An Evaluation of the Design of Use Case Points (UCP)

Joost Ouwerkerk[1] and Alain Abran[1]

[1] École de Technologie Supérieure - ETS
1100 Notre-Dame Ouest, Montréal, Canada H3C 1K3
joosto@gmaillcom; alain.abran@etsmtl.ca

Abstract. The increasing popularity of use-case driven development methodologies has generated an industrial interest in software size and effort estimation based on Use Case Points (UCP). This paper presents an evaluation of the design of the UCP measurement method. The analysis looks into the concepts, as well as the explicit and implicit principles of the method, the correspondence between its measurements and empirical reality and the consistency of its system of points and weights.

Keywords: Use Cases, Use Case Points, UCP, Function Points, FP, Functional Size Measurement.

1 Introduction

In the early days of measurement, a software project's size was measured by lines of code (LOC), and, since the early '80s, by functional sizing methods like Function Points Analysis (FPA) [ALB79, ISO 20926] [ABR03, ISO 19761]. With the subsequent increase in popularity of development methodologies based on use cases (UML, Agile, RUP) has come a concomitant interest in new methods for measuring the size of use-case-driven software projects.

The use case points (UCP) sizing method was initially designed by Gustav Karner of the company Objectory AB in 1993 [KAR93], and there has been little modification of it since then. UCP is an adaptation of FPA for measuring the size of projects, the functional requirements specifications of which are described by a use-case model. The use of UCP has been reported in [CAR05] [CLE06] [MOH05], and some commercial tools to help with the calculation of UCP have already emerged in the marketplace[1].

To evaluate the usefulness and validity of UCP sizing, this article proposes an analysis of the design of the UCP measurement method, looking into the explicit and implicit principles of the UCP method, the correspondence between its measurements and empirical reality, and the consistency of its system of points and weights. This evaluation approach is based on an evaluation process previously used for investigating the design of cyclomatic complexity [ABR04], Halstead's metrics [ALQ05] and IFPUG function points [ABR94].

[1] For example: Duvessa Estimate Easy Use Case - www.duvessa.com

The structure of the article is as follows: section 2 introduces the UCP method, its origins and its procedures; section 3 presents the evaluation model; and section 4 the analysis results for the UCP method. In section 5, the results are discussed and the conclusions summarized.

2 Use Case Points

2.1 Origins

The use-case approach was developed by Ivar Jacobson while at Ericsson in the sixties. The idea was first described in 1987 [JAC87]. Like many of Jacobson's ideas, such as the sequence diagram, the concept of use cases was integrated into the Rational Unified Process (RUP) when Objectory AB was bought by Rational (now IBM) in 1995.

A use case is a simple but flexible technique for capturing the requirements of a software system. Each use case is composed of a number of *scenarios*, written in an informal manner in the language of the business domain to describe the interactions between the actors and a system.

In the Objectory and RUP process models, use cases are key to ensuring the traceability of requirements throughout the development cycle. It should be possible to trace a use case from requirements specification through to the architecture, design, code and testing phases?

In 1993, Gustav Karner [KRA93] proposed the UCP measurement method, an adaptation of FPA, to measure the size of software developed with the Objectory use-case approach. This method is aimed at measuring software functional size as early as possible in the development cycle.

2.2 Description

The design of UCP takes three aspects of a software project into account: 1) use cases; 2) technical qualities; and 3) development resources.

A) Unadjusted Use Case Points – UUCP

Use Cases are represented by a number of Unadjusted Use Case Points (UUCP). To derive the UUCP, the complexity of the system's actors and use cases must be evaluated. Table 1 summarizes the calculation of UCCP points:

- Each simple actor has a weight of 1,
- Each actor of average complexity has a weight of 2,
- and so on.

B) Technical qualities

Technical qualities are represented by a Technical Complexity Factor (TCF), which consists of 13 technical qualities (Table 3), each with a specific weight, combined into a single factor. To calculate the TCF, an expert must assess the relevance to the project of each technical quality, evaluated on a scale from 0 to 5 (where 0 is 'not applicable' and 5 is 'essential'). The weights are balanced in such a way that a relevance factor of 3 for each quality will produce a TCF equal to 1.

The TCF is thus the sum of all the relevance factors (one for each technical quality) multiplied by their corresponding weights plus two constants, C2 (0.1) and C1 (0.6): Table 3 lists the quality factors and their corresponding weights. Karner bases his design for these weights on the constants and weights of the FPA Value Adjustment Factors [ALB79].

C) Development Resources

Development resources are represented by Environment Factors (EF), also referred to as experience factors [CAR05]. The UCP model identifies eight such factors (Table 4) contributing to the effectiveness of the development team. To calculate the EF, an expert must assess the importance of each factor and classify it on a scale from 0 to 5 (0 meaning 'very weak'; 5 meaning 'very strong'). The selection of the weights is balanced such that a value of 3 for each factor will produce an EF of 1. The EF is the sum of all the factors multiplied by their weights and two constants, C2 (-0.03) and C1 (1.4).

The number of UCP is the product of these three components, UUCP, TCTP and EF.

Table 1: ACTOR Weights

Complexity	Definition.	Weight
Simple	System interaction via API.	**1**
Average	Average interaction system via protocol, or Human interaction via a command line.	**2**
Complex	Complex human interaction via a graphical user interface	**3**

Table 2: USE CASE Weights

Complexity	Definition	Weight
Simple	3 transactions or fewer; 5 analysis classes or fewer	**5**
Average	4 to 7 transactions; 5 to 10 analysis classes	**10**
Complex	Over 7 transactions; Over 10 analysis classes	**15**

Table 3: Technical Quality Factors – TCF		
Factor	**Description**	**Weight**
F1	Distributed system	**2**
F2	Performance (response time or flow)	**1**
F3	Efficiency of user interface	**1**
F4	Processing complexity	**1**
F5	Reusability	**1**
F6	Installability	**0.5**
F7	Operability	**0.5**
F8	Portability	**2**
F9	Maintenability	**1**
F10	Simultaneous access	**1**
F11	Security	**1**
F12	Direct access for third parties	**1**
F13	Training features or online help	**1**

Table 4: Environmental Factors – EF		
Factor	**Description.**	**Weight**
F1	Familiarity with the methodology	**1.5**
F2	Part-time status	**-1**
F3	Analysis capability	**0.5**
F4	Experience with the application	**0.5**
F5	Experience with object-oriented methodology	**1**
F6	Motivation	**1**
F7	Difficulty of the programming language	**-1**
F8	Stability of the specifications	**2**

For estimation purposes, the number of UCP can then be combined with a productivity constant (referred to as Mean Resources – MR) representing the ratio of man-hours per UCP. Karner proposes that each organization require its own productivity constant. The MR constant for the Objectory projects described in Karner's paper was approximately 20.

2.3 Related work on industrial applications of UCP

Four studies have been identified on the use of variants of UCP, and these are discussed next. No other detailed study has been identified documenting the use of UCP as is for estimation purposes.

On the basis of a single case study, [NAG01] reports that the UCP method can be more reliable than FPA to predict testing effort. The approach described in [NAG01] is a variant of Karner's, proposing nine technical qualities (instead of thirteen) associated with testing activities (for example, test tools and test environment) and ignores the development resource factors (EF). For the single project studied, the

effort estimated by the UCP method was reported to be only 6% lower than the actual effort. The author himself stresses that this "*estimation technique is not claimed to be rigorous*". Of course, this study represents a single case study and lacks generalization power.

The adapted UCP method described in [MOH05] suggests that iterative projects need special rules to account for the ongoing reestimation of changing requirements during the course of a project. Realizing that an organization's resources are more or less constant, the adapted method replaces the resource factor (EF) by a simple increase in the productivity constant (MR).

In this variant of the UCP method, another dimension is introduced: the overhead factor (OF), which represents the effort of project management and another activities not directly related to functionality. For the two projects presented in [MOH05], the efforts estimated by UCP were 21% and 17% lower than the actual effort. Again, this study refers to only two projects and lacks generalization power.

The method used in [CAR05] is, in essence, the same as Karner's, but with the addition of a new "risk coefficient", specific to each project, to account for risk factors not already covered by the basic model (for instance, "reuse effort". The risk coefficient is a simple percentage of effort added according to the opinion of an estimation expert. The method described in the [CAR05] study was reportedly used with over 200 projects over a 5-year period, and produced an average deviation of less than 9% between estimated effort (using the UCP method) and recorded effort; no information is provided about the dispersion across this average, nor about the statistical technique used to build the estimation model or the classical quality criteria of estimation models such as the coefficient of determination (R^2) and Mean Relative Error. In brief, no documented details are given to support the claim about the accuracy of the estimates.

In summary, the UCP measurement method itself was modified in the three studies surveyed, and care must be exercised when comparing data from these three method variants, since no information about convertibility back to the original UCP is provided. Similarly, the empirical information given in these three studies cannot be generalized to a larger context due to either the very small sample (one or two case studies) or lack of supporting evidence.

3 Evaluation approach

As shown in [ALQ05], a measurement method can be analyzed by reverse engineering its design. Accordingly, the framework for measurement design proposed in [HAB06] is used here to study the measurement design of UCP, the terminology in [HAB06] being mostly based on ISO standards.

The analysis of the UCP measure that follows is structured according to the design activities identified in [HAB06]:

- Definition of measurement principles: knowledge and understanding of the concepts being measured, i.e. the model of the domain and the entities and attributes being measured;
- Definition of the measurement method: how the measurement makes the link between the empirical world and the numerical world;
- Definition of the operational procedures of measurement: the operational practices used to apply the method.

[HAB06] reminds us that there are different types of mathematical scales, and categorical rules for transforming one scale to another. The scale types typically used are: nominal (values are non-ordered categories), ordinal (values are ordered categories), interval (there is a concept of relative "distance" between the values) and ratio (absolute intervals with a meaningful zero-value).

4 Analysis of the UCP measurement design

4.1 Empirical principles

UCP is aimed at measuring the functional size of a software system described by a functional requirement specification written in use-case form.

Although based on the FP structure, [KAR93] proposes several other factors, including non-functional requirements as well as development resource characteristics.

Table 5 lists the five entities currently being measured by the UCP method described in [KAR93].

As identified in [SMI99], the fairly vague definition of a use case poses an important problem for the calculation of UCP. The way in which a use case is written is not standardized; it can be identified and documented at any level of granularity. What is an elementary use case? How can we even characterize the level of granularity of a particular use case? Are use cases being characterized in a consistent manner by one or more analysts working on the same project, and across projects?

Successful measurement using the UCP method will therefore directly and strongly depend on a degree of uniformity in the writing of use cases from one project to another.

For the entities listed in Table 5, eleven different attributes have been identified (Table 6), each of which is taken into account and measured by the UCP method described in [KAR93].

The set of entities and attributes measured by UCP is illustrated in Figure 1. The UCP measurement process clearly includes a mix of several entities (actors, use cases, requirements, team, programming language) for which distinct attributes are quantified and then combined.

In summary:

- The resulting units of measurement are unknown and unspecified despite the UCP label.
- It is obvious from Tables 5 and 6 that the end-result cannot be uniquely expressed in terms of use cases.

Table 5: ENTITIES

Entity	Description
Actor	A use case, as defined by [JAC87], describes the interaction between the actors and the system. The actor is any agent (machine or human) that acts upon system functionality.
Use Case	A simple functional requirement description for a specific goal, written in the form of a sequence of interactions between an actor and the system.
Specification of requirements	The set of planned requirements for a system, including the functional requirements (written in use-case form) and other non-functional requirements.
Development team	The human resources participating in the project of designing, programming and testing the system.
Programming language	The computer programming language used by the development team to code the software system (Java or C++, for instance).

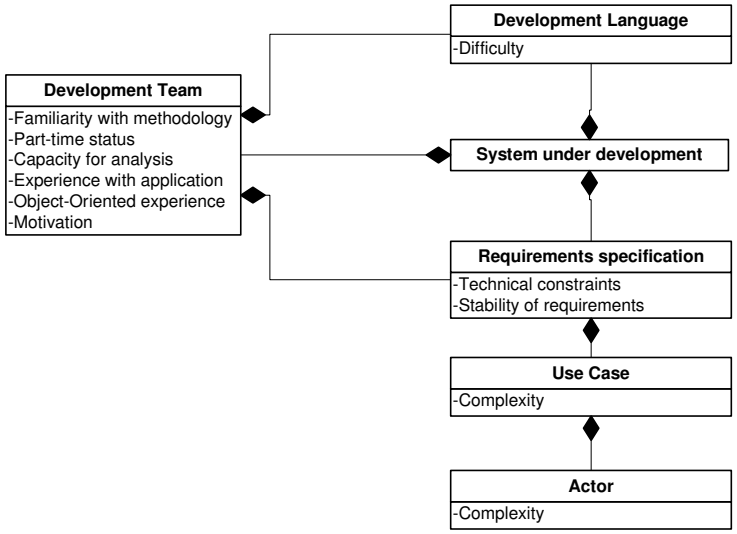

Figure 1: Set of entities and attributes measured in UCP

Table 6: ATTRIBUTES		
Entity	**Attribute**	**Measurement rule**
Actor	Complexity (of actor)	The type of complexity (simple, average or complex) of the interaction between the actor and the system.
Use case	Complexity (of use case)	The type of complexity (simple, average or complex) measured in the number of transactions.
Specification of requirements	Relevance of the technical quality requirements	The level of relevance (from 0 to 5) of each of the 13 known non-functional qualities
	Stability of requirements	The level of stability (from 0 to 5) of the functional and non-functional requirements
Development team	Familiarity with the methodology	The level (from 0 to 5) of skills and knowledge of the development methodology in use for the project.
	Part-time status	The level (from 0 to 5) of part-time staff on the team
	Analysis capability	The level (from 0 to 5) of analysis capabilities of the development team with respect to project needs.
	Application experience	The level (from 0 to 5) of team experience with the application domain of the system
	Object-oriented experience	The level (from 0 to 5) of team experience with object-oriented design
	Motivation.	The level (from 0 to 5) of team motivation
Programming language	Difficulty	The level (from 0 to 5) of programming difficulty

4.2 Scale types of the attributes

The UCP measurement method includes two categories of attributes for the six entity types being measured: types and levels. These attributes are quantified using a variety of scales types, and then they are combined. This section analyzes these scale-type manipulations and identifies the corresponding measurement issues.

The “complexity” attribute, assigned to actors and use cases, is at the core of UCP (collectively defined as the UUCP factor). It is categorized as being of the ordinal scale type using a scale of three values: simple, average and complex. Thus an actor categorized by the measurer as “simple” is considered less complex than an “average” actor, and an “average” actor less complex than a “complex” actor. The scale is similar for use cases: the same category labels are used (simple, average and complex), however it cannot be assumed that the categories and the categorization process are similar, since different entity types are involved.

The technical and resource factors are also all evaluated through a categorization process on an ordinal scale, but one with integers from 0 to 5 (0, 1, 2, 3, 4 or 5); it must be noted that these numbers do not represent numerical values on a ratio scale, but merely a category on an ordinal scale; that is, they are merely ordered labels and not numbers. Thus, a programming language assigned a difficulty level of 1 is considered to be less difficult for the development team than a programming language of difficulty level 2, but cannot be considered to be exactly one unit less difficult than one categorized as having a difficulty level of 2 because these levels are being measured on an ordinal scale. There is indeed no justification provided in the description of the UCP model to support a ratio scale (for example, that a programming language of factor 4 is twice as difficult as a programming language of factor 2). The levels must therefore be regarded as being on an interval scale. It must also be noted that, even though they have the same labels, e.g. 1,2,3, etc., the intervals are not necessarily regular; for example, each label might represent a different interval, and each interval may not be, and does not need to be, regular within an attribute being measured – see, for instance, the measurement rules for the Technical Adjustment Factors in FPA [ABR94].

4.3 The measurement method for the entities

The measurement process for assigning a value to the complexity attributes of actors and use cases is based on categorical rules. For example, if an actor acted on the system by means of a graphical user interface, then the value assigned to it is labeled "complex"; as explained in the previous section, this value is on an ordinal scale.

For use-case entities, the measurement process for assigning a value comes first from counting the number of transactions or the number of analysis classes, and then, as a second step, looking up the correspondence rules in a two-dimensional table (transactions and analysis classes) to assign a corresponding ordered label of an ordinal-type scale. Thus, if a use case contains four transactions, it is assigned the category label "simple".

Both the technical factors and the resources are evaluated qualitatively by a measurer. The UCP method does not prescribe any rule for assigning a value to, for example, the level of familiarity with a methodology. Without criteria for the evaluation of levels, the UCP measurement process lacks strict repeatability, or even reproducibility; one measurer (beginner or expert) can easily evaluate a factor differently from another measurer.

4.4 Correspondence mapping between the empirical and numerical worlds

The principle of homomorphism requires that the integrity of attribute ratios and relationships be maintained when translating from the empirical model to the numerical model.

Below, we evaluate these numerical correspondence mappings in UCP for each attribute of the empirical model.

Actor complexity: If we accept that an application programming interface (API) represents less functionality than a command-line interface, which in turn represents less functionality than a graphical user interface, then we can say that the categorization of the actors by their type of interface represents a valid correspondence. But if, after assigning a "weight" of 1, 2 or 3 to the actor types, the model uses these ordered values in sums and products, then it is effectively making a translation from an ordinal scale (complexity categories) to a ratio scale (UCP weights) without justification, which is not a mathematically valid operation. Furthermore, there is no documented data to demonstrate that a graphical interface represents three times more "functionality" than an application programming interface.

Use-case complexity: The UCP model transforms the measurements of use-case complexity from a ratio-type scale (the number of transactions or classes of analysis) into an ordinal-type scale (complexity categories), and then back to a ratio-type scale (UCP weights). The arbitrary assignment of the weights (5 for simple, 10 for average and 15 for complex) could have been avoided if the number of transactions or classes of analysis had been kept as numbers on a ratio-type scale (rather than losing this quantitative precision by mapping them to only three ordered categories with arbitrarily assigned values of 5, 10 and 15.

Technical qualities: the UCP model does not propose any criteria for measuring the attribute values of technical qualities. The result is an entirely subjective measure. To address the same kind of arbitrary assignment of values in the "technical adjustment factors" of the FP model, the IFPUG organization worked out detailed evaluation criteria for each of the non functional characteristics of the system and documented these in its Counting Practices Manual.

Both the constants and weights of UCP's technical qualify factors (TCF) calculation are derived directly from the Albrecht's FP model. It has been noted in [ABR94] that the TCF suffer from a large number of inadmissible mathematical operations moving up and down the scale types, and then using these numbers in inappropriate ways.

Resource factors: the UCP model does not propose any criteria for measuring resource factors. Once again, this "measurement" is effectively a subjective interpretation by a measurer.

It should be noted that Karner [KAR93] had indicated that the EF constants and weights were preliminary and estimated. That being said, more recent sources like [CLE06] and [CAR05] have not revisited this issue and have integrated the same values without modification into their own measurement models of UCP variants.

4.4 Inadmissible numerical operations

The formula to calculate the number of UCP is as follows:

$$\mathbf{UCP = UUCP * TCF * EF}$$

Albrecht [ALB79] had initially characterized FP as being "dimensionless coefficients". As UCP is an adaptation of FP for use cases, some have taken for granted that the UCP are numbers without units: that is, it is assumed that one arrives at the UCP number by multiplying (apparently) dimensionless constants with (again, apparently) dimensionless weights with values on ordinal or nominal scales. This issue has been documented in [ABR94] and [ABR96].

More specifically, for UCP, the final UUCP value is calculated by multiplying the number of use cases and actors of each type of complexity by their weights. In doing so, the ordinal-scale values (categories of complexity) are transformed into interval-scale values. These are than multiplied, resulting in a value on a ratio scale, another algebraically inadmissible mathematical operation.

This analysis applies equally to TCF and EF factor calculations, which, in addition to the transformations of ordinal-type scale into interval-type scale (confounding numerical values with the ordering of categories by numerical label), also introduce multiplications with ratio-scale constants. [ABR94] described this same error in the FP measurement method, the origin of the EF and TCF calculations.

It has further been pointed out, in [ABR94, ABRA96], that the interaction between technical factors would have been better accounted for by multiplying the factors together rather than adding them. This analysis also applies for the EF factors in this case. In summary, TCF has inherited most of the defects of FP – see Appendix, and in some cases has compounded them by adding additional inadmissible mathematical operations.

4 Observations and conclusions

To summarize, this investigation of the measurement principles behind the UCP method has raised the following measurement issues:

• Use cases can be described at various levels of granularity – the UCP method does not take this into account and does not describe any means to ensure the consistency of granularity from one use case to another. The impact is the following: quantities as the outcome of UCP are not necessarily comparable and a poor basis for benchmarking and estimation.

• The UCP measurement method measures several entities (actors, use cases, requirements, etc.) and attributes (complexity, difficulty, etc.). Combining, as it does,

too many concepts at once makes it a challenge to figure out what the end-result is from a measurement perspective. The impact of this is that the end-result is of an unknown and unspecified entity type; that is, we do not know what has been measured.

• The UCP measurement method is based on the constants and weights of Albrecht's FP Value Adjustment Factors without supporting justification.

• The evaluation of attributes is performed by a measurer without criteria or a guide to interpretation – a 5 for one measurer could be a 2 for another. Therefore, repeatability as well as reproducibility could be very poor.

• UCP method calculations are based on several algebraically inadmissible scale type transformations. It is unfortunate that this has not yet been challenged, either by UCP users or the designers of subsequent UCP variants.

The measurement of functional size using use cases (therefore, very early in the development life cycle) represents an interesting opportunity for an industry which is increasingly using RUP and Agile use-case-oriented methodologies. Unfortunately, the UCP method suggested by Karner appears fundamentally defective: by adopting the basic structure of FP, UCP has – from a measurement perspective – inherited most of its structural defects.

There have been claims for the relevance of UCP for estimation purposes, but with very limited empirical support. Even when empirical support is claimed, it is based either on UCP variants and anecdotal support or on undocumented evidence.

The idea of measuring size and estimating effort with use cases remains attractive, and Karner's method represents a first step towards that objective. That said, the lessons learned through this investigation of the UCP design clearly indicate that:

- On the one hand, significant improvements to UCP design are required to build a stronger foundation for valuable measurement;
- On the other hand, much more robust and documented empirical evidence is required to demonstrate the usefulness of UCP for estimation purposes.

Appendix: FPA use of scales in the measurement of Data Files (EIF and ILF)

Step	Entities	Operations	Scale: From	Scale: To	Math. Validity	Implicit transformation
Count elements of data and records	Data	Count	Absolute	Absolute	Yes	No
	Record	Count	Absolute	Absolute	Yes	No
Execute data algorithm	Data	Identify range	Absolute	Ordinal	Yes	Yes, and loss of information
	Record	Identify range	Absolute	Ordinal	Yes	Yes, and loss of information
	Function of ranges of (data,record)	Position in matrix	Ordinal	Nominal	Yes	Yes, and loss of information
	Function of position in matrix	Name and order	Nominal	Ordinal	No	Yes, and loss of information
	Function of perceived values	Assign weights	Ordinal	Ratio	No	Yes, and loss of information
Add all points	Weights of internal files	Add	Ratio	Ratio	Yes	No
	Weights of external files	Add	Ratio	Ratio	Yes	No
	Weights: internal + external	Add	Ratio	Ratio	Yes	No

References

[ALB79] Albrecht, A.J, "Measuring Application Development Productivity," Proceedings of IBM Application Development Joint SHARE/GUIDE Symposium, Monterey, CA, USA, 1979, pp. 83-92.

[AND01] Anda, B., Dreiem, H., Sjøberg, D.I.K. and Jørgensen, M. "Estimating Software Development Effort Based on Use Cases – Experiences from Industry," 4th International Conference on the Unified Modeling Language (UML2001), October 1-5, 2001, Toronto, Canada, Springer Verlag, pp. 487-502.

[ABR94] Abran, Alain; Robillard, P.N., "Function Points: A Study of their Measurement Processes and Scale transformation," Journal of Systems and Software, vol 65, 1994, pp. 171-184.

[ABR96] Abran, Alain; Robillard, P.N., "Function Points Analysis: An Empirical Study of Its Measurement Processes," IEEE Transactions on Software Engineering, Vol. 22, no. 12, December 1996, pp. 895-910,

[ABRA03] Abran, A., Desharnais, JM., Oligny, S., St-Pierre, D., Symons, C., COSMIC-FFP version 2.2 – The COSMIC Implementation Guide to ISO/IEC 19761, École de technologie supérieure, Université du Québec, Montréal, Canada, 2003 url: www.gelog.etsmtl.ca/cosmic-ffp

[ABR04] Abran, A., Lopez, M., Habra, N., "An Analysis of the McCabe Cyclomatic Complexity Number," 14th International Symposium on Software Measurement IWSM-Metrikon, 2004, Magdeburg, Germany, pp. 391-405.

[ALQ05] Al Qutaish, R. E., Abran, A., "An Analysis of the Design and Definitions of Halstead's Metrics," 15th International Symposium on Software Measurement (IWSM), 2005, Montréal, Canada.

[CAR05] Carroll, Ed, "Software Estimating Based on Use Case Points," The Cursor, Software Association of Oregon, Website: http://www.sao.org/newsletter

[CLE06] Clemmons, Roy K., "Project Estimation With Use Case Points," Crosstalk, February 2006, pp. 18-22

[COC97] Cockburn, Alistair, "Structuring Use Cases with Goals," Journal of Object-Oriented Programming, September-October, 1997, and November-December, 1997.

[HAB06] Habra, N., Abran, A., Sellami, A., "A Framework for Software Measurement Design," submitted for publication.

[ISO19761] ISO/IEC19761:2003, Software Engineering – COSMIC-FFP – A Functional Size Measurement Method, International Organization for Standardisation – ISO, Geneva, 2003.

[ISO20926] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, International Organization for Standardisation – ISO, Geneva, 2003.

[JAC87] Jacobson, Ivar, "Object Oriented Development in an Industrial Environment," OOPSLA '87 Proceedings, October 4-8, 1987, pp. 183-191.

[JAC97] Jacquet, Jean-Philippe; Abran, Alain, “From Software Metrics to Software Measurement Methods: A Process Model,” Third International Symposium on Software Engineering Standards, ISESS ’97, Walnut Creek, CA. June 2-6, 1997.

[KAR93] Karner, Gustav, “Resource Estimation for Objectory Projects,” Objective Systems SF AB, 1993.

[MOH05] Mohagheghi, Parastoo, “Effort Estimation of Use Cases for Incremental Large-Scale Software Development,” IEEE International Conference on Software Engineering – ICSE ’05, May 15-21, 2005, St.Louis, MI, USA, pp. 303-311.

[NAG01] Nageswaran, Suresh, “Test Effort Estimation Using Use Case Points,” Quality Week 2001, San Francisco, CA, USA, June 2001.

[RIB01] Ribu, K., “Estimating Object-Oriented Software Projects with Use Cases,” 2001, University of Oslo, Oslo, Norway. Website: http://www.stud.ifi.uio.no/~kribu/oppgave.pdf

[SMI99] Smith, John, “The Estimation of Effort Based on Use Cases,” Rational Whitepaper, Rational Software, 1999.