# An Effort Estimation by UML Points in the Early Stage of Software Development

**SangEun Kim**
Department of Computer Science
Texas A&M University
College Station, TX USA

**William Lively**
Department of Computer Science
Texas A&M University
College Station, TX USA

**Dick Simmons**
Department of Computer Science
Texas A&M University
College Station, TX USA

**Abstract -** *UML-based object-oriented metrics are fully capable of software measurement. Many researchers have produced effort estimation models for software systems. The estimation effort in the early stages of software development is one of the most important problems faced by software developers and managers. UML related information can be used as an accurate source for effort estimation. In this paper, we propose an automatic software metrics analysis tool and a methodology for early stage effort estimation for software systems. Using this method, the developer/manager can analyze a software system with function point-like analysis. UML Points is a new concept, combining Use Case Points and Class Points with our own definitions to provide software system size information. Based on UML Points, we generate an effort estimation model after correlation analysis for determining the relationship between effort and UML Points.*

**Keywords:** Object-oriented metrics, correlation analysis, UML, software measurement, effort estimation.

## 1 Introduction

Three approaches are used in developing highly trustworthy software systems. The first is developing new methodologies to improve software quality. An example of this is instance object-oriented, component-based software development. New methodologies are widely used for developing software systems in both the academic and industrial arenas. The second approach is process improvement. This approach has improved software quality and reliability. The third approach is software measurement. We need accurate metrics for measuring software systems and predicting the effort required for development.

The approaches used in the first two categories affect how the software is measured. For example, object-oriented methodology generates new metrics relating to object-oriented technology. Process improvement can be enhanced through metrics of each process. In the research of software measurement in terms of effort estimation, several criticisms exist: lack of a theoretical basis, lack of desirable measurement properties, being insufficiently generalized, being too implementation technology dependent, being a subjective measurement based on expert decision and being too labor-intensive for collecting information [1][2].

It is widely recognized that Unified Modeling Language (UML) is a de facto standard to describe software systems using object-oriented concepts through visualization. UML provides a well-structured architecture and overview of a system through various diagrams representing different viewpoints of the target system. Though UML is not yet an architecture description language, by using various UML diagrams, useful information can be extracted for measuring the complexity and size of software systems. Capturing useful information from UML diagrams provides the benefit of a language-independent measurement in the upstream level of software development.

This paper presents an automatic software measurement tool based on UML diagrams, and an effort estimation model based on that measurement to improve productivity. UML Points consist of Use Case Points and Class Points from the use case diagrams and class diagrams of UML, respectively. The main contributions of this paper are the development of an automated tool to calculate UML Points from UML diagrams, introducing an effort estimation model based on UML Points through correlation analysis. This approach is proofed by theoretical and empirical validation.

We introduce the basic concepts of size measurement in terms of effort estimation. Size measurement is one of the most operative factors of the software development effort. Then, we present a proposed size measurement, UML Points, and provide validation of its usefulness and applicability. After that, we show an effort estimation model based on UML Points through statistical analysis, providing experimental results. We conclude our paper by summarizing and analyzing our results.

## 2 Upstream vs. Downstream

Estimation and prediction of software system development cost has been widely researched for several

decades. In this section, we review some basic concepts of software size measurement and effort prediction, which are the most effective factors in developing software systems on time and on budget. These foundations will affect the method of software measurement and effort estimation.

## 2.1 Early--as soon as possible

Software metrics were used as the basic foundation of prediction of effort. The traditional approaches focused on source code or expert decision-based analysis to provide accurate information for calculation. These approaches and their pros and cons are shown in Table 1[3].

[Table 1] Pros and Cons of Software Estimation

| Estimation Approaches | Pros | Cons |
|---|---|---|
| Analogy-based | - Accurate estimation<br>- Very simple to apply to similar projects<br>- Rapid estimation with detailed documentation | - Increasing unreliability<br>- Difficulties with real environment and given data |
| Work Break Structure | - Applicable to original projects<br>- Inherent local calibration<br>- Well documented process | - Highly dependent on expert's abilities and decisions |
| Function Point Analysis | - Reliable size estimation<br>- Can be applicable in the early stage of project life cycle<br>- Language and platform independent<br>- Large user base--active effort | - Manual/high labor cost<br>- Not applicable to latest software development methodology<br>- Not ideal in the requirements capture period |
| COCOMO/II | - Live effort estimation<br>- Transparent algorithm<br>- Local calibration<br>- Free implementation | - Highly dependent on size input<br>- Small data set to determine the parameter heuristics |
| ObjectMetrix | - Live commercial effort estimation<br>- Supporting modern development methodology such as OO design; interactive development<br>- Can be applicable in early stage of project life cycle | - Lack of public information<br>- Not as widely used as other methods<br>- Only commercial implementation |

Common problems with these approaches are lack of early estimation, over-dependence on expert decision, and subjective measurement of each metric. A new approach is required to overcome these existing difficulties. We move upstream in the software development process to requirement analysis and design. Currently, UML diagrams are widely used in the software development industry for requirement analysis and detail design before jumping into the coding processes.

We surveyed 47 different object-oriented metrics to identify appropriate software measurement from UML diagrams and developed a well-structured tree for the UML-based object-oriented software measurement to assist effort estimation. The results of this classification follow:

- Primitive measurements that represent a skeleton/structure of UML diagrams. These metrics help overcome a lack of desirable measurement properties and information.
- Fault-proneness measurements that predict a class's fault-proneness.
- Coupling measurements, which provide locality information among objects, classes and packages. We propose a new metric, package-level coupling. This coupling represents locality dependency between package components.
- Object-oriented software measurements, which are related to inheritance, information hiding, and complexity of scenarios.

Through this classification, we found it necessary to develop a simple approach for providing useful information for software effort estimation while maintaining accuracy. This new approach will be to provide an early estimation of effort as soon as possible during the project. Based on this estimation, the project manager can finish on schedule.

## 2.2 Classification of software size and cost estimation models

To estimate software development cost, several approaches exist. Table 2 shows one of the classification methods in literature.

[Table 2] Classification of cost estimation models [3]

| Effort & Scheduling Computation / Complexity & Size Metrics | Parametric models | Non-parametric models(Machine Learning Approaches) |
|---|---|---|
| Source Lines of Code (KSLOCs) | SLIM<br>COCOMO | Regression Trees |
| More complex elements (Dimensions) | Function Points<br>Object-Oriented Approaches | Regression Trees<br>Neural Networks<br>Analogies |

From this classification, we combine the parametric and non-parametric models to effectively estimate costs at the early stage of software development. To do this we need our own definition of class points, use case points, and UML points. In the next section, we define each in detail.

# 3 UML Points

To glean useful information early in the software development process, we focus on the following UML diagrams: requirement negotiation information between the customer and developer in a use case diagram, and detail

design information in class diagrams. The UML points approach will provide simple calculation, will be easy to implement, and will provide reasonable cost estimation in the upper stage of software development. In this section, we provide an overview of the use case points and class points approaches to provide input for the effort estimation model.

## 3.1 Use Case Points

Use case diagrams contain the functional behavior of the target system, determined during the requirement analysis phase. The Use Case Points (UCP) approach was introduced by Karner[4] as a software project effort estimation model. UCP effort estimation is an extension of existing estimation methods, such as function point analysis and MK II function points analysis. Figure 1 depicts an effort estimation main flow based on the UCP calculation steps.

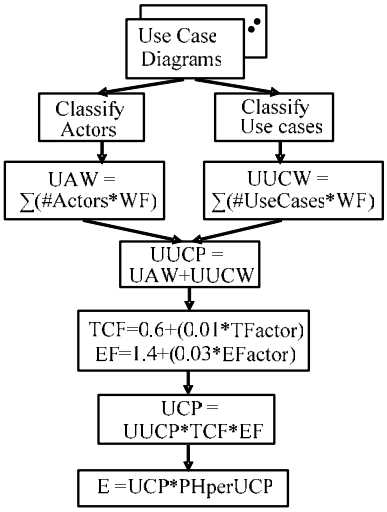


Fig. 1. The UCP effort estimation steps.

A detailed description of each step is shown in [5]. The first step is counting the number of actors and assigning weighting values based on the categorization for unadjusted actor weights (UAW). The second step is enumerating the number of use cases and calculating its weighting value by the number of transactions for unadjusted use case weights (UUCW). Step 3 is calculating unadjusted use case points (UUCP) by adding the previous two results. Step 4 is determining the technical factors for system and environmental factors for the team by given equations. In step 5, the adjusted use case points (UCP) is calculated by multiplying UUCP, technical complexity factor (TCF), and environmental factor (EF). The final step, step 6, is generating estimated effort by multiplying UCP and person-hours per UCP (PHperUCP). Table 3 shows how each factor was determined and what value was assigned at each step [5].

[Table 3] Factors and descriptions

| Factor | | Description | Weight |
|---|---|---|---|
| Actors | Simple | Program interface | 1 |
| | Average | Interactive, or protocol-driven, interface | 2 |
| | Complex | Graphical interface | 3 |
| Use Cases | Simple | 3 or fewer transactions | 5 |
| | Average | 4 to 7 transactions | 10 |
| | Complex | More than 7 transactions | 15 |
| Tech. | T1 | Distributed system | 2 |
| | T2 | Response or throughput performance objectives | 1 |
| | T3 | End-user efficiency (online) | 1 |
| | T4 | Complex internal processing | 1 |
| | T5 | Code must be reusable | 1 |
| | T6 | Easy to install | 0.5 |
| | T7 | Easy to use | 0.5 |
| | T8 | Portable | 2 |
| | T9 | Easy to change | 1 |
| | T10 | Concurrent | 1 |
| | T11 | Includes special security features | 1 |
| | T12 | Provides direct access for third parties | 1 |
| | T13 | Special user training facilities are required | 1 |
| Env. | F1 | Familiar with the Rational Unified Process | 1.5 |
| | F2 | Application experience | 0.5 |
| | F3 | Object-Oriented Experience | 1 |
| | F4 | Lead analyst capability | 0.5 |
| | F5 | Motivation | 1 |
| | F6 | Stable requirements | 2 |
| | F7 | Part-time workers | -1 |
| | F8 | Difficult programming language | -1 |

This approach, however, has weak points when applied to general software projects. UCP lacks information, only counting the number of actors and use cases. It also relies heavily on the estimating expert regarding the weighting of UAW/UUCW and the technical and environmental factors. The determined value of each of these factors will be highly dependent on the expert's opinion, and will therefore increase variance in the final results. To overcome these problems, we propose a new approach that will be easier to calculate, exclude the expert's decision, and focus more on the diagram itself. The use case diagram has much information about the early development stage's concept and the target system's dynamic viewpoint. The developer uses this diagram for communicating with the customer to decrease the conceptual gap between them, having sufficient knowledge of the target system. We newly define several concepts of use case points as follows:

- Number of actors (NOA) – The number of actors used to develop the target system.

$$NOA = \sum_{i=1}^{n} noa_i, \quad (1)$$

- Number of use cases (NOUC) – The number of use cases of the UML model. This is one of main artifacts affecting effort prediction.

$$NOUC = \sum_{i=1}^{n} nouc_i, \quad (2)$$

- Number of roles (NOR) – This shows the logical functionality between actor and use case. The detail behavior of these roles will be implemented at the next software development stage.

$$NOR = \sum_{i=1}^{n} nor_i, \quad (3)$$

- Average Number of Actors per Use Case (ANA_UC) – This concept reveals a ratio value of complexity of each use case in terms of the number of actors.

$$ANA\_UC = NOA / NOUC, \quad (4)$$

- Average Number of Roles per Use Case (ANR_UC) – This ratio value represents the complexity of use cases in terms of the number of roles.

$$ANR\_UC = NOR / NOUC, \quad (5)$$

- Usecase Points (UCP) – This definition represents the usecase points of the target system.

$$UCP = \sum (NOA + NOUC + NOR), \quad (6)$$

The ratio values ANA_UC and ANR_UC are easily calculated by using (1), (2), and (3) equations to provide a more general overview of use case points. In general, the use case diagram was used in communication between developer and customer to reduce conceptual gaps between them, so it has sufficient knowledge about the target system. We, therefore, can use the value of the use case points as an input for our effort estimation model. For instance, if the value of NOA is high then it means that the system has a great deal of interface with its environment. UCP is calculated by adding up all of the use case points as in equation (6).

## 3.2 Class Points

In object-oriented development, the class diagram has a great deal of quantification information based on the design document. It contains the structural functionality of the target system and its class hierarchy, which are the logical blocks of the developed system. The class points approach was introduced in 1998[6]. This was based on the function points analysis approach to represent the internal attributes of a software system in terms of counting.

There are three major steps to measure a target system, as shown in Figure 2. Each step consists of major activities required to gather quantification information of classes.

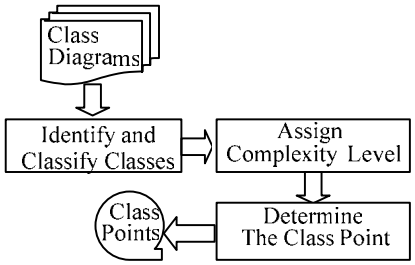


Fig. 2. Three steps of the Class Points.

The first step is to identify and classify the classes into four system types: problem domain type, human interaction type, data management type, and task management type, each in terms of the characteristics of the target system. This classification will be helpful in distinguishing between complex systems and will provide easier comparison among them. After identification and classification of classes, the class points will describe the complexity level of each class, as determined by the number of external methods, the number of services requested, and the number of attributes. Finally the class points will be calculated by applying a technical complexity factor of the target system. The technical complexity factor was determined by the degree of influence of 18 different target software system characteristics, each on a scale of 0 to 5. The detailed procedure and equations of this measurement are described in [6].

This approach, however, has weak points when applied to general software projects. CP has a lack of information problem, counting only of the number of external methods, the number of services requested, and the number of attributes. There is additional useful information affecting effort estimation of target systems such as number of inheritance/uses/realize relationships, number of parameters and number of classes. Additionally, CP uses expert decision on, for instance, component type, complexity level, TDI (Total Degree of Influence), and TCF (Technical Complexity Factor). The determined value of each factor will be highly dependent on the expert's decision, creating variance of final results.

To solve these problems, we propose a new concept of class points. This approach has similar benefits to use case points described in the previous section. We focus on the diagram itself, excluding subjective factors such as expert decision. This new definition of class points will increase understanding of a system's architectural complexity. We define it as follows:

- Number of Classes (NOC) – The number of classes used to design the target system is highly relevant to effort estimation and describes the architectural complexity of the system.

$$NOC = \sum_{i=1}^{n} noc_i, \quad (7)$$

- Number of Inheritance Relationships (NOIR) – This definition shows one of the relationship attributes between classes, specifying how many inheritance relationships were used to design the target system.

$$NOIR = \sum_{i=1}^{n} noir_i, \quad (8)$$

- Number of Use Relationships (NOUR) – This definition shows one of the relationship attributes between classes, specifying how many use relationships were used to design the target system.

$$NOUR = \sum_{i=1}^{n} nour_i, \quad (9)$$

- Number of Realize Relationships (NORR) – One of the relationship attributes between classes is how many realize relationships were used to design the target system.

$$NORR = \sum_{i=1}^{n} norr_i, \quad (10)$$

- Number of Methods (NOM) – How many methods were used to design the target system. It will be highly relevant with effort estimation and also describes the architectural complexity of system.

$$NOM = \sum_{i=1}^{n} nom_i, \quad (11)$$

- Number of Parameters (NOP) – This definition shows how many parameters were used in given methods of classes. It will be highly relevant with effort estimation and describes the architectural complexity of the system.

$$NOP = \sum_{i=1}^{n} nop_i, \quad (12)$$

- Number of Class Attributes (NOCA) – How many class attributes were used to design the target system.

$$NOCA = \sum_{i=1}^{n} noca_i, \quad (13)$$

- Number of Associations (NOASSOC) – How many associations were used to design the target system.

$$NOASS = \sum_{i=1}^{n} noass_i, \quad (14)$$

- Average Number of Methods per Class (ANM_CLS) – The ratio value of the number of methods per class in the target system.

$$ANM\_CLS = NOM / NOC, \quad (15)$$

- Average Number of Parameters per Class (ANP_CLS) – The average number of parameters per class in the target system.

$$ANP\_CLS = NOP / NOC, \quad (16)$$

- Average Number of Class Attributes per Class (ANCA_CLS) – The average number of class attributes per class in the design document.

$$ANCA\_CLS = NOCA / NOC, \quad (17)$$

- Average Number of Associations per Class (ANASSOC_CLS) – The average number of associations per class in the target system.

$$ANASS\_CLS = NOASS / NOC, \quad (18)$$

- Average Number of Relationships per Class (ANREL_CLS) – The average number of relationships per class in the target system.

$$ANREL\_CLS = \frac{(NOIR + NOUR + NORR)}{NOC}, \quad (19)$$

- Class Points (CP) – The class points of the target system.

$$CP = \sum (NOC + NOIR + NOUR + NORR + NOM + NOCA + NOASS), \quad (20)$$

Equation (7) to (12) is used to gather fundamental information from class diagrams for recognizing its structural complexity. Equations (13) to (19) are easily calculated with previous equations to provide relative information about structure complexity of class diagrams. The CP, finally, will be calculated by adding up all of the class points values as in equation (20).

These UML-based use case points and class points provide the project manager and developer a better understanding of the architectural complexity of the target system. The size measurement UML points can be used to estimate project effort. UML points are calculated by adding use case points and class points.

## 3.3 UML Points Generator

We developed an automatic tool, the UML Points Generator, to generate the UML points. The UML Points Generator's conceptual flow is as follows: 1) UML diagrams will be the input of the UML Points Generator; and 2) the UML Points Generator takes these UML diagrams and generates the UCP and CP as outputs based on given user inputs. The UML Points Generator was developed in the Java language with JBuilder 4.0, so it can run on any machine running the JVM. It has fewer than 1,200 total source lines making it a very light-weight software. The currently developed architecture of the UML Points Generator is depicted in Figure 3. It consists of three major modules: the User Input Handling & Parsing Module, the Metrics Calculate Module, and the Report Generator Module.

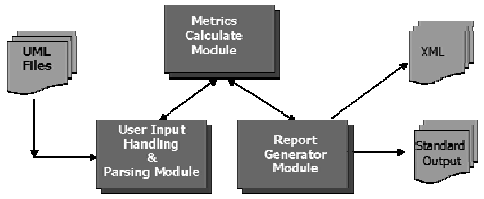

Fig. 3. Architecture of the UML Points Generator.

The User Input Handling & Parsing Module interprets and parses command-line input from users. There are two options for selecting metrics (use case points, class points, or both) and output formats (standard screen or XML output). This module has two sub-modules, lexer and parser. The lexer handles input files to generate tokens, which are the processing units of the UML Points Generator. These tokens are the input of the parser. The parser creates several vectors based on each token's kind. These vectors will be traversed to calculate each metric. The Metrics Calculate Module evaluates UML diagrams and calculates metrics using the use case points and class points. This module has mathematical calculation routines for each metric with their own algorithms. The Report Generator Module presents the metrics in standard output or XML format. XML-formatted metrics data can be used with other (e.g., statistical) tools, providing interoperability between commercial tools.

# 4 Case Study

There are several ways to utilize the proposed UML points and software effort estimation model with UML points: formalized validation with theoretical validation, experimental validation through running the pilot project, statistical analysis of the given metrics data, and application to real projects. In the research process, these validation processes were required to prove the software measurement's usefulness. We chose two validation procedure approaches, a theoretical approach to show utilization, and an empirical approach to provide experimental case studies.

## 4.1 Theoretical Approach

Several approaches have proposed theoretical principles and frameworks for software measures to provide a formal basis and foundation for their validation procedures. We followed the Briand et al. method proposed in [7]. They suggest a pragmatic approach to providing a mathematical framework to gather more practical results from huge, complex software products. They defined convenient and intuitive formalisms and properties to apply to measurement concepts such as size, length, complexity, cohesion, and coupling.

We follow their definition as a formal validation procedure to apply to our own proposed size measures. This approach was used to provide the theoretical foundation for formal software measurement validation. They defined the representation of systems and modules in relational systems. A system S consists of a pair <E, R>, where E represents the set of elements of S and R is a binary relation on E($R \subseteq E \times E$) representing the relationships between S's elements. Given a system S = <E, R>, a system m = <$E_m$, $R_m$> is a module of S if and only if $E_m \subseteq E$, $R_m \subseteq E_m \times E_m$, and $R_m \subseteq R$. The elements of a module are connected to the elements of the rest of the system by incoming and outgoing relationships. They also defined three basic size measurement properties: nonnegativity, null value, and module additivity. The first says that the size of a system S is nonnegative. The second says that the size of a system S is null if E is empty. The third property says that the size of a system S is equal to the sum of the sizes of their modules [7].

Based on their definitions and properties, we can provide our own formal validation to prove those properties in our model. The nonnegativity, null value, and module additivity properties hold for the UML points size measure. The value of the UML points is calculated by summation of the nonnegative numbers of the UCP and CP, so the nonnegativity property holds. If there are no class and use case diagrams in the system design, the UML points value is null, so the null value property is also satisfied. If a system consists of several modules, the values of UCP and CP are unchanged by system development no matter how the use case and class diagrams were used in the system.

## 4.2 Experimental Approach

To do experimental validation of the proposed model, we chose the linear regression test, which is used for developing an effort estimation model based on the 30 UML files and the proposed size metrics. We used the SPSS tool to do this work automatically. A T-test was performed to understand the correlation between the metrics. In the meantime, a number of researchers were studying object-oriented and traditional metrics, but they did not analyze the relationship between the metrics themselves. This statistical analysis helps to understand the cooperative relationship of complex metrics. Basically, we assumed no tight relationship between metrics, and needed to test the reasonableness of this hypothesis. Therefore we performed a Pearson correlation analysis of the SPSS tool. Table 4 shows the result of the correlation analysis between the metrics and the total effort. The value of the Pearson correlation can represent three different relationships: a positive (close to 1), no (close to 0), or a negative (close to -1) relationship between metrics. Through this relationship analysis, we can generate a useful assessment of the target system. In Table 4, we found that NORR and NOUC have the highest positive relationship among the metrics. Based on this statistical

analysis, there exist several tight relationships between metrics and effort model, whether negative or positive.

[Table 4] Pearson Correlation

| UML Points | Metrics | Pearson Correlation |
|---|---|---|
| CP/UCP | NORR vs NOUC | 0.770 |
| CP/UCP | NORR vs NOA | 0.755 |
| CP/CP | NOC vs NOP | 0.700 |
| CP/CP | NOM vs NOP | 0.638 |
| CP/CP | NOASS vs NOCA | 0.613 |
| CP/UCP | NOCA vs NOR | 0.612 |

# 5 Conclusions

For our contribution, UML points was proposed to measure the size of object-oriented applications developed using UML diagrams. An automatic size measurement tool, the UML points generator, was developed to provide function points like measurement from UML diagrams, especially from use case diagram and class diagram. In this paper, we propose size measurement for the UML design specification at the early design phase. To show the utilization of the size metrics, an effort estimation model was developed with the metrics parameters based on analysis of 30 UML files from a real project. This effort estimation model can be used to predict the effort of future projects. We did statistical analysis between metrics to increase understanding of the relationship among them through Pearson correlation analysis of the SPSS.

This work can be expanded to develop additional metrics extracted from other UML diagrams such as interaction diagrams and component diagrams. The current work focuses on class and use case diagrams. Further analyses are necessary to understand more useful relationships between metrics and complexity.

# 6 References


[1] Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493, June 1994.

[2] Nasib S. Gill and P.S. Grover, "Software Size Prediction Before Coding," ACM SIGSOFT Software Engineering Notes, Vol. 29, No. 5, pp. 1-4, September 2004.

[3] K. Kavoussanakis and Terry Sloan, "UKHEC Report on Software Estimation," http://www.ukhec.ac.uk/publications/reports/estimation.pdf, December 2001.

[4] Parastoo Mohagheghi, Bente Anda, and Reidar Conradi, "Effort Estimation of Use Cases for Incremental large-Scale Software Development," Proceedings of the International Conference on Software Engineering (ICSE'05), pp. 303-311, May 15-21 2005.

[5] Shinji Kusumoto, Fumikazu Matukawa, Katsuro Inoue, Shigeo Hanabusa, and Yuusuke Maegawa, "Estimating Effort by Use Case Points: Method, Tool and Case Study," Proceedings of the 10th International Symposium on Software Metrics (METRICS'04), pp. 292-299, September 14-16, 2004.

[6] Gennaro Costagliola and Genoveffa Tortora, "Class Point: An Approach for the Size Estimation of Object-Oriented Systems," *IEEE Transactions on Software Engineering*, Vol. 31, No. 1, pp. 52-74, Jan. 2005.

[7] Lionel Briand, Sandro Morasca, and Victor R. Basili, "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, pp. 68-86, January 1996.