracy of B. In addition, SOM has some mathematical errors related to statistics, exponential functions, and the nature of discrete vs. continuous data [36].

In spite of its problems, SOM is a promising method for OO size estimations.

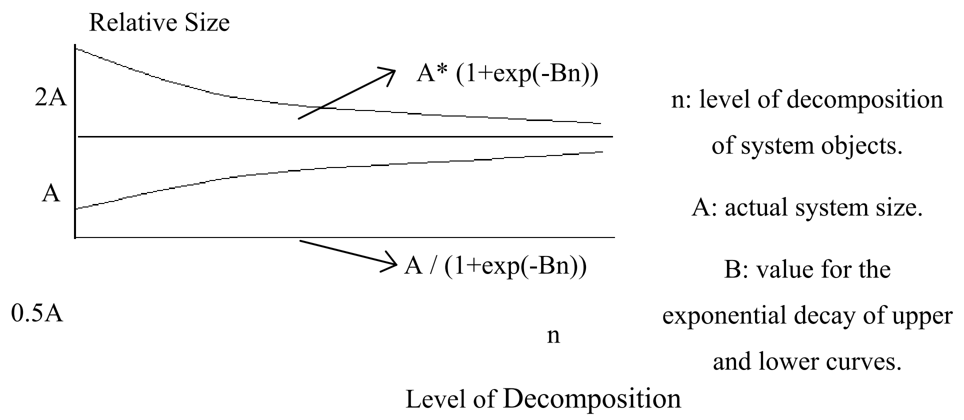


Figure 2.1 Size Estimation Accuracy by Using Object Decomposition

2.2.6 Object Oriented Project Size Estimation

Traditional metrics are not adequate to measure the size of OO systems, because these systems have a different structure from the procedural programming. So, different techniques are needed. *"Object Oriented Project Size Estimation (OOPS)"*[16] is a technique that is suitable for measurement of such OO systems.

OOPS is a statistical technique based on the study of Dr. S. Moser and Dr. O. Nierstasz [16]. Here, the main idea is to get the object's *"point value"* depending on the object components. Then, this value is used to determine the days required to develop that object. The following steps are used in the calculations [16]:

1. The objects are defined in a class model including the objects' names, number of each object's attributes, methods and parameters to those methods.

2. For each object, first the point value is set to 0. Then, each unique token in the object's name, attributes, methods and parameters to the methods are counted and added to this point value. The repeated tokens are not counted.

3. The final point value is used to determine the days required to develop the object. The formula is:

Days Required to Develop= (B1*Points) + (B2*Points^2)

where B1 is 0.367 and B2 is 0.0000696.

The above values of B1 and B2 were obtained from the data of 36 OO systems. So, they reflect industry averages. To increase the accuracy of the results, the organization using OOPS should calibrate these parameter values with the data from its past projects and by using some suggested equations [16]. Also, if some organizational factors are changed, B1 and B2 should be recalibrated. The accuracy of these new parameter values can be statistically checked by using RSQUARED technique.

Today OOPS is being used in two organization in Colorado. It has also an automated tool to calculate the equations. However, little data is available about this new technique's usefulness and accuracy.

2.2.7 Distance-Based Approach

"Distance-Based Approach" is a mathematical method in which definition of distance is used to measure the size of OO specifications [57]. The method defines size of an object as the distance between this object and a reference object. Size is defined as:

V x Є X_t: s(x) is the distance from x to r_{st}

where t is the measurement object, s is the size sub-attribute; r_{st} is the reference object of type t with the lowest value of s and X_t is the set of measurement objects of type t.

Distance-Based Approach is a formal method in the sense that it satisfies the necessary size measurement axioms [57]. Since binary concatenation operations are not needed to justify the axioms, this method is especially suitable for OO specifications and can even be used for object-based ones. The method is also flexible because; different reference objects can be chosen for different measurement objects and by choosing suitable reference objects, the method can be applied to any size sub-attribute (i.e. length, functionality, complexity and reuse). Moreover, calculations can be transferred to a metric space (X_t,δ_t) as:

$V\ x \in X_t$: $\delta_t(x_t,r_{st})$ measures $s(x)$

However, Distance-Based Approach is subjective in the choice of reference values. And like many other sizing methods, more research is needed in this area to prove its success.

2.2.8 Vector-Based Approach

The *"Vector-Based Approach"* introduced by Hastings and Sajeev [35], is based on two concepts: *"Vector Size Measure (VSM)"* to size the system and *"Vector Prediction Model (VPM)"* to estimate the corresponding effort.

The approach attempts to measure the system size from the algebraic specifications described in the Algebraic Specification Language (ASL). The algebraic specifications are based on abstract data types (ADTs) and ASL provides a mathematical description of the system. The approach accepts Fenton's multidimensional definition of size. Also similar to Halstead's method, the ADT properties are defined in terms of operators and operands.

To measure VSM, first system functionality and complexity is calculated. Then, system length is derived from these values.

For an ADT, A, the functionality is calculated as:

$f_a = \Sigma\ OP_f$

where OPs are the operators and operands in the syntactic section of the ADT, A.

Similarly, the complexity of an ADT, A, is calculated as:

$c_a = \Sigma\ OP_c$

where OPs are the operators and operands in the semantic section of the ADT, A.

Finally, the length is derived from these values as:

$I_A = \Sigma\ OP_A = \Sigma\ OP_f + \Sigma\ OP_c = f_a + c_a$

The total system size SS with N ADT is calculated as:

$S_S = \Sigma\ S_N$

where $S_A = (f_a, ca)$ is the size of an ADT, A.

The size can also be represented mathematically by a vector of functionality and complexity. The magnitude and direction of this vector is defined as below:

magnitude (m) $=\sqrt{} (f^2 + c^2)$

direction theta = $\tan^{-1}$ (c/f)

The ratio between complexity and functionality is:

gradient g = c/f

In the above calculations, size and gradient are in the absolute scale. Functionality, complexity and magnitude are in a discrete ratio scale. However, for magnitude, the scale is mapped to real numbers and for size, it is also possible to define it as scalar attributes and a vector combination.

After measuring the size, VPM can be used to predict the corresponding effort. VPM takes magnitude and gradient as inputs, uses a regression model and establishes a relation between effort and these inputs [35].

This approach is similar in concept to the *"Distance-Based Approach"* (see section 2.5). Since the OO specifications can be easily transformed to algebraic ones, this approach is very suitable for OO systems.

There are some pluses of the approach. First, the methods used (VSM and VPM) are theoretically and empirically validated ones based on mathematical foundations. Second, the specifications written in ASL provide an early estimation by VSM and VPM. Also, ASL specifications make it possible to automate VSM calculations. On the other hand, the approach bases solely on ASL. This means additional work to convert the existing specifications into ASL and a need for expertise on this modeling language. Another problem with ASL is that not all problem domains can be completely represented by this language, and this may result in inaccurate estimations. Another limitation of the approach is its fixed definition of the relationship between length, functionality and complexity. This makes the approach inflexible and even such a size relationship may not be required or needed in general [57].

2.2.9 Object Points

"Object Points" is a size measurement method suitable for software systems building from objects and modules. This method is developed for ICASE environments with object-based repositories [42].

Since object points can be easily derived from initial specifications, the method is being used for the early effort and cost estimations in COCOMO 2.0 [29].

The idea behind object points is very similar to FPA. However, it differs from FPA in that it also takes into account reuse to find the total object points. To calculate object points, first the driver objects of the system i.e. screens, reports and components are identified. Secondly, based on their complexity levels, each object is assigned a value as simple, medium or difficult. Table 2.7 illustrates these values for COCOMO 2.0. Then, depending on the object types and complexity levels, objects are weighted (Table 2.8). Finally, these weights are summed and multiplied by reuse amount to get the total object points [42], [72], [29,]:

Total Object Points = Object Points * (100 – reuse) /100

Object Points method is an easily understandable one that can be obtained accurately in the early phases of a project to be used as a size input to cost and effort estimations. Also, automation is another advantage of the method. However, its usage is restricted to object-based ICASE environments [2], [42]. In addition, for each different ICASE environment, the components of the metric need to be customized [42]. Another important disadvantage of the method is that it cannot be directly comparable with FP [31].

Table 2.7 Object Point Complexity Values in COCOMO 2.0

For Screens			
	Number and Source of Data Tables		
Number of Views Contained	Total <4 (<2 Server, <2 Client)	Total <8 (2-3 Server, 3-5 Client)	Total 8+ (>3 Server, >5 Client)
<3	Simple	Simple	Medium
3-7	Simple	Medium	Difficult
8+	Medium	Difficult	Difficult
For Reports			
	Number and Source of Data Tables		
Number of Sections Contained	Total <4 (<2 Server, <2 Client)	Total <8 (2-3 Server, 3-5 Client)	Total 8+ (>3 Server, >5 Client)
0 or 1	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
4+	Medium	Difficult	Difficult

Table 2.8 Object Point Weights in COCOMO 2.0

Object Type	**Simple**	**Medium**	**Difficult**
Screen	1	2	3
Report	2	5	8
3GL Component	-	-	10

2.2.10 Predictive Object Points

Some software practitioners who think that the idea behind traditional size metrics are not suitable for OO systems, have found out a new set of metrics:

"Predictive Object Points (POPs)" [77]. In fact, these are a collection of existing OO metrics in the literature. The idea behind POPs is similar to Chidamber and Kemerer's metrics for OO [19], [20].

POPs metrics are based on the three dimensions of OO size i.e. functionality, complexity and reuse. The main part of POPs is the *weighted methods per class (WMC)*. It is the average number of methods per class and used for calculating functionality and complexity. The other POPs metrics are:

- *Number of top-level classes (TLC)*: Number of root classes in the class diagram. It is used for calculating the amount of reuse.
- *Average depth of inheritance tree (DIT)*: Number of levels from the root to a class. It is used for calculating the amount of reuse.
- *Average number of children per base class (NOC)*: Number of descendents of a class. It is used for calculating the amount of reuse.

In POPs, first weights are assigned to the methods of each top-level class and WMC is calculated. Table 2.9 and Table 2.10 are used in these calculations. Then the results are combined with TLC, NOC and DIT values. The below formula is used to find the final POPs value:

$$\text{POPs (WMC, NOC, DIT, TLC)} = (\text{WMC} * f_1(\text{TLC, NOC, DIT})) / 7.8 * f_2(\text{NOC, DIT})$$

where

f_1 = overall system size.

f_2 = effect of reuse.

The values of these parameters are not provided by the author in the article. However, they are embedded in the Price Systems tool.

It may be difficult to find some of the information for these calculations in the early phases of a project. However, there are some techniques that use the available project information to make the above calculations easier in the early phases [77]. Also, using use cases may be another solution for this problem.

Although POPs seem to give promising results, it is based on a small set of data. Therefore, data collection on different projects should continue, more study on using POPs with use cases should be done and automation procedures should be found to increase the usefulness of these metrics.

Table 2.9 Method Weightings by Type and Complexity

Method Type	Method Complexity	Weight
Destructors/Constructers	Low	1
	Average	4
	High	7
Selectors	Low	1
	Average	5
	High	10
Modifiers	Low	12
	Average	16
	High	20
Iterators	Low	3
	Average	9
	High	15

Table 2.10 Complexity Assignments

	Number of Properties		
Message Responses	**0->1**	**2->6**	**7 or more**
0->1	Low	Low	Avg
2->3	Avg	Avg	Avg
4 or more	Avg	High	High

2.2.11 M.Shepperd and M.Cartwright Size Prediction System

In one of their studies to determine the effectiveness and usability of Chidamber and Kemerer metrics for OO systems, M.Shepperd and M.Cartwright [66] found that some of these metrics were difficult to collect especially in the early phases of a project. By using the data from a large real time C++ system, they found that STATES (i.e. count of states per class in the state model) could be a good predictor of size (LOC) [66].

The size equation that was obtained by linear regression is:

Size (LOC) = 1101.01 + 170.68 * (STATES)

This is a simple and easily applicable prediction system. In contrast to Chidamber and Kemerer metrics, STATES can be easily counted in the early analysis and design phases. Also, CASE tools can be used to automate the STATES counts. However, this study is based on the local data of only one project of an organization. Therefore, this prediction system may not be directly applicable to other systems. On the other hand, the ideas mentioned here could be a model for others to find such simple and usable size equations.

2.2.12 Use Case Points Method

The approach of modeling the functional requirements of a system based on use cases has a great interest in the OO software engineering community. Most studies have shown that such requirement specifications can be used successfully in size and effort estimations [7], [9].

Today there are many different methods available that base on use cases to make estimations of size, effort and productivity. One such a method is the *"Use Case Points"* method. This method was developed by Gustav Karner as a diploma thesis at the University of Linköping in 1993. Now it is the copyright of Rational Software. The idea behind use case points method is similar to the FPA method [7]:

1. The actors of the use case model are categorized depending on their properties and assigned weights (Table 2.11).

Table 2.11 Actor Categories and Corresponding Weight Factors

Actor Categories	Properties	Weight Factors
Simple	Another system with a defined API	1
Average	Another system interacting through a protocol such as TCP/IP	2
Complex	Such a person indicating through a graphical user interface or a web page.	3

The number of actors in each category is counted. Each of these counts is multiplied with the corresponding weight factors, and then summed to get the *unadjusted actor weight (UAW).*

2. Depending on the number of transactions included, the use cases are categorized and assigned weights (Table 2.12). “A transaction is a set of activities, which is either performed entirely, or not at all”. Here included and extending use cases are omitted. To make these calculations accurately and easily, the use cases of the system should be defined correctly with a suitable degree of detail.

Table 2.12 Use Case Categories and Corresponding Weight Factors

Use Case Categories	Number of Transactions Included	Weight Factors
Simple	>=3	5
Average	4-7	10
Complex	<7	15

The number of use cases in each category is counted. Each of these counts is multiplied with the corresponding weight factors, and then summed to get the *unadjusted use case weights (UUCW).* From UAW and UUCW, the *unadjusted use case points (UUPC)* is obtained:

UUPC = UAW + UUCW

3. By using technical complexity factors (Table 2.13) and environmental factors (Table 2.14), use case points are adjusted.

Table 2.13 Technical Complexity Factors for Use Case Points Method

Description	Weight
Distributed system	2
Response or throughput performance objectives	2
End-user efficiency	1
Complex internal processing	1
Reusable code	1
Easy to install	0.5
Easy to use	0.5
Portable	2
Easy to change	1
Concurrent	1
Includes security features	1
Provides access for third parties	1
Special user training facilities are required	1

Table 2.14 Environmental Factors for Use Case Point Method

Description	Weight
Familiar with Rational Unified Process	1.5
Application experience	0.5
OO experience	1
Lead analyst capability	0.5
Motivation	1
Stable requirements	2
Part-time workers	-1
Difficult programming language	-1

First, a value between 0 and 5 is assigned to each factor in Table 2.13. These values are determined depending on the rate of influence of each factor to the system. Then, each of these values is multiplied with the corresponding weight factors, and then summed to get the *Tfactor.* The below formula is use to calculate technical complexity factors:

TCF = 0.6 + (0.01*Tfactor)

The same process is applied to Table 2.14 to get the *Efactor* and by the below formula environmental factors are calculated:

EF = 1.4 + (-0.03*Efactor)

Finally, the *adjusted use case points (UCP)* are calculated by the following formula:

UCP = UUCP*TCF*EF

4. A previously determined amount of man-hours per use case point is used to find the total effort of the project. However, this determined amount seems to differ from author to author [7]. For example, for Karner, this is 20 man-hours per use case point. On the other hand, Schneider and Winters take into account the environmental factors when determining the number of man- hours per use case point [62]. The number of factors F1 through F6 that are below 3 are counted and added to the number of factors in F7 through F8 that are above 3. If the total is 2 or less, 20 man-hours per UCP; if the total is 3 or 4, 28 man-hours per UCP is recommended. If this number is large than 4, changes in the project is recommended to adjust the number. Increasing the number of man-hours to 36 per use case point is also possible. Therefore, calibration of this value to the organization may be needed.

There are some applications of use case points method to the industry. One such study was done by B.Anda, H.Dreiem, D.I.K.Sjoberg and M.Jorgensen [7]. They applied the method to three projects of a software development company. Based on their case studies, they got the following results:

- The estimates can be affected by the structure of the use case model being used. Even for the same model, different estimators can interpret the actor and use cases differently.
- In the assignment of technical and environmental factors, there is some subjectivity. However, this subjectivity can be reduced by having more experience with past projects and calibration of the method to the organization.
- Use case points method should be used not in place of but with expert estimations. So, misjudgments can be reduced.
- Time sheets of the organization and use cases should be designed in a consistent manner to get better feedback.
- When compared with FP method, use case point method is based on structured models. So, it needs less effort and automation is easier. On the other hand, unlike FP, use case points method has no internationally accepted standards, which may result in differences in counts by different estimators.
- Other project activities such as training should be added in some way to the use case estimates to get a complete project estimate

In another study conducted by M.Arnold and P.Pedross [9], use case points method was applied to a major Swiss Banking Institute that uses OO methods in their software development activities. Although the ideas behind Arnold and Pedross' method are similar to Karner's, it is a new method. In this method, first system functionality is measured based on use cases and scenarios. Then, eight technical factors are used for calibration. The results of this study are:

- Use cases and scenarios can be reliably used in size estimations.
- Technical factors are easy to calculate. However, requirements modeled differ in their degree of details and this can affect the use case counts and as a result the final system size.
- Free textual use case descriptions are insufficient to measure size.
- There is a lack of defined abstraction mechanism for use cases.
- The graphical notations for use cases and scenarios are insufficient.
- Simple tools can be used to simplify the calculations.

Kirsten Ribu, in her Master of Science thesis at the University of Oslo [59], conducted two case studies in a major software company and also examined some student projects. She concluded that:

- The use case points method can be applied to many different kind of software.
- To get accurate results, the use cases should be written in a suitable level of detail. However, most projects lack standardized use case descriptions.
- The use case points method can be easily learned and applied in a short time.
- The main idea of the method is based on FPA and Mark II FP, so it can be easily accepted by the companies.
- Dropping the technical complexity factors can give better results.

- There are some ways of converting the use case points to man-hours. However, the available ones sometimes give unreliable results. So, more study is needed in this subject.
- More applications of the method, especially to the large, real-time and complex algorithmic projects are needed.

In her thesis, to solve some of the mentioned problems, Ribu also makes some suggestions. For the use case standardization problem, she gives guidelines to write use cases and if use case descriptions are not at a suitable level of detail, suggests some alternative ways. Moreover, she purposed an extension of the use case points method to solve the technical complexity factor problem.

From all the above results, it can be seen that more research is needed to increase the accuracy of the method. Especially the modeling processes should be improved and standardized to get the correct level of detailing in use case definitions and so to reduce the inconsistencies in size estimations.

2.2.13 J.Kammelar's Sizing Approach

This sizing approach [41] applies the idea behind FPA to the OO concepts with new counting rules rather then mapping the OO concepts to FPA.

Similar to FPA's *function points*, here the functional size is defined in terms of *component object points (COPs).* In the counting process, first the counting elements are determined. There are two kinds of counting elements:

- *User Domain Elements (FUR's):* include the use cases and business objects.
- *System Domain Elements (BFC'S):* include services, classes, operations and transformations.

Then three different estimations are conducted. These are *domain model count, analysis count* and *design count*. Analysis and design counting is a nine step process:

1. Count types are determined i.e. one of the domain model count, analysis count or design count is selected.

2. The counting boundary and granularity are determined. The actors of the system are defined.

3. Specifications are reviewed to see whether they fulfill the minimum requirements for this counting technique. Also inconsistencies if exist, are determined.

4. All use case services are identified and valuated. For the analysis and design counts, different steps are applied to find the *service functionality*:

Analysis Count Rules:

- For each use case, related services are determined. Here, service definition is equivalent of the FP transactions and have to comply with the elementary process definition.
- Per service per use case is counted as 2 points.

Design Count Rules:

- For each use case, related services are determined. Here, service definition is accepted similar to the elementary process definition.
- The services per use case are counted based on the Service Valuation Matrix defined in Table 2.15.

Table 2.15 Service Valuation Matrix

Number of Operations / Transformations	1	2 - 3	> 3
COP’s	1	2	4

5. Use case service/class relations are identified and valuated. This step is only applicable to the analysis count:

Counting Service/ Class Relations:

- For each use case, the relations between the services and all classes that collaborate to provide (parts of) those services are found.
- For every unique service / class relation, 3 points is counted and accumulated to the appropriate class.

Counting Transformations:

- For each use case, transformations of the services are determined and 5 points are given for each one.
- Transformation functionality is accumulated separately from the service functionality.

6. Classes and structures of the domain model are identified and valuated. In this step, the class attributes and objects’ structures are counted.

Counting Class Attributes:

- For each class, the number of attributes are counted and assigned points using Table 2.16. Here, inherited attribute are not taken into account.

Table 2.16 Class Attribute Valuation Matrix

Attribute Part	**Number of Attributes**	< 3	3 – 6	> 6
	COP's	2	5	7

Counting Object Structures:

Objects structures, i.e. generalization/specialization and aggregation/composition structures are counted. Depending on Table 2.17 and 2.18, the below counting rules are applied.

Table 2.17 Structure Determination

Q1	Is the class is a generalization?	Y	Go to A1 in Table 2.18
		N	
Q2	Is the class an elementary specialization?	Y	Apply counting rule for gen./spec. structure.
		N	
Q3	Is the class an aggregation/composition?	Y	Apply counting rule for aggr./comp. structure.
		N	

Table 2.18 Generalization Counting

A1	Has the structure from which the actual class is part of already been counted?	Y	No action.
		N	Apply couting rule gen./spec. Return to Q3 in Table 2.17

- For the generalization/specialization structures, all structure levels, including the highest super-class are counted and 3 points are given for each structure level.
- For the aggregation/composition structures, all component sub-classes of the structure are counted 2 points are given for each sub-class.

Finally, Object Structure Valuation Matrix defined in Table 2.19 is used.

Table 2.19 Object Structure Valuation Matrix

Structure Part	**AssociationType**	**Gen./Spec.**	**Aggr. /Comp.**
	COP's	Number of Levels*3	Number of Sub-classes*2

Total Class Valuation = Total $\text{COPs}_{\text{attribute Part}}$ + Total $\text{COPs}_{\text{Structure Part}}$

7. For the design count, operations and transformations are identified and valuated.

Counting Operations:

- For each service, the relation between the service and the first responsible class is determined.
- The Operation/Transformation Valuation Matrix defined in Table 2.20 is applied to the operation.
- Number of points is added to the class.

Counting Transformations:

- For each use case, transformations are determined.
- The Operation/Transformation Valuation Matrix defined in Table 2.20 is used to find the points of the related transformations.

Table 2.20 Operation/Transformation Valuation Matrix

Collaborating Classes / **Operation Type**	N	Y
Query	2	3
Modify	3	4
Transformation	4	5

8. For the design count, reusable elements are determined. However, this step is still being developed.

9. The service functionality is added to the class functionality to determine the total size.

Kammelar's size measure is very suitable for OO or object-based components and does not face with the problems when FPA is applied to such systems. It also takes into account reusability and takes use cases as a base in its calculations. Especially the analysis count gives very accurate and promising results. However, for each count type a minimal set of specifications is required [41]. In addition, like FPA, the weights being used in calculations were determined by trial. In spite of its limitations, this approach can be a base for component-based estimations.

CHAPTER 3

CASE STUDY

A case study on the related subject is presented in this chapter. Five of the size metrics and methods defined in Chapter 2 are selected and applied to an OO project whose requirements are defined in a use case model.

3.1 Definition of Work

The case study is composed of 3 headings. These are:

1. Data Collection: To better understand the application of metrics and methods and comparison results, some characteristics of the related project is given.

2. Application of Metrics and Methods: From the size metrics and methods mentioned in Chapter 2, five of them are chosen to estimate the project size. These are "Line of Codes", "Mark II Function Points", "Object-Oriented Project Size Estimation", "Use Case Points Method", and "J.Kammelar's Sizing Approach". Then size and effort estimations are made for each of these metrics and methods.

3. Results Evaluation: The results of size and effort estimations for the selected metrics and methods are compared with the actual results.

3.2 Data Collection

The case study reported in this thesis was carried out in a software development company located in Ankara. To better compare the study results, one of the completed projects of this company was chosen for the size estimations. This was an industrial project on civil engineering. The aim of the project was to automate the business functions (Hakediş Applications and Reporting) of the company.

The project data was collected from a brief software requirements specification document and a use case model defined in Rational Rose. The model consisted of use case diagrams with brief textual descriptions for each use case, sequence diagrams, class diagrams and logical and component view definitions. Also, general project information such as actual project size, effort, and software architecture were collected by e-mail communication with one of the project members.

It was the second version (V.2) of the project used in this thesis. The actual size and effort values of the first version (V.1) were given as 21.790 KLOC and 239 man-days respectively by the project team. Accepting a month as 20 days:

Actual Effort (V.1) = 239 / 20 = 11.9 man-months

The second version (V.2) contains some additions and changes to the first one and its actual effort value was given as 630 man-hours. Accepting a working day as 8 hours, a month as 20 days and with 3 developers, the actual effort for V.2 is:

Actual Effort (V.2) = 630 / (8 * 3) = 26.2 man-days

= 26.2 / 20 = 1.3 man-months

From the above information, actual project effort can be calculated as:

Actual Project Effort = Actual Effort (V.1) + Actual Effort (V.2)

= 11.9 + 1.3 = 13.2 man-months

Here, because only small changes were made in V.2, the effort of this version was accepted as only the effort for new additions. The effort for changes was not taken into account.

On the other hand, there was no size info for V.2. However, we know that only small changes and additions were made in this version. This means a small increase in V.1 size when calculated in terms of SLOC. And assuming this increase would cause a slight change in the actual project effort, the actual project size was accepted as the size of V.1 in this thesis.

For this project, 48 use cases and 48 classes were defined in Rational Rose. However, only 27 sequence diagrams were available. There were some problems about the project information. The use case definitions were too much detailed. On the other hand, the level of details for the sequence diagrams were lacking. There were three entity classes referenced in the sequence diagrams, but these classes were not defined in the class model. Also there were control classes being used in the sequence diagrams with no related entity classes. Moreover, some input/output information had either general definitions or no definitions.

The general characteristics of the project are given in Table 3.1[i]. Since this is an industrial project, the detailed use case and class definitions cannot be given in this thesis. This information is kept private and only the results are presented and evaluated. For each use case and class, instead of their names, ids were used to

[i] The structure of Table 3.1 is taken from [57].

distinguish them. However, to give an idea about the project, a sample class diagram, an object definition and a use case diagram are included in Appendix C and Appendix D. Moreover, some of use case and class properties can be obtained from Table E.1, Table F.1, Table G.1, Table H.1, Table H.2, Table H.3, Table H.4 and Table H.5 in Appendices.

Table 3.1 General Characteristics of the Project

General Characteristics	Project Info
Size/Effort	≈ 21.790 KLOC and 13,2 man-months
Software Architecture	Following another finished project, so known architecture
Programming Environment	Visual Age for Java
Project Members	3 developers with 2-3 years programming experience and 1-2 years Java experience and 1 reviewer.
Application Domain	Civil Engineering (Hakediş Applications and Reporting)
Number of Use Cases	48
Number of Classes	48

3.3 Application of Metrics and Methods

Knowing that for each method the best results can be obtained only when the requirements of these methods are hold, and finding that the available use case modeling and the project data could better meet the requirements of five of the methods ("LOC", "OOPS", "Use Case Points Method", "J.Kammelar's Sizing Approach" and "Mark II FP") defined in the previous chapter, these five methods were chosen and applied in this case study. Also in this method selection, both the traditional and the new OO methods were chosen to compare these different approaches (See also CHAPTER 1).

Sample use case diagrams, class diagrams and object definitions used in the below estimations can be found in Appendix C and Appendix D.

3.3.1 Lines of Code

Here, J.Smith's method [67] whose steps are defined in Section 2.2.2 was used to find the system size in terms of LOC:

1. The structural hierarchy consists of one subsystem and its classes.

2. For Level 1, there are 48 use cases.

3. Total Size = [(48/10)*7+(0/10)*56+(0/10)*448+(0/10)*3584] KSLOC

$= 33.6$ KSLOC = 33600 SLOC

4. For Level 1, EN1= $(0.1*48+0.8*0+6.4*0+51.2*0)^{0.11} = 1.19$

Since we have a simple business system consisting of only Level 1 use cases with an effort multiplier of 1.19, the effort per use case for this level is 1.19*55= 66 hrs/use case.

5. For Level 1, the total effort is:

Total Effort = 66*48 = 3168 hours

Assuming that the staff is working 8 hours a day and 20 days in a month, 3168hrs is equal to 19.8 months. Since we have 3 staff for coding activities, the total effort will be 19.8/3 = 6.6 man-months.

3.3.2 Mark II Function Points:

The steps explained in Section 2.2.4.2 were used to make the calculations under this heading:

For this project, the viewpoint was chosen as the *Project Viewpoint* because the aim of this thesis is to determine the size of the functionality delivered by the software and then to use this size information to estimate the project effort.

Then the boundary of the application was determined. Since use case descriptions are accepted as logical transactions and they also contain system actors, the boundary of the use cases were used to specify the boundary of the application. For this system, there are no automated users but only business users, which are Kullanici, Yetkili and UstYetkili. Also the application has no interfaces with other applications.

Definition of logical transactions is identical to the definition of use cases. Therefore, for this thesis use cases were used to define the logical transactions of MARK II FP. However, the project use cases were too much detailed. So, to better calculate the size, each use case was divided into measurable logical transactions each having their input fields, response fields and referenced object classes [61]. For these determinations, Use Case Diagrams defined in Rational Rose were used.

After this, for each logical transaction, Input Data Element Types, Data Entity Types Referenced and Output Data Element Types were determined and counted. Entity classes used or referenced in a use case were accepted as Data Entity Types Referenced. For this counting, Use Case Diagrams, Sequence Diagrams and Class Diagrams defined in Rational Rose were used.

The project use cases and their corresponding number of Logical Transactions, Input DET's, Output DET's and referenced Entities can be found in Appendix E, Table E.1.

$N_I = 159$

$N_E = 236$

$N_O = 196$

From the above information, the *Functional Size* i.e. *Function Point Index* was calculated as:

$$\begin{aligned} FPI &= W_I * \Sigma N_I + W_E * \Sigma N_E + W_O * \Sigma N_O \\ &= 0.58 * 159 + 1.66 * 236 + 0.26 * 196 \\ &= 535 \text{ MARK II FP (V1.3.1)} \end{aligned}$$

In the actual Project Plan document, effort was calculated by converting FP values into SLOC values by multiplying 30, which is a company specific value. However, this constant value is 65 in more likely for the Java applications. To better compare the actual results by the estimated ones and to see how such a difference in the selection of constants affects the final results, in this thesis the estimations were made for each of these constants.

For the constant 30:

Size = 535 * 30 = 16050 SLOC = 16.050 KLOC

For the constant 65:

Size = 535 * 65 = 34775 SLOC = 34.775 KLOC

Effort corresponding to transactions was calculated by taking productivity as 1.25 man-months for the project team based on average personal data. Total project effort was found as:

For the constant 30:

Effort = KLOC / Productivity = 16.050 KLOC / 1.25 man-months

= 12.8 man-months

For the constant 65:

Effort = KLOC / Productivity = 34.775 KLOC / 1.25 man-months

= 27.8 man-months

By taking a month as 20 days:

For the constant 30:

Effort = 12.8 man-months * 20 days/month = 256 man-day.

For the constant 65:

Effort = 27.8 man-months * 20 days/month = 556 man-day.

Since in the Counting Practices Manual, Technical Complexity Adjustment is an optional part, in the above calculations, it was not used. On the other hand, the Technical Complexity Adjustment (TCA) value was given as 0.765 by the project team. When this adjustment value was taken into account, the fallowing result were obtained:

Adjusted FPI = 535 * 0.765 = 409 MK II FP (V1.3.1)

For the constant 30:

Size = 409 * 30 = 12270 SLOC = 12.270 KLOC

For the constant 65:

Size = 409 * 65 = 26585 SLOC = 26.585 KLOC

For the constant 30:

Effort = KLOC / Productivity

= 12.270 KLOC / 1.25 man-months

= 9.8 man-months

For the constant 65:

Effort = KLOC / Productivity

= 26.585 KLOC / 1.25 man-months

= 21.2 man-months

By taking a month as 20 days:

For the constant 30:

Effort = 9.8 man-months * 20 days/month

= 196 man-day.

For the constant 65:

Effort = 21.2 man-months * 20 days/month

= 424 man-day.

3.3.3 Object Oriented Project Size Estimation

The calculations made under this heading are based on the method defined in Section 2.2.6:

1. The class model defined in Rational Rose was used for the objects' information. All the classes not only entity classes were taken. An example class model is in Appendix C.

2. For each object tokens and points were counted. These counting results can be found in Appendix F, Table F.1.

3. For each object Days Required to Develop values were calculated. These values can be found in Appendix F, Table F.1.

From the information in Appendix F, Table F.1, the Total Size and the Total Days Required to Develop the System is calculated as:

Total Size = 744 Points

Total Days Required to Develop the System = 247.4 days

Accepting 1 month as 20 days, the total project lasts in:

247.4 / 20 = 12.3 months

3.3.4 Use Case Points Method:

The Use Case Points Method was applied as the steps defined in Section 2.2.12:

1. The actors of the use case model are Yetkili, Ust Yetkili and Kullanıcı. Both of them are persons communicating with the system through a graphical user interface. So their categories were accepted as complex. (Table 2.11). These actors and their corresponding weights are given in Table 3.2.

Table 3.2 Actors and Weight Factors

Actors	Actor Categories	Weight Factors
Yetkili	Complex	3
Ust Yetkili	Complex	3
Kullanici	Complex	3

Since there are three actors with actor category of complex, *Unadjusted Actor Weight* was calculated as:

Unadjusted Actor Weight (UAW) = 3 * 3 = 9

2. Depending on the number of transactions included, the use cases were categorized by using Table 2.12. Here, use case steps were counted to get the corresponding number of transaction counts. As defined in the method, included and extending use cases were omitted. The system use cases and their corresponding categories can be found in Appendix G, Table G.1.

The number of use cases in each category was counted. Each of these counts was multiplied with the corresponding weight factors as defined in Table 2.12, and then summed to get the *unadjusted use case weights (UUCW).* Table 3.3 gives these calculations:

Table 3.3 Unadjusted Use Case Weights (UUCW)

Use Case Categories	Number of UseCases	Weight Factors	Total Weight
Simple	4	* 5	= 20
Average	25	* 10	= 250
Complex	19	* 15	= 285
UnadjustedUse CaseWeights (UUCW)			= 555

The *unadjusted use case points (UUCP)* was obtained as:

UUPC = UAW + UUCW = 9 + 555 = 564

3. For technical complexity factors (Table 2.13) and environmental factors (Table 2.14), values were determined by one of the project members depending on their rate of influence the system. Each of these values was multiplied with the corresponding weight factors given in Table 2.13 and 2.14. Then the multiplied values in the Table 3.4 were summed to get the *Tfactor* and the multiplied values in the Table 3.5 were summed to get the *Efactor*.

Table 3.4 TFactor Calculation

Description	Value	Weight	
Distributed system	0	* 2	= 0
Response or throughput performance objectives	4	* 2	= 8
End-user efficiency	5	* 1	=5
Complex internal processing	2	* 1	= 2
Reusable code	3	* 1	= 3
Easy to install	5	* 0.5	= 2.5
Easy to use	5	* 0.5	= 2.5
Portable	1	* 2	= 2
Easy to change	3	* 1	= 3
Concurrent	0	* 1	= 0
Includes security features	3	* 1	= 3
Provides access for third parties	0	* 1	= 0
Special user training facilities are required	3	* 1	= 3
TFactor			34

Table 3.5 EFactor Calculation

Description	Value	Weight	
Familiar with Rational Unified Process	2	* 1.5	= 3
Application experience	4	* 0.5	= 2
OO experience	4	* 1	= 4
Lead analyst capability	3	* 0.5	= 1.5
Motivation	4	* 1	= 4
Stable requirements	3	* 2	= 6
Part-time workers	3	* (-1)	= -3
Difficult programming language	3	* (-1)	= -3
EFactor			= 14.5

Technical complexity factors were calculated as:

TCF = 0.6 + (0.01*Tfactor) = 0.6 + (0.01*34) =0.94

Environmental factors were calculated as:

EF = 1.4 + (-0.03*Efactor) = 1.4 + (-0.03*14.5) = 0.97

Finally, the *adjusted use case points (UCP)* were calculated by the following formula:

UCP = UUCP*TCF*EF = 564*0.94*0.97 = 514

4. By Karner's recomended value [7] i.e. 20 man-hours per use case point, the project effort was estimated as:

Effort = 514*20 = 10280 man-hours

Assuming 3 developers each working 8 hours a day (3 * 8 = 24 hours a day):

Effort = 10280 / 24 = 428.3 man-days

And assuming a month as 20 days:

Effort = 428.3 / 20 = 21.4 man-months

4.3.5 J.Kammelar's Sizing Approach

In this approach, system size was calculated in terms of *component object points (COPs)* depending on the criteria motioned in Section 2.2.13:

1. Since it is better to estimate the size as early as possible and the available project specifications met the required analysis count specifications list defined in [41], analysis count type was selected.

2. The actors of the system were defined as Yetkili, UstYetkili and Kullanici.

3. Specifications i.e. use case descriptions and class diagrams were reviewed to see whether they fulfill the minimum requirements for this counting technique. Some inconsistencies were found in object naming. However, these inconsistencies were not so serious to affect the calculations.

4. For each use case, related services were determined. The logical transactions defined in Section 3.3.2 for MARK II FP were accepted as the related services for this method. Since this is an analysis count, for each service, 2 points were given (Appendix H, Table H.1).

Service Functionality (COPs) = 210

5. Counting Service/ Class Relations:

- For each use case, the relations between the services and all classes that collaborate to provide (parts of) those services were found and for every unique service / class relation, 3 points was counted (Appendix H, Table H.2). Here the significant classes were accepted same as the entity classes used in previous methods.

Total $\text{COPs}_{\text{Class Part}} = 696$

Counting Transformations:

- For each use case, transformations of the services were determined and 5 points were given for each one (Appendix H, Table H.3).

Total $\text{COPs}_{\text{Transformation Part}} = 25$

6. Counting Class Attributes:

- For each significant class, the number of attributes were counted and assigned points using Table 2.16. Inherited attributes were not taken into account (Appendix H, Table H.4).

Total $\text{COPs}_{\text{attribute Part}} = 72$

Counting Object Structures:

Generalization/specialization and aggregation/composition structures were counted depending on Table 2.17 and 2.18. Then, Object Structure Valuation Matrix defined in Table 2.19 was used to calculate the COPs (Appendix H, Table H.5).

Total $\text{COPs}_{\text{Structure Part}} = 36$

Total Class Valuation = Total $\text{COPs}_{\text{attribute Part}}$ + Total $\text{COPs}_{\text{Structure Part}}$

$= 72 + 36 = 108$ COPs

Class Functionality = Total $\text{COPs}_{\text{Class Part}}$ + Total $\text{COPs}_{\text{Transformation Part}}$

+ Total Class Valuation $= 696 + 25 + 108 = 829$ COPs

7. This step is only applicable to the design count.

8. This step is only applicable to the design count.

9. Total Size = Service Functionality + Class Functionality

= 210+ 829 = 1039 COPs

3.4 Evaluating the Results

Table 3.6 gives the summary of the estimation results found in Sections 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 and 3.3.6. Since each estimation method uses different size metrics, to get a common point for comparisons, effort values were calculated for each method and then these values were compared with each other.

When the estimations made in thesis were compared with the actual one, the fallowing results were obtained:

Lines of Code (Smith): When Smith's [67] method was applied to the project, the estimated size was found as 33.600 KLOC and effort as 6.6 man-moths. This size differs from the actual one nearly 11.000 KLOC more and the effort from the actual one as 6.6 months less (%50 underestimation).

Smith's method is assumed to applicable to C++ or equivalent level languages and takes C++ as the base language in its calculations. On the other hand, the project used in this thesis was written in Visual Age for Java. Therefore, reestimation of the method parameters by taking Java as the base language may be necessary to get better results. However, such a reestimation is out of the scope of this thesis.

Mark II Function Points: For Mark II FP method, when the company specific constant 30 was used, the results were found to be 535 MARK II FP (16.050 KLOC) for size and 12.8 man-months for effort. These results are very close to the actual ones. However, when the industrial constant 65 was used, the results were found to be 535 MARK II FP (34.775 KLOC) for size and 27.8 man-months for effort. These values are nearly twice as much as the actual ones.

Table 3.6 Method Comparison

Estimation Method	Size	Effort
Lines of Code (Smith)	33.600 KLOC	6.6 man-months
MARK II FP	535 MARK II FP or 16.050 KLOC (For const. 30)	12.8 man-months
	535 MARK II FP or 34.775 KLOC (For const. 65)	27.8 man-months
MARK II FP (Adjusted)	409 MARK II FP or 12.270 KLOC (For const. 30)	9.8 man-months
	409 MARK II FP or 26.585 KLOC (For const. 65)	21.2 man-months
OOPS	744 Points	12.3 man-moths
Use Case Points Method	514 UCP	21.4 man-months
J.Kammelar's Sizing Approach	1039 COPs	
Actual Project	≈ 21.790 KLOC	13.2 man-months

On the other hand, when adjustment was made, for both of the constants, fewer size and effort values (9.8 man-months for const. 30 and 21.2 man-months for const. 65) then the unadjusted MARK II FP were obtained. This was because of the

Technical Complexity Adjustment (TCA) value being given by the project team. This value is based on expert estimation and so open to biasing. Therefore, inaccurate estimation of this value may have caused such an underestimation when compared with the unadjusted MARK II FP. Even there was an underestimation, constant 30 still gave better results then the constant 65.

I also faced with some difficulties when calculating the Mark II FP. The fallowing findings affected my estimations:

- The project use cases were too much detailed. To better estimate the size, I divided them into logical transactions combining some use case steps. By doing so, I may have made a major assumption that might lead to significant size differences.
- Some input/output DETs had some general definitions in the Use Case and Sequence Diagrams. For example, in one of the use cases, the general definition "musavir kaydi" was shown as an input. In fact, "musavir kaydi" should contain more than 1 input field. But there were no more explanation in either Use Case or Sequence Diagrams. In such cases, I counted only 1 DET. This caused underestimations.
- Even some input/output information was absent. I tried to guess them form other project information. This may have caused underestimations in my calculations.
- For referenced entity information and some input/output DETs, I used sequence diagrams. However, there were only 27 of them. So only half of the use cases had their sequence diagrams. This made my calculations very difficult. I had to guess much information that I could easily obtain from the Sequence Diagrams. I used available diagrams to draw the absent ones. Therefore, there is some biasing in the referenced entity values.

- The level details of the Sequence Diagrams were lacking. In some diagrams, there were control classes but no related entity classes. In such cases, I had to count only this control classes.

Object Oriented Project Size Estimation (OOPS): This method differs from the others in that not the use cases but the objects (classes) are taken as the base. Since in most analysis phases not only the use cases but also the related classes (usually the entity classes) are determined, using of this method did not contradict the assumptions of this thesis. For this thesis, the class definitions in Rational Rose were used to calculate the point values and the effort.

The obtained results were 744 points for size and 12.3 man-months for the effort. So, nearly 1-month difference occurred with the actual effort.

In these calculations, I took all the classes defined in the class model. These included not only the entity classes but also others such as control and GUI classes.

One important problem was that there were three entity classes referenced in the sequence diagrams but having any definitions in the class model. Since no information was available about these classes, I did not add them to my OOPS calculations. This may have caused an underestimation.

Also, the constants of the formula reflect the industry averages. The calibration of these values using organizational data would increase the accuracy of the results. However, this is out of the scope of this thesis.

Use Case Points Method: For this method, size was found as 514 UCP and the effort as 21.4 man-months. A %55 difference was found between the actual and the estimated effort.

The following reasons may explain this overestimation:

- Most of the project use cases were written in too much detail. There were many use cases with more than 10 transactions. A. Cockburn [21] says that use cases with more than 10 steps are usually the ones whose definitions are at a too low level. So, I tried to remove the unnecessary steps from each use case. This caused many use cases to change their categories from “complex” to ”average”. But even with this simplification, the final effort resulted %55 more than the actual one.
- Some use case descriptions were nearly the same even their names were different. I took them as different use cases. This may lead to an overestimation.
- Assigning values to technical complexity and environmental factors may have affected the final results. I wanted the project team to assign these values. They tried to guess these values by expert judgment. However, since meaning of these factors differs from people to people and since it is difficult to be objective when people valuate their works, there may be some inaccuracy in calculating the technical and environmental factors. To overcome such subjectivity, these factors can be assigned values by using organizational past project data. Such a calibration may give better results.
- The most important factor affecting the final effort is the determination of the rate of man-hours per use case point. There are many recommendations for this value. However, these recommended values can cause big differences even for small changes in the point values. Therefore, this value should be calibrated for the organization. But such a calibration is out of the scope of this thesis. So, since Karner’s recommended value is 20 man-hours and when I applied Schneider and Winters’ method I again obtained 20 man-hours, I used this value in my calculations.

J.Kammelar's Sizing Approach: When I applied this method, it gave a size of 1039 COPs. However, there was no explanation anywhere on how this value can be used to get a corresponding effort value. I contacted with Kammelar and learned that he has left the company and this study stopped at this point without any definition of a size/effort relationship. Therefore, in this case study I could only estimate the size but no corresponding effort.

In this method, I used the logical transaction of MARK II FP in Section 4.3.2. I accepted the entity classes of previous methods as the significant classes of this method. By doing so, I again faced with the problems I mentioned before i.e. some existing entity classes referenced in the sequence diagrams but having any definitions in the class model and sequence diagrams having control classes but no related entity classes. For the first case, I had to guess the generalization/specialization and aggregation/composition relations of such classes by taking the similar available classes defined in the class model. For the second case, instead of entity classes I had to count only the related control classes.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

In this study, five sizing methods were chosen and applied to an OO project, whose requirements were defined in terms of a use case model. From the methods, OOPS and MARK II FP with constant 30 gave the best results. On the other hand, LOC (Smith) and adjusted MARK II FP with constant 30 underestimated the project whereas MARK II FP and adjusted MARK II FP with constant 65 and Use Case Points Method resulted in overestimation.

The case study has shown that for the use case based estimations; the most important factor is the structure of the use case modeling being used. To get accurate estimates, use cases should be defined in a suitable level of detail where each transaction can be easily identified and counted. However, in the software community, there is no such standardization in use case descriptions. This lack of standardization also became a big problem in this case study. Most of the project use cases were written in too much detail causing overestimations. So, I tried to remove the unnecessary parts from each use case. This correction resulted in better estimates, but there were still overestimations in some methods.

Another problem was about the sequence diagrams and the class model. In some of the estimations they were needed, but nearly half of the sequence diagrams were not defined and many of the available ones had little detail. Moreover, there were classes being referenced in the sequence diagrams but absent in the class model. Such inconsistencies have made the estimates difficult and open to bias.

Although all the methods being used in this case study are objective ones, the lack of standardization in the use case descriptions and the lack of some necessary project data in the sequence diagrams and the class model made my estimation results subjective. I had to simplify many use cases and try to guess the unavailable data from other similar project information, which caused overestimations and underestimations for the different methods. Therefore, knowing that experience and personal perception differences can affect the assumptions, when another person will conduct the same case study, the results will be different.

Calibration of the methods was also an important point. The method parameters being used in this case study were the ones reflecting the industrial averages. Since there was only one project available for this case study, no calibration could be done. However, if possible, such a calibration could give better results and reduce the subjectivity.

Also the selection of the constant for the MARK II FP calculations had a great impact on the final results. When the company specific value was used, better results were obtained. However, the usage of the industrial average 65 caused and overestimation. Therefore care should be taken when choosing such a constant to convert the MARK II FP values.

Besides these general problems, each of the five methods had some requirements specific to them:

- For LOC (Smith), the method is based on C++.

- For MARK II FP, use cases should be defined in a way where each logical transaction can be easily identified. Also, entity class definitions are needed.
- For OOPS, a detailed class model including objects' names, attributes, methods and parameters is a necessity.
- For Use Case Points Method, the transactions should be defined at a suitable level of detail.
- For J. Kammelar's Approach, use case services, class/service relations, class definitions and class hierarchy should be available.

With all the above reasons, organizations that want to do use case based size estimations should define their standards on use case modeling to get better results. Moreover, knowing that there are many different methods with different requirements, and knowing that each method gives the most accurate results when these requirements are hold, organizations should prefer the ones that best suit their needs and the use case modeling they used. Finally, calibration of the methods to the organizational data should be done to obtain the most benefit form these methods.

4.2 Future Work

This study is an application of five chosen methods to an average size business project. To better compare the applicability of these methods, they should be applied to other projects of different sizes and structures whose requirements are defined in a use case model.

Other OO sizing methods different from these five should also be studied to see whether there are better ones.

Finally, when possible, calibration of the methods to the organizations using past data should be done to get more accurate results.

REFERENCES

[1] A. Abran, D. St-Pierre, M. Maya, and J. M. Desharnais, “Full Function Points for Embedded and Real-Time Software”, UKSMA Fall Conference, London (UK), October 30-31, 1998.

[2] A. Abran, “COSMIC FFP 2,0: An Implementation of COSMIC Functional Size Measurement Concepts”, FESMA’99, Amsterdam, October 7, 1999.

[3] A. Abran, C. Symons, and S. Oligny, “An Overview of COSMIC-FFP Field Trial Results”, ESCOM 2001, London, England, April 2-4, 2001.

[4] F. B. Abreu, and R. Carapuça, “Candidate Metrics for Object Oriented Software with a Taxonomy Framework”, Proceedings of AQUIS’93 Conference, Venice, Italy, October 1993.

[5] A. J. Albrecht, “Measuring Application Development Productivity”, in Proc. , IBM Applications Develop. Symp., Monterey, CA, October 14-17, 1979; GUIDE Int. and SHARE, INC., IBM Corp., pp. 83.

[6] A. J. Albrecht, and J. E. Gaffney, “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November 1983.

[7] B. Anda, H. Dreiem, D. I. K. Sjoberg, and M. Jorgensen, ”Estimating Software Development Effort based on Use Cases-Experiences from Industry", 4th International Conference on the Unified Modeling Language (UML2001), Gogolla, M. and Kobryn, C. (editors), Toronto, Canada, October 1-5, 2001, pp. 487-502, LNCS 2185, Springer-Verlag, 2001.

[8] C. Archer, and S. Michael “Object Oriented Software Measures”, Technical Report, CMU/SEI-95-TR-002, ESC-TR-95-002, April 1995.

[9] M. Arnold, and P. Pedross, “Software Size Measurement and Productivity Rating in a Large- Scale Software Development Department”, Proc, Proceedings of the 1998 International Conference on Software Engineering. IEEE Comput. Soc., Los Alamitos, CA, USA, 1998.

[10] R. D. Banker, and C. F. Kemerer, “Scale Economies in New Software Development”, IEEE Transactions on Software Engineering, vol. 15, no. 10, October 1989.

[11] C. A. Behrens, “Measuring the Productivity of Computer Systems Development Activities with Function Points”, IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November 1983.

[12] V. E. Berard, “Metrics for Object Oriented Software Engineering”, http://www.ipipan.gda.pl/~marek/objects/TOA/moose.html

[13] J. M. Bieman, “Metric Development for Object Oriented Software”, Software Measurement, pp. 75-92, 1996.

[14] F. Bootsma, “Applying Full Function Points to Drive Strategic Business Improvement Within the Real-Time Software Environment”, Annual IFPUG Conference, New Orleans, October 18-22, 1999.

[15] G. J. Bozoki, “An Expert Judgment Based Software Sizing Model”, Lockheed Missiles & Space Company and Target Software.

[16] D. Bradine, “Oops, There It Is”, Enterprise Systems Journal, March 2000.

[17] Calvert, “Software Metrics”, July 1996, http://hebb.cis.uoguelph.ca/~dave/27320/new/metrics.html

[18] D. N. Card, K. El Emam, and B. Scalzo, “Measurement of Object Oriented Software Development Projects”, Technical Report, Software Productivity Consortium, January 2001.

[19] S. R. Chidamber, and C. F. Kemerer, “Towards a Metric Suite for Object Oriented Design”, pp. 197-211. Procedings: OOPSLA’91. Phoneix, Arizona, October 6-11, 1991. New York, New York: ACM SIGPLAN Notices, 1991.

[20] S. R.Chidamber, and C. F. Kemerer, “A Metrics Suit for Object Oriented Design”, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.

[21] A. Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2000.

[22] C. J. Coppick, and T. J. Cheatham, “Software Metrics for Object Oriented Systems”, pp. 317-322, Proceedings: ACM CSC’92 Conference. Kansas City, Missouri, March 3-5, 1992. New York, New York: ACM Press, 1992.

[23] *COSMIC-FFP Measurement Manual V2.1*, 2001.

[24] COSMIC Team, “COSMIC FFP-the New Software Functional Sizing Method”, September 1999.

[25] B. J. Cox, “Planning the Software Industrial Revolution”, IEEE, pp. 25-33, November 1990.

[26] F. Faghih, “Software Effort and Schedule Estimation”, 1997, http://www.enel.ucalgary.ca/People/Smith/619.94/prev689/1997.94/reports/farshad.htm

[27] R. E. Fairley, “Recent Advances in Software Estimation Techniques”, Proceedings of the 14th International Conference on Software Engineering, pp. 382-391, May 11-15, 1992, Melbourne, Australia.

[28] N. Fenton, “Software Measurement: A Necessary Scientific Basis”, IEEE Transactions on Software Engineering, vol. 20, no. 3, March 1994.

[29] N. E. Fenton, and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, Second Edition, International Thomson Computer Press, 1996.

[30] N. E. Fenton, and M. Neil, “Software Metrics: Successes, Failures and New Directions”, The Journal of Systems and Software 47 pp. 149-157, 1999.

[31] T. Fetcke, A. Abran, and T. H. Nguyen, “Mapping the OO-Jacobson Approach into Function Point Analysis”, Proceedings of TOOLS-23’97, 28 July – 1 August 1997, Santa Barbara, CA.

[32] D. Garmus, and D. Herron, “Estimating Software Earlier and More Accurately”, CrossTalk, The Journal of Defense Software Engineering, June 2002.

[33] J. W. E. Greene, “Measures for Excellence”, Quantitative Software Management Ltd.

[34] M. H. Halstead, *Elements of Software Science*, New York, Elsevier, North-Holland, 1977.

[35] T. E. Hastings, and A. S. M. Sajeev, “A Vector-Based Approach to Software Size Measurement and Effort Estimation”, IEEE Transactions on Software Engineering, vol. 27, no. 4, April 2001.

[36] B. Henderson-Sellers, “Corrigenda: Software Size Estimation of Object Oriented Systems”, IEEE Transactions on Software Engineering, vol. 23, no. 4, April 1997.

[37] M. Hitz, and B. Montazeri, “ Chidamber & Kemerer’s Metrics Suit: A Measurement Theory Perspective”, IEEE Transactions on Software Engineering, vol. 22, no. 4, April 1996.

[38] A. Hoekstra, “Metrics for Object Oriented Design”, 2001, http://www.cs.vu.nl/~ahoekst/metrics.html

[39] D. R. Jeffery, G.C. Low, and M. Barnes, “A Comparison of Function Point Counting Techniques”, IEEE Transactions on Software Engineering, vol. 19, no. 5, May 1993.

[40] C. Jones, “Strengths and Weaknesses of Software Metrics”, Technical Report, Software Productivity Research, Inc., 1997, http://dec.bournemouth.ac.uk/ESERG/downloads/Capers-strength.rtf

[41] J. Kammelar, “A Sizing Approach for OO-environments”, ECOOP, June 2000, www.iro.umontreal.ca/~sahraouh/qaoose/papers/Kammelar.pdf

[42] R. Kauffman, and R. Kumar, “Investigating Object-Based Metrics for Representing Software Output Size”, January 2002, http://academic.alliant.edu/rkumar/research/research2.htm

[43] J. K. Kearney, R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, and M. A. Adler, “Software Complexity Measurement”, Communications of ACM, vol. 29, no. 11, pp. 1044-1050, November 1986.

[44] C. F. Kemerer, “Reliability of Function Points Measurement”, Communications of the ACM, vol. 36, no. 2, February 1993.

[45] A. Lake, “Factor Analysis of Metrics”, 2001, http://prg.cpe.ku.ac.th/pipermail/prg/2001-July/000920.html

[46] L. A. Laranjeira, “Software Size Estimation of Object Oriented Systems”, IEEE Transactions on Software Engineering, vol. 16, no. 5, May 1990.

[47] C. Lokan, and A. Abran, “Multiple Viewpoints in Functional Size Measurement”, 9th International Workshop on Software Measurement, September 8-10, 1999.

[48] D. Longstreet, “Use Cases and Function Points”, November 2002, http://www.softwaremetrics.com/Articles/usecases.htm

[49] G. C. Low, and R. D. Jeffery, “Function Points in the Estimation and Evaluation of the Software Process”, IEEE Transactions on Software Engineering, vol. 16, no. 1, January 1990.

[50] J. E. Matson, B. E. Barret, and J. M. Mellichamp, “Software Development Cost Estimation Using Function Points”, IEEE Transactions on Software Engineering, vol. 20, no. 4, April 1994.

[51] M. Maya et.al., “Measuring the Functional Size of Real-Time Software”, 1998, www.lrgl.uqam.ca/publications/pdf/330.pdf

[52] E. Miranda, “Establishing Software Size Using the Paired Comparisons Method”, Ericsson Research, August 1999.

[53] E. Miranda, “Improving Subjective Estimates Using Paired Comparisions”, IEEE Software, pp. 87-91, January/February 2001.

[54] S. Oligny, J. M. Desharnais, and A. Abran, “A Method for Measuring the Functional Size of Embedded Software”, 3th International Conference on Industrial Automation, June 7-9, 1999, Montreal, Canada, http://www.lrgl.uqam.ca/news/

[55] T. Ozdamar, “Automating Function Point Analysis in Object Oriented Analysis and Design”, Master of Science Thesis, Informatics Institute, Middle East Technical University, 2001.

[56] S. Oligny, and A. Abran, “On the Compatibility Between Full Function Points and IFPUG Function Points”, Project Control for Software Quality, Shaker Publishing, 1999. ISBN 90-423-0075-2.

[57] G. Poel,, “Towards a Size Measurement Framework for Object Oriented Specifications”, Proc. Of the FESMA’98, Antwerp, Belgium, May 6-8, 1998, pp. 379-394.

[58] L. H. Putnam, and A. Fitzsimmons, “Estimating Software Costs”, Datamation, pp. 189-198, September 1979, continued in Datamation, pp. 171-178, October 1979, pp. 137-140, November 1979.

[59] K. Ribu, “Estimating Object Oriented Software Projects with Use Cases”, Master of Science Thesis, University of Oslo, November 2001.

[60] P. G. Rule, “Scenarios, Use Cases and Mark II FPA”, the Journal of The Guild of Independent Function Point Analysts (GIFPA), Issue 2, Summer 1998.

[61] P. G. Rule, “Using Measures to Understand Requirements”, 1999, www.gifpa.co.uk/library/papers/Rule/20010404_escom_pgr/v1.a.html

[62] G. Schneider, and J. Winters, *Applying Use Cases*, Addison-Wesley, 1998.

[63] L. G. M. Shaw, “Practical Software Engineering”, January 1996, http://pages.cpsc.ucalgary.ca/~mildred/451/CostEffort.html

[64] V. Y. Shen, S. D. Conte, and H. E. Dunsmore, “Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support”, IEEE Transactions on Software Engineering, vol. SE-9, no. 2, March 1983.

[65] M. Shepperd, C. Schofield, and B. Kitchenham, “Effort Estimation Using Analogy”, ICSE18, Berlin 1996.

[66] M. Shepperd, and M. Cartwright, "An Empirical Investigation of Object Oriented Software System", Technical Report No. TR 97/01, Dept. of Computing, Bournemouth University, UK, 1997.

[67] J. Smith, "The Estimation of Effort Based on Use Cases", Rational Software, 2001,

http://www.rational.com/products/whitepapers/finalTP171.jsp

[68] R. K. Smith, A.Parrish, and J. Hale, "Cost Estimation for Component Based Software Development", ACM, 1998.

[69] Software Productivity Research, "What are Feature Points?", February 2002.

[70] I. Sommerville, *Software Engineering*, Sixth Edition, Addison-Wesley, 2001.

[71] K. Srinivasan, and D. Fisher, "Meachine Learning Approaches to Estimating Software Development Effort", IEEE Transactions on Software Engineering, vol. 21, no. 2, February 1995.

[72] R. D. Stutzke, " Software Estimating Technology: A Survey", Science Applications International Corporation, Software Engineering 5th edition, 1998, pp.204-215.

[73] C. R. Symons, "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, vol. 14, no. 1, January 1988.

[74] C. Symons, "Conversion between IFPUG 4.0 and MkII Function Points", Version 3.0, August 1999.

[75] C. Symons, “Come Back Function Point Analysis (Modernized) – All is Forgiven!)”, 2001,

http://www.software-measurement.com

[76] D. P. Tegarden, S.D. Sheetz, and D. E. Monarchi, “Effectiveness of Traditional Metrics for Object Oriented Systems”, Proceedings 24th Hawaii International Conference on System Sciences 4, Kauai, Hawaii, pp. 359-368, January 7-10, 1992. Los Alamitos, California: IEEE Computer Society Press, 1991.

[77] G. Teologlou, “Measuring Object Oriented Software with Predictive Object Points”, Shaker Publishing, ISBN 90-423-0075-2, 1999.

[78] I. Vessey, and R. Weber, “Research on Structured Programming: An Empiricist’s Evaluation”, IEEE Transactions on Software Engineering, vol. SE-10, no. 4, pp. 394-407, July 1984.

[79] E. J. Weyuker, “Evaluating Software Complexity Measures”, IEEE Transactions on Software Engineering, vol. 14, no. 9, pp.1357-1365, September 1988.

[80] S. A. Whitmire, “3D Function Points: Scientific and Real-Time Extensions to Function Points”, Proc. Pacific Northwest Software Quality Conf., 1992.

[81] United Kingdom Software Metrics Association (UKSMA), “MK II Function Point Analysis Counting Practices Manual Version 1.3.1”, 1998

APPENDICES

APPENDIX A

ANNOTATED BIBLIOGRAPHY

Books

Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2000.

N. E. Fenton, and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, Second Edition, International Thomson Computer Press, 1996.

M. H. Halstead, *Elements of Software Science*, New York, Elsevier, North-Holland, 1977.

G. Schneider, and J. Winters, *Applying Use Cases*, Addison-Wesley, 1998.

I. Sommerville, *Software Engineering*, Sixth Edition, Addison-Wesley, 2001.

www

V. E. Berard, "Metrics for Object Oriented Software Engineering", http://www.ipipan.gda.pl/~marek/objects/TOA/moose.html

Calvert, “Software Metrics”, July 1996,
http://hebb.cis.uoguelph.ca/~dave/27320/new/metrics.html

F. Faghih, “Software Effort and Schedule Estimation”, 1997,
http://www.enel.ucalgary.ca/People/Smith/619.94/prev689/1997.94/reports/farshad.htm

A. Hoekstra, “Metrics for Object Oriented Design”, 2001,
http://www.cs.vu.nl/~ahoekst/metrics.html

C. Jones, “Strengths and Weaknesses of Software Metrics”, Technical Report, Software Productivity Research, Inc., 1997,
http://dec.bournemouth.ac.uk/ESERG/downloads/Capers-strength.rtf

J. Kammelar, “A Sizing Approach for OO-environments”, ECOOP, June 2000,
www.iro.umontreal.ca/~sahraouh/qaoose/papers/Kammelar.pdf

R. Kauffman, and R. Kumar, “Investigating Object-Based Metrics for Representing Software Output Size”, January 2002,
http://academic.alliant.edu/rkumar/research/research2.htm

A. Lake, “Factor Analysis of Metrics”, 2001,
http://prg.cpe.ku.ac.th/pipermail/prg/2001-July/000920.html

D. Longstreet, “Use Cases and Function Points”, November 2002,
http://www.softwaremetrics.com/Articles/usecases.htm

M. Maya et.al., “Measuring the Functional Size of Real-Time Software”, 1998,

www.lrgl.uqam.ca/publications/pdf/330.pdf

S. Oligny, J. M. Desharnais, and A. Abran, “A Method for Measuring the Functional Size of Embedded Software”, 3th International Conference on Industrial Automation, June 7-9, 1999, Montreal, Canada,

http://www.lrgl.uqam.ca/news/

P. G. Rule, “Using Measures to Understand Requirements”, 1999,

www.gifpa.co.uk/library/papers/Rule/20010404_escom_pgr/v1.a.html

L. G. M. Shaw, “Practical Software Engineering”, January 1996,

http://pages.cpsc.ucalgary.ca/~mildred/451/CostEffort.html

J. Smith, “The Estimation of Effort Based on Use Cases”, Rational Software, 2001,

http://www.rational.com/products/whitepapers/finalTP171.jsp

C. Symons, “Come Back Function Point Analysis (Modernized) – All is Forgiven!)”, 2001,

http://www.software-measurement.com

Journals, Periodicals and Articles

A. J. Albrecht, and J. E. Gaffney, “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November 1983.

R. D. Banker, and C. F. Kemerer, “Scale Economies in New Software Development”, IEEE Transactions on Software Engineering, vol. 15, no. 10, October 1989.

C. A. Behrens, “Measuring the Productivity of Computer Systems Development Activities with Function Points”, IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November 1983.

J. M. Bieman, “Metric Development for Object Oriented Software”, Software Measurement, pp. 75-92, 1996.

G. J. Bozoki, “An Expert Judgment Based Software Sizing Model”, Lockheed Missiles & Space Company and Target Software.

D. Bradine, “Oops, There It Is”, Enterprise Systems Journal, March 2000.

S. R.Chidamber, and C. F. Kemerer, “A Metrics Suit for Object Oriented Design”, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.

COSMIC Team, “COSMIC FFP-the New Software Functional Sizing Method”, September 1999.

B. J. Cox, “Planning the Software Industrial Revolution”, IEEE, pp. 25-33, November 1990.

N. Fenton, “Software Measurement: A Necessary Scientific Basis”, IEEE Transactions on Software Engineering, vol. 20, no. 3, March 1994.

N. E. Fenton, and M. Neil, “Software Metrics: Successes, Failures and New Directions”, The Journal of Systems and Software 47 pp. 149-157, 1999.

D. Garmus, and D. Herron, “Estimating Software Earlier and More Accurately”, CrossTalk, The Journal of Defense Software Engineering, June 2002.

J. W. E. Greene, “Measures for Excellence”, Quantitative Software Management Ltd.

T. E. Hastings, and A. S. M. Sajeev, “A Vector-Based Approach to Software Size Measurement and Effort Estimation”, IEEE Transactions on Software Engineering, vol. 27, no. 4, April 2001.

B. Henderson-Sellers, “Corrigenda: Software Size Estimation of Object Oriented Systems”, IEEE Transactions on Software Engineering, vol. 23, no. 4, April 1997.

M. Hitz, and B. Montazeri, “ Chidamber & Kemerer’s Metrics Suit: A Measurement Theory Perspective”, IEEE Transactions on Software Engineering, vol. 22, no. 4, April 1996.

D. R. Jeffery, G.C. Low, and M. Barnes, “A Comparison of Function Point Counting Techniques”, IEEE Transactions on Software Engineering, vol. 19, no. 5, May 1993.

J. K. Kearney, R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, and M. A. Adler, “Software Complexity Measurement”, Communications of ACM, vol. 29, no. 11, pp. 1044-1050, November 1986.

C. F. Kemerer, “Reliability of Function Points Measurement”, Communications of the ACM, vol. 36, no. 2, February 1993.

L. A. Laranjeira, “Software Size Estimation of Object Oriented Systems”, IEEE Transactions on Software Engineering, vol. 16, no. 5, May 1990.

G. C. Low, and R. D. Jeffery, “Function Points in the Estimation and Evaluation of the Software Process”, IEEE Transactions on Software Engineering, vol. 16, no. 1, January 1990.

J. E. Matson, B. E. Barret, and J. M. Mellichamp, “Software Development Cost Estimation Using Function Points”, IEEE Transactions on Software Engineering, vol. 20, no. 4, April 1994.

E. Miranda, “Establishing Software Size Using the Paired Comparisons Method”, Ericsson Research, August 1999.

E. Miranda, “Improving Subjective Estimates Using Paired Comparisions”, IEEE Software, pp. 87-91, January/February 2001.

T. Ozdamar, “Automating Function Point Analysis in Object Oriented Analysis and Design”, Master of Science Thesis, Informatics Institute, Middle East Technical University, 2001.

S. Oligny, and A. Abran, “On the Compatibility Between Full Function Points and IFPUG Function Points”, Project Control for Software Quality, Shaker Publishing, 1999. ISBN 90-423-0075-2.

L. H. Putnam, and A. Fitzsimmons, “Estimating Software Costs”, Datamation, pp. 189-198, September 1979, continued in Datamation, pp. 171-178, October 1979, pp. 137-140, November 1979.

K. Ribu, “Estimating Object Oriented Software Projects with Use Cases”, Master of Science Thesis, University of Oslo, November 2001.

P. G. Rule, “Scenarios, Use Cases and Mark II FPA”, the_Journal of The Guild of Independent Function Point Analysts (GIFPA), Issue 2, Summer 1998.

V. Y. Shen, S. D. Conte, and H. E. Dunsmore, “Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support”, IEEE Transactions on Software Engineering, vol. SE-9, no. 2, March 1983.

R. K. Smith, A.Parrish, and J. Hale, “Cost Estimation for Component Based Software Development”, ACM, 1998.

Software Productivity Research, “What are Feature Points?”, February 2002.

K. Srinivasan, and D. Fisher, “Meachine Learning Approaches to Estimating Software Development Effort”, IEEE Transactions on Software Engineering, vol. 21, no. 2, February 1995.

R. D. Stutzke, “ Software Estimating Technology: A Survey”, Science Applications International Corporation, Software Engineering 5th edition, 1998, pp.204-215.

C. R. Symons, “Function Point Analysis: Difficulties and Improvements”, IEEE Transactions on Software Engineering, vol. 14, no. 1, January 1988.

G. Teologlou, “Measuring Object Oriented Software with Predictive Object Points”, Shaker Publishing, ISBN 90-423-0075-2, 1999.

I. Vessey, and R. Weber, “Research on Structured Programming: An Empiricist’s Evaluation”, IEEE Transactions on Software Engineering, vol. SE-10, no. 4, pp. 394-407, July 1984.

E. J. Weyuker, “Evaluating Software Complexity Measures”, IEEE Transactions on Software Engineering, vol. 14, no. 9, pp.1357-1365, September 1988.

Conferences

A. Abran, D. St-Pierre, M. Maya, and J. M. Desharnais, “Full Function Points for Embedded and Real-Time Software”, UKSMA Fall Conference, London (UK), October 30-31, 1998.

A. Abran, “COSMIC FFP 2,0: An Implementation of COSMIC Functional Size Measurement Concepts”, FESMA’99, Amsterdam, October 7, 1999.

A. Abran, C. Symons, and S. Oligny, “An Overview of COSMIC-FFP Field Trial Results”, ESCOM 2001, London, England, April 2-4, 2001.

F. B. Abreu, and R. Carapuça, “Candidate Metrics for Object Oriented Software with a Taxonomy Framework”, Proceedings of AQUIS’93 Conference, Venice, Italy, October 1993.

A. J. Albrecht, “Measuring Application Development Productivity”, in Proc. , IBM Applications Develop. Symp., Monterey, CA, October 14-17, 1979; GUIDE Int. and SHARE, INC., IBM Corp., pp. 83.

B. Anda, H. Dreiem, D. I. K. Sjoberg, and M. Jorgensen, ”Estimating Software Development Effort based on Use Cases-Experiences from Industry", 4th International Conference on the Unified Modeling Language (UML2001), Gogolla, M. and Kobryn, C. (editors), Toronto, Canada, October 1-5, 2001, pp. 487-502, LNCS 2185, Springer-Verlag, 2001.

M. Arnold, and P. Pedross, “Software Size Measurement and Productivity Rating in a Large- Scale Software Development Department”, Proc, Proceedings of the 1998 International Conference on Software Engineering. IEEE Comput. Soc., Los Alamitos, CA, USA, 1998.

F. Bootsma, “Applying Full Function Points to Drive Strategic Business Improvement Within the Real-Time Software Environment”, Annual IFPUG Conference, New Orleans, October 18-22, 1999.

S. R. Chidamber, and C. F. Kemerer, “Towards a Metric Suite for Object Oriented Design”, pp. 197-211. Procedings: OOPSLA’91. Phoneix, Arizona, October 6-11, 1991. New York, New York: ACM SIGPLAN Notices, 1991.

C. J. Coppick, and T. J. Cheatham, “Software Metrics for Object Oriented Systems”, pp. 317-322, Proceedings: ACM CSC’92 Conference. Kansas City, Missouri, March 3-5, 1992. New York, New York: ACM Press, 1992.

R. E. Fairley, “Recent Advances in Software Estimation Techniques”, Proceedings of the 14th International Conference on Software Engineering, pp. 382-391, May 11-15, 1992, Melbourne, Australia.

T. Fetcke, A. Abran, and T. H. Nguyen, “Mapping the OO-Jacobson Approach into Function Point Analysis”, Proceedings of TOOLS-23’97, 28 July – 1 August 1997, Santa Barbara, CA.

C. Lokan, and A. Abran, “Multiple Viewpoints in Functional Size Measurement”, 9th International Workshop on Software Measurement, September 8-10, 1999.

G. Poel,, “Towards a Size Measurement Framework for Object Oriented Specifications”, Proc. Of the FESMA’98, Antwerp, Belgium, May 6-8, 1998, pp. 379-394.

M. Shepperd, C. Schofield, and B. Kitchenham, “Effort Estimation Using Analogy”, ICSE18, Berlin 1996.

D. P. Tegarden, S.D. Sheetz, and D. E. Monarchi, “Effectiveness of Traditional Metrics for Object Oriented Systems”, Proceedings 24th Hawaii International Conference on System Sciences 4, Kauai, Hawaii, pp. 359-368, January 7-10, 1992. Los Alamitos, California: IEEE Computer Society Press, 1991.

S. A. Whitmire, “3D Function Points: Scientific and Real-Time Extensions to Function Points”, Proc. Pacific Northwest Software Quality Conf., 1992.

Technical Reports and Manuals

C. Archer, and S. Michael “Object Oriented Software Measures”, Technical Report, CMU/SEI-95-TR-002, ESC-TR-95-002, April 1995.

D. N. Card, K. El Emam, and B. Scalzo, “Measurement of Object Oriented Software Development Projects”, Technical Report, Software Productivity Consortium, January 2001.

COSMIC-FFP Measurement Manual V2.1, 2001.

C. Symons, “Conversion between IFPUG 4.0 and MkII Function Points”, Version 3.0, August 1999.

M. Shepperd, and M. Cartwright, “An Empirical Investigation of Object Oriented Software System”, Technical Report No. TR 97/01, Dept. of Computing, Bournemouth University, UK, 1997.

United Kingdom Software Metrics Association (UKSMA), “MK II Function Point Analysis Counting Practices Manual Version 1.3.1”, 1998

APPENDIX B

OO SIZE METRICS AND METHODS

Table B.1 Sizing Methods and Related Metrics

Methods	**Related Metrics**
Expert estimations	No special metrics, usually LOC
Counting LOC	NCLOC, CLOC, ES, DSI, bytes of computer storage, number of characters
Software Science	Operators and Operands
Function Point Analysis (FPA) and its alternatives	Function Points
Statistical Object Model (SOM)	Object Decomposition and Learning Curves
Object-Oriented Project Size Estimation (OOPS)	Systemmeter, Point Value
Distance-Based Approach	Definition of distance
Vector-Based Approach	Operator and Operands of ADTs
Object Points	Object Points
Predictive Object Points (POPs)	TLC, DIT, NOC, WMC
Use Case Points Method	Use Case Points
J.Kammelar's Sizing Approach	Component Object Points

APPENDIX C

SAMPLE CLASS DIAGRAM AND OBJECT DEFINITION

C.1 Sample Class Diagram

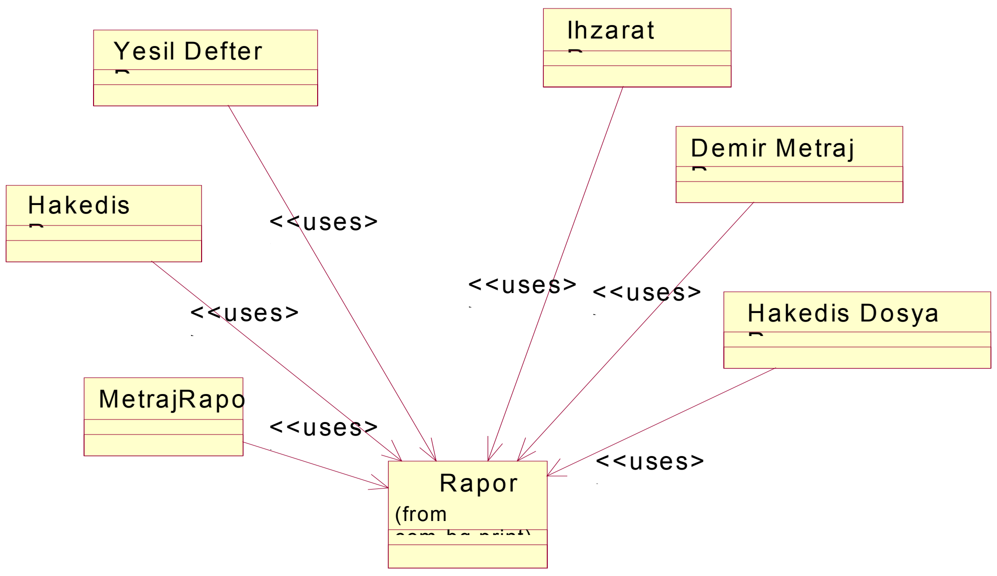


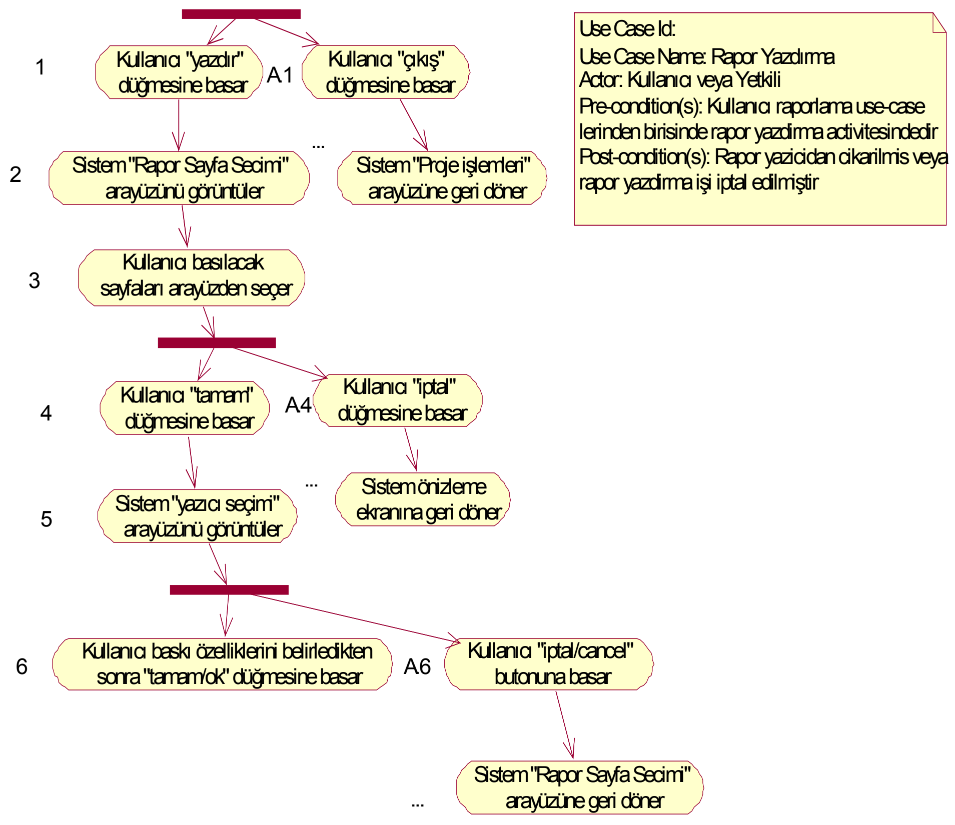
Figure C.1 Reporting Class Diagram

C.2 Sample Object Definition

```
class KullaniciGirisArayuzu

{

boolean create ()

boolean b_Yeni ()

boolean b_Sil ()

boolean b_Kaydet ()

boolean displayKullaniciKaydi (Hashtable pFields)

boolean clearFormFields ()

boolean b_Ileri ()

boolean b_Geri ()

}
```

APPENDIX D

A SAMPLE USE CASE DESCRIPTION


1
Kullanıcı "yazdır" düğmesine basar
A1
Kullanıcı "çıkış" düğmesine basar
2
Sistem "Rapor Sayfa Secimi" arayüzünü görüntüler
...
Sistem "Proje işlemleri" arayüzüne geri döner
3
Kullanıcı basılacak sayfaları arayüzden seçer
4
Kullanıcı "tamam" düğmesine basar
A4
Kullanıcı "iptal" düğmesine basar
5
Sistem "yazıcı seçimi" arayüzünü görüntüler
...
Sistem önizleme ekranına geri döner
6
Kullanıcı baskı özelliklerini belirledikten sonra "tamam/ok" düğmesine basar
A6
Kullanıcı "iptal/cancel" butonuna basar
...
Sistem "Rapor Sayfa Secimi" arayüzüne geri döner

Use Case Id:
Use Case Name: Rapor Yazdırma
Actor: Kullanıcı veya Yetkili
Pre-condition(s): Kullanıcı raporlama use-case lerinden birisinde rapor yazdırma activitesindedir
Post-condition(s): Rapor yazicidan cikarilmis veya rapor yazdırma işi iptal edilmiştir

Figure D.1 Rapor Yazdırma Use Case Description

APPENDIX E

MARK II FUNCTION POINTS

Table E.1 MARK II FP Count

Use Case Id	# of Logical Transactions	# of Input DET's	# of Ouput DET's	Entities Referenced
1	2	2	3	11
2	3	11	9	9
3	2	11	-	7
4	2	2	11	5
5	2	2	4	4
6	2	2	3	4
7	2	2	9	4
8	2	1	-	4
9	2	1	1	6
10	3	9	8	8
11	2	1	2	6
12	2	6	4	5
13	2	2	7	5
14	2	4	4	5
15	3	2	3	6
16	2	1	1	5
17	2	5	7	5
18	2	2	2	3
19	3	11	12	11
20	5	12	10	15
21	2	4	13	5
22	2	3	3	5
23	2	3	3	5
24	2	3	5	5

Table E.1 MARK II FP Count (cont.)

25	4	12	11	7
26	2	1	1	5
27	2	2	2	5
28	2	3	4	5
29	1	-	-	-
30	2	1	2	8
31	1	-	9	2
32	2	7	7	5
33	1	-	-	-
34	2	3	4	5
35	3	2	1	1
36	2	3	3	5
37	2	1	2	5
38	2	1	-	-
39	2	1	1	-
40	2	1	1	-
41	2	1	1	2
42	2	1	1	2
43	3	3	5	6
44	2	1	-	5
45	3	6	8	6
46	2	4	-	5
47	2	2	8	4
48	2	1	1	5
Total		**159**	**196**	**236**

APPENDIX F

OBJECT ORIENTED PROJECT SIZE ESTIMATION

Table F.1 OOPS Count

Object Name	# of Tokens for Name	# of Tokens for Attributes	# of Tokens for Methods	# of Tokens for Method Parameters	Total Points	Days Required to Develop
1	4	0	1	0	5	1.83
2	4	0	11	10	24	8.84
3	1	0	0	9	10	3.67
4	2	0	0	0	2	0.73
5	1	0	0	0	1	0.36
6	2	1	6	2	11	4.04
7	0	0	0	9	9	3.3
8	1	0	0	11	12	4.41
9	1	0	0	9	10	3.67
10	0	0	0	9	9	3.30
11	0	0	4	0	4	1.46
12	1	6	30	37	74	27.53
13	0	14	5	15	34	12.55
14	0	0	0	14	14	5.15
15	0	0	0	14	14	5.15
16	0	1	0	13	14	5.15
17	0	0	0	14	14	5.15
18	0	0	0	14	14	5.15
19	1	0	0	12	13	4.78
20	1	0	2	3	6	2.20
21	0	0	1	26	27	9.95
22	0	0	1	2	3	1.10
23	1	0	4	1	6	2.20

Table F.1 OOPS Count (cont.)

24	1	0	0	2	3	1.10
25	0	7	2	23	32	11.81
26	0	3	1	21	25	9.21
27	0	0	1	2	3	1.10
28	0	0	0	6	6	2.20
29	0	0	0	2	2	0.73
30	0	11	0	14	25	9.21
31	0	23	0	14	37	13.67
32	0	7	0	15	22	8.10
33	1	27	8	21	57	21.14
34	0	12	0	19	31	11.44
35	0	12	0	19	31	11.44
36	1	12	0	19	32	11.81
37	0	12	0	19	31	11.44
38	0	12	0	19	31	11.44
39	1	12	0	19	32	11.81
40	1	0	0	1	2	0.73
41	0	0	0	2	2	0.73
42	1	0	0	2	3	1.10
43	0	0	0	4	4	1.46
44	0	0	1	1	2	0.73
45	0	0	0	1	1	0.36

APPENDIX G

USE CASE POINTS METHOD

Table G.1 Use Cases and Corresponding Categories

Use Case Id	**Number of Transactions Included**	**Use Case Categories**
1	7	Average
2	9	Complex
3	7	Average
4	3	Simple
5	11	Complex
6	13	Complex
7	5	Average
8	4	Average
9	4	Average
10	10	Complex
11	5	Average
12	6	Average
13	5	Average
14	8	Complex
15	7	Average
16	10	Complex
17	13	Complex
18	7	Average
19	9	Complex
20	14	Complex
21	5	Average
22	6	Average
23	7	Average

Table G.1 Use Cases and Corresponding Categories (cont.)

24	9	Complex
25	12	Complex
26	10	Complex
27	10	Complex
28	7	Average
29	2	Simple
30	5	Average
31	3	Simple
32	15	Complex
33	2	Simple
34	7	Average
35	11	Complex
36	10	Complex
37	7	Average
38	4	Average
39	4	Average
40	5	Average
41	5	Average
42	5	Average
43	17	Complex
44	7	Average
45	17	Complex
46	7	Average
47	5	Average
48	10	Complex

APPENDIX H

J.KAMMELAR’S SIZING APPROACH

Table H.1 Use Cases and Corresponding Number of Services

Use Case Id	Number of Services Included	COPs
1	2	*2 = 4
2	3	*2 = 6
3	2	*2 = 4
4	2	*2 = 4
5	2	*2 = 4
6	2	*2 = 4
7	2	*2 = 4
8	2	*2 = 4
9	2	*2 = 4
10	3	*2 = 6
11	2	*2 = 4
12	2	*2 = 4
13	2	*2 = 4
14	2	*2 = 4
15	3	*2 = 6
16	2	*2 = 4
17	2	*2 = 4
18	2	*2 = 4
19	3	*2 = 6
20	5	*2 = 10
21	2	*2 = 4
22	2	*2 = 4
23	2	*2 = 4

Table H.1 Use Cases and Corresponding Number of Services (cont.)

24	2	*2 = 4
25	4	*2 = 8
26	2	*2 = 4
27	2	*2 = 4
28	2	*2 = 4
29	1	*2 = 2
30	2	*2 = 4
31	1	*2 = 2
32	2	*2 = 4
33	1	*2 = 2
34	2	*2 = 4
35	3	*2 = 6
36	2	*2 = 4
37	2	*2 = 4
38	2	*2 = 4
39	2	*2 = 4
40	2	*2 = 4
41	2	*2 = 4
42	2	*2 = 4
43	3	*2 = 6
44	2	*2 = 4
45	3	*2 = 6
46	2	*2 = 4
47	2	*2 = 4
48	2	*2 = 4
Service Functionality (COPs)		**210**

Table H.2 Use Cases and Corresponding Service-Class Relationships

Use Case Id	# of Services	Related Classes	COPs
1	2	11	*3= 33
2	3	9	*3= 27
3	2	7	*3= 21
4	2	5	*3 = 15
5	2	4	*3= 12
6	2	4	*3= 12
7	2	4	*3 = 12
8	2	4	*3= 12
9	2	6	*3= 18
10	3	8	*3 = 24
11	2	6	*3= 18
12	2	5	*3= 15
13	2	5	*3 = 15
14	2	5	*3= 15
15	3	6	*3= 18
16	2	5	*3 = 15
17	2	5	*3= 15
18	2	3	*3= 9
19	3	11	*3 = 33
20	5	15	*3= 45
21	2	5	*3= 15
22	2	5	*3 = 15
23	2	5	*3= 15
24	2	5	*3= 15
25	4	7	*3 = 21
26	2	5	*3= 15
27	2	5	*3= 15
28	2	5	*3 = 15
29	1	-	*3= 0
30	2	8	*3= 24
31	1	2	*3 = 6
32	2	5	*3= 15
33	1	-	*3= 0

Table H.2 Use Cases and Corresponding Service-Class Relationships (cont.)

34	2	5	*3 = 15
35	3	1	*3= 3
36	2	5	*3= 15
37	2	5	*3 = 15
38	2	-	*3= 0
39	2	-	*3= 0
40	2	-	*3 = 0
41	2	2	*3= 6
42	2	2	*3= 6
43	3	6	*3 = 18
44	2	5	*3= 15
45	3	6	*3= 18
46	2	5	*3 = 15
47	2	5	*3= 15
48	2	5	*3 = 15
Total COPs$_{\textbf{Class Part}}$			**696**

Table H.3 Use Cases and Corresponding Number of Transformations

Use Case Id	Number of Transformations Included	COPs
1	0	*5 = 0
2	0	*5 = 0
3	0	*5 = 0
4	0	*5 = 0
5	0	*5 = 0
6	0	*5 = 0
7	0	*5 = 0
8	0	*5 = 0
9	1	*5 = 5
10	0	*5 = 0
11	1	*5 = 5
12	0	*5 = 0
13	0	*5 = 0
14	0	*5 = 0
15	0	*5 = 0
16	0	*5 = 0
17	0	*5 = 0
18	0	*5 = 0
19	0	*5 = 0
20	1	*5 = 5
21	0	*5 = 0
22	0	*5 = 0
23	0	*5 = 0
24	2	*5 = 10
25	0	*5 = 0
26	0	*5 = 0
27	0	*5 = 0
28	0	*5 = 0
29	0	*5 = 0
30	0	*5 = 0
31	0	*5 = 0

Table H.3 Use Cases and Corresponding Number of Transformations (cont.)

32	0	*5 = 0
33	0	*5 = 0
34	0	*5 = 0
35	0	*5 = 0
36	0	*5 = 0
37	0	*5 = 0
38	0	*5 = 0
39	0	*5 = 0
40	0	*5 = 0
41	0	*5 = 0
42	0	*5 = 0
43	0	*5 = 0
44	0	*5 = 0
45	0	*5 = 0
46	0	*5 = 0
47	0	*5 = 0
48	0	*5 = 0
Total COPs$_{\textbf{Transformation Part}}$		**25**

Table H.4 Classes and Corresponding Number of Attributes

Class Id	Number of Attributes Included	COPs
6	1	2
12	2	2
21	0	2
13	9	7
14	0	2
15	0	2
16	1	2
17	0	2
18	0	2
25	3	5
26	3	5
30	6	5
32	5	5
31	14	7
39	12	7
46	Not identified	5
47	Not identified	5
48	Not identified	5
Total COPs$_{\textbf{attribute Part}}$		**72**

Table H.5 Object Structure Valuation Matrix for the Project

Structure Part (Classes)	**Ass. Type**	**Number of Levels**	**Number of Sub-Classes**	**COPs**
6	-	-	-	0
12	-	-	-	0
21	-	-	-	0
13	Gen./Spec.	1	-	1*3 = 3
15	Gen./Spec.	1	-	1*3 = 3
16	Gen./Spec.	1	-	1*3 = 3
17	Gen./Spec.	1	-	1*3 = 3
18	Gen./Spec.	1	-	1*3 = 3
25	-	-	-	0
26	-	-	-	0
30	Gen./Spec.	1	-	1*3 = 3
31	Gen./Spec.	1	-	1*3 = 3
39	Gen./Spec.	-	-	0
46	Gen./Spec.	1	-	1*3 = 3
47	Gen./Spec.	1	-	1*3 = 3
48	Gen./Spec.	1	-	1*3 = 3
14	Gen./Spec.	1	-	1*3 = 3
32	Gen./Spec.	1	-	1*3 = 3
Total COPs$_{\textbf{Structure Part}}$				**36**