Effect of Different UML Diagrams to Evaluate the Size Metric for Different Software Projects

Preety Verma Dhaka[1]

[1] Jayoti Vidyapeeth Womenas University, Jaipur

Received: 8 December 2014 Accepted: 1 January 2015 Published: 15 January 2015

Abstract

In Software Engineering, an important activity is measuring of the Software in different ways. For Measuring the Software, appropriate metrics are needed. Using Software metrics we are able to attain the various qualitative and quantitative aspects of Software. To measure the Software in terms of quality, size, efforts, efficiency, and reliability, performance etc. we have different metrics available in Software Engineering and it has been an area of interest for the various researchers. Measures of specific attributes of the process, project and product are used to compute Software metrics. This work proposes a similar approach of measuring software using various UML diagrams and applied Software size metric to evaluate the size of the Software. This paper discusses the proposed approach using two different case studies and their source codes. This paper discusses the different results obtained using different perspectives of the Software size metric measurements and maintainability of the Software.

Index terms— software metrics, size metric, uml diagrams, use cases, cocomo, maintainability

1 Introduction

he objectives of this analysis square measure to create associate empirical analysis of computer code size metrics supported UML with the assistance of 2 case studies then calculate that empirical information consisting of actual values and therefore thereby showing that however the computer code size metrics are going to be derived from associate UML model via category Diagrams and the below listed interaction diagrams. 1. Activity Diagrams 2. State chart Diagrams

2 Component Diagrams 4. Collaboration Diagrams

For winding up this analysis, 2 real case studies particularly (i) Virtual category space and (ii) information Secrecy System are going to be taken for sensible analysis. The UML modeling of those systems are going to be done and therefore the computer code size metrics of those systems are going to be evaluated supported the UML models, the non-functional techniques (LOC, FP, and COCOMO-II). The metrics are going to be such UML extension mechanism then are going to be calculated with the assistance of a tool. The calculable values are going to be compared with the particular computer code. Thus, the aim of our analysis is to judge the empirical worth sets of UML models and thereby, showing the utilization of assorted size metrics and validate their extraction procedure from UML style with the assistance of interaction diagrams. II.

3 Existing Work

Many scientists and researchers have studied the package metrics supported UML models. And therefore have given their large contributions to the sector of analysis within the laptop sciences .A lot of labor has been done until date within the space of analysis whereas considering package metrics associated with UML style.

In their paper Tong Yi et al. [7] analyzed and compared some typical metrics for UML category diagrams from totally different viewpoints , differing types of relationships, differing types of metric values, complexity, and fragrance theoretical and empirical validation. They need tried to investigate the present well-liked metrics for UML category diagrams each on paper and by experimentation from many totally different viewpoints. The analysis shows that the majority current metrics have their shortcomings whereas being effective or economical for a few special characteristics of the system.

Li Wei dynasty et al. [8] has conferred AN empirical study of OO metrics in 2 unvaried methodes: the short-cycled agile method and therefore the longcycled framework evolution process. They need found that OO metrics area unit effective in predicting style efforts and supply lines of code superimposed, changed, and deleted within the short-cycled agile method and ineffective in predicting identical aspects within the long-cycled framework method. This leads them to believe that OO metrics' prophetic capability is proscribed to the planning and implementation changes throughout the event iterations, not the long evolution of a longtime system in numerous releases.

Mitchell et al. [9] conferred a footing paper outlining a programmed of analysis supported the quantification of run-time parts of Java programs. Especially, we tend to adapt 2 common object-oriented metrics, coupling and cohesion, so they'll be applied at run-time. The results conferred during this paper area unit of a preliminary nature, and don't offer a excusable basis for generalization. However, she believed that they are doing offer a sign that the analysis of package

4 (C)

metrics at run-time will offer a motivating measurement of a program.

Through their paper Christodoulakis et al. [10] have derived the results on metrics employed in object orientating environments. Their survey includes a tiny low set of the foremost renowned and ordinarily applied ancient package metrics that can be applied to objectoriented programming and a group of object-oriented metrics (i.e. those designed specifically for objectoriented programming). These metrics were evaluated mistreatment existing meta-metrics further as metametrics derived from our studies, primarily based on the practitioner's purpose of read, and accenting pertinence in 3 totally different programming environments: Object Pascal, C++ and Java.

In this paper M. Das et al. [11] have expressed that Component-Based package Engineering (CBSE) has shown important prospects in speedy production of huge package systems with increased quality, and stress on decomposition of the built systems into purposeful or logical elements with well-defined interfaces used for communication across the elements. During this paper, a series of metrics projected by numerous researchers are analyzed, evaluated and benchmarked mistreatment many large-scale publically. A scientific analysis of the values for numerous metrics has been administrated and several other key inferences are drawn from them. Varieties of helpful conclusions are drawn from numerous metrics evaluations, which embrace inferences on quality, reusability, testability, modularity and stability of the underlying elements. The inferences area unit argued to be useful for CBSE-based package development, integration and maintenance.

Jamali [12] has expressed the central role that package development plays within the delivery and application of data technology, managers' area unit progressively that specializes in method improvement within the package development space. The main target on method improvement has inflated the demand for package measures, or metrics with that to manage the method. The necessity for such metrics is especially acute once a corporation is adopting a brand new technology that established practices have however to be developed. He has self-addressed these wants through the event and implementation of a collection of metrics for OO style.

Shaik Amjan, et al. [13] has conferred the getable and new package metrics helpful within the totally different part of the Object-Oriented package Development Life Cycle. Metrics area unit utilized by the package trade to itemize the development; operation and maintenance of package. They have conferred metrics for Object-oriented package systems. A mechanism is provided for comparison measures, that examine identical ideas in numerous ways that, and facilitating a lot of rigorous decision-making, relating to the reason of latest measures and therefore the choice of existing measures for a selected goal of menstruation.

5 III.

6 Proposed Methodology

This work is the UML diagrams to calculate the dimensions metrics. It's been found that existing researches specialize in the utilization CASE to be the UML diagram for analysis of the dimensions metric. Inclusion of the opposite UML diagrams in analysis method of the dimensions metric has been projected during this analysis. The whole work is being carried in following steps: 1. Taken 2 case studies and their ASCII text file because the input of this work 2. UML diagrams of the case studies has been drawn and enclosed for the evaluations of the dimensions metric 3. Meta Mil computer code is getting used to come up with the XMI document for analysis of the dimensions metric 4. Generated XMI file is employed with the Mount Rushmore State Metric tool for analysis of the metric values. 5. For comparison purpose 2 alternative size metric techniques are used i.e. Lines of Codes and performance purpose Analysis 6. After analysis of the metrics varied strategies, a chart of the all the metric values are going to be generated to indicate the results.

The proposed work shall be carried out using the following structural diagram: Unified Modeling Language (UML) is well-liked these days for capturing necessities and for describing the design of a software-intensive system. One among the UML constructs may be a use case, that diagrammatically depicts the manner within which a user can act with the system to perform one operate or one category of functions. 3 aspects of use cases are often useful as inputs to a size estimate: the quantity of use cases, the quantity of actors concerned in every use case, and therefore the variety of situations. AN actor may be a person or system that interacts with the system beneath consideration; usually, there's one actor per use case, however typically there square measure additional. A state of affairs may be a potential outcome from exploitation the software; the quantity of situations will vary from one to thousands or millions, counting on the system and its quality. This technique may be helpful once the dimensions estimate is needed once a UML specification is completed. It may also be used as a insure of another method; if the answers from each strategies square measure similar, the analysts could have additional confidence within the result.

IV.

7 Metrics of Sdmetric

Metric NumAttr: the amount of attributes within the category. The metric counts all properties no matter their kind (data kind, category or interface), visibility, quality (read solely or not), and owner scope (class-scope, i.e. static, or instance attribute). Not counted square measure genetic properties, associate degreed properties that square measure members of an association, i.e., that represents passable association ends. Metric NumOps: the amount of operations during a category. Includes all operations within the category that square measure expressly modeled (overriding operations, constructors, destructors), no matter their visibility, owner scope (class-scope, i.e., static), or whether or not they square measure abstract or not. Genetic operations don't seem to be counted. Metric NumPubOps: the amount of public operations during a category. This can be same as metric NumOps, however solely counts operations with public visibility. It measures the dimensions of the category in terms of its public interface. Metric Setters: the amount of operations with a reputation beginning with 'set'. Note that this metric doesn't perpetually yield correct results. As an example, associate degree operation settle Account are going to be counted as setter methodology. Metric Getters: the amount of operations with a reputation beginning with 'get', 'is', or 'has'. Note that this metric doesn't perpetually yield correct results. As an example, associate degree operation isolate Node are going to be counted as getter methodology.

Metric Nesting: The nesting level of the category (for inner classes). Measures however deeply a category is nested at intervals different categories. Categories not outlined within the context of another category have nesting level zero, their inner categories have nesting level one, etc. Nesting levels deeper than one square measure unusual; associate degree excessive nesting structure is troublesome to know, and may be revised.

Metric IFImpl: the amount of interfaces the category implements. This solely counts direct interface realization links from the category to the interface. as an example, if a category C implements associate degree interface I, that extends another interfaces, solely interface I'll be counted, however not the interfaces that I extends (even although category c implements those interfaces, too). Metric NumDesc: the amount of descendents of the category (UML Generalization). This counts the amount of youngsters of the category, their kids, and so on. Metric NumAnc: the amount of ancestors of the category. This counts the amount of fogeys of the category, their oldsters, and so on. If multiple inheritances don't seem to be used, the metric yields constant values as telegraphic signal.

Metric DIT: The depth of the category within the inheritance hierarchy. This can be calculated because the longest path from the category to the basis of the inheritance tree. The telegraphic signal for a category that has no oldsters is zero. A class with high telegraphic signal inherits from several categories and so harder to know. Also, categories with high telegraphic signal might not be correct specializations of all of their ascendant categories.

Metric CLD: category to leaf depth. This can be the longest path from the category to a leaf node within the inheritance hierarchy below the category.

Metric OpsInh: the amount of genetic operations. An outsized variety of kid categories could indicate particle of the parent category.

The amount of descendents of the category UML Counts the amount of youngsters of the category, their variety of ancestors of the category i.e. parents of the category, their parents, and so on. If multiple inheritances don't seem to be used, the metric yields constant values because the depth of the category within the inheritance this can be calculated because the longest path from the basis of the inheritance tree.

The telegraphic signal for a category that has no oldsters is zero. Classes with from several categories and so is harder to know. Also, categories with high telegraphic signal might not be correct specializations of sophistication to leaf depth. This can be calculated because the ad of metric NumOps seized all ascendant categories of the category.

8 a) Lines of Codes

This methodology tries to assess the seemingly variety of lines of code within the finished merchandise. Clearly, associate actual count typically created only the merchandise is complete; lines of code area unit often thought-about to be inappropriate for size estimates early within the project life cycle. However, since several of the size-estimation strategies specific size in terms of lines of code, we will contemplate lines of code as a separate methodology in this it expresses the dimensions of a system in an exceedingly explicit method.

9 b) Function Point Analysis

Function points were developed by Albrecht (1979) at IBM as the simplest way to live the quantity of practicality in an exceedingly system. They're derived from the wants. In contrast to lines of code, that capture the dimensions of associate actual product, operate points don't relate to one thing physical however, rather, to one thing logical which will be assessed quantitatively.

10 IFPUG FPA:

Formal methodology to live size of business applications. It introduces complexness issue for size outlined as operate of input, output, query, external input data and internal logical file. All elements area unit rated as Low, Average or High After the elements are classified together of the 5 major elements (EI's, EO's, EQ's, ILF's or EIF's), a ranking of low, average or high is allotted. For transactions (EI's, EO's, EQ's) the ranking relies upon the variety of files updated or documented (FTR's) and also the number of knowledge part sorts (DET's). For each ILF's and EIF's files the ranking relies upon record part sorts (RET's) and information part sorts (DET's). A record part sort could be a user recognizable subgroup of knowledge parts among associate ILF or EIF. A knowledge part sort could be a distinctive user recognizable, non algorithmic, field.

Each of the subsequent tables assists within the ranking method (the numerical rating is in parentheses). As an example, associate EI that references or updates a pair of File sorts documented (FTR's) and has seven information parts would be allotted a ranking of average and associated rating of four. Wherever FTR's area unit the combined variety of Internal Logical Files (ILF's) documented or updated and External Interface Files documented. Like all components, EQ's are rated and scored. Basically, an EQ is rated (Low, Average or High) like an EO, but assigned a value like and EI. The rating is based upon the total number of unique (combined unique input and out sides) data elements (DET's) and the file types referenced (FTR's) (combined unique input and output sides). If the same FTR is used on the input and output side, then it is counted only one time. If the same DET is used on the input and output side, then it is only counted one time.

For both ILF's and EIF's the number of record element types and the number of data elements types are used to determine a ranking of low, average or high. A Record Element Type is a user recognizable subgroup of data elements within an ILF or EIF. A Data Element Type (DET) is a unique user recognizable, non recursive field on an ILF or EIF.

Table **??** The counts for every level of complexness for every variety of part may be entered into a table like the subsequent one. Every count is increased by the numerical rating shown to work out the rated price. The totals are then summed across the table, giving a complete price for every variety of part. These totals are then summed all the way down to reach the overall range of Unadjusted perform Points.

The value adjustment issue (VAF) relies on fourteen general system characteristics (GSC's) that rate the final practicality of the appliance being counted. Every characteristic has associated descriptions that facilitate confirm the degrees of influence of the characteristics. The degrees of influence vary on a scale of zero to 5, from no influence to robust influence. The IFPUG investigating Practices Manual provides elaborated analysis criteria for every of the GSC'S, the table below is meant to produce a summary of every GSC. Rate every issue (Fi, i=1 to14) on a scale of zero to 5: Are there distributed processing functions? F5. Will the system run in an existing, heavily utilized operational environment? F6. Does the system require on-line data entry? F7. Does the on-line data entry require the input transaction to be built over multiple screens or operations? F8. Are the master files updated on-line? F9. Are the inputs, outputs, files or inquiries complex? F10. Is the internal processing complex? F11. Is the code designed to be reusable? F12. Are conversion and installation included in the design? F13. Is the system designed for multiple installations in different organizations? F14. Is the application designed to facilitate change and ease of use by the user?

Once all the fourteen GSC's are answered, they must be tabulated victimization the IFPUG price Adjustment Equation (VAF) –14 VAF = 0.65 + [(Ci) / 100] .i = is from one to fourteen representing every GSC.

Where: Ci = degree of influence for every General System Characteristic

The final operate purpose Count is obtained by multiplying the VAF times the Unadjusted operate purpose (UAF).

11 FP = UAF * VAF

Summary of advantages of operate purpose Analysis Function Points may be accustomed size software system applications accurately. Filler is a vital element in decisive productivity (outputs/inputs).

They can be counted by totally different folks, at totally different times, to get a similar live at intervals an affordable margin of error.

Function Points are simply understood by the non technical user. This helps communicate filler data to a user or client.

Function Points may be accustomed confirm whether or not a tool, a language, associate surroundings, is additional productive in comparison with others.

12 c) Cocomo-Ii

The COCOMO II model makes its estimates of needed effort (measured in Person-Months -PM) based mostly totally on your estimate of the software system project's size (as measured in thousands of SLOC, KSLOC):Effort = 2.94 * EAF * (KSLOC) E ... (**3**)

Where EAF is that the Effort Adjustment issue derived from the price Drivers. E Is a disciple derived from the 5 Scale Drivers. As associate example, a project with all Nominal value Drivers associated Scale Drivers would have an EAF of one.00 and exponent, E, of 1.0997. presumptuous that the project is projected to accommodates eight,000 supply lines of code, COCOMO II estimates that twenty eight.9 Person Months of effort is needed to finish it: Effort = a pair of.94 * (1.0) * (8)1.0997 = 28.9 Person-Months

13 d) MAINTAINABILITY

In engineering, maintainability is that the ease with that a product may be maintained so as to:

? isolate defects or their cause,

? correct defects or their cause, ? repair or replace faulty or worn-out elements while not having to switch still operating components, ? prevent surprising breakdowns, ? maximize a product's helpful life, ? maximize potency, dependableness, and safety, ? meet new needs, ? make future maintenance easier, or ? Cope with modified surroundings.

In some cases, maintainability involves a system of continuous improvement -learning from the past so as to boost the flexibility to take care of systems, or improve dependableness of systems supported maintenance expertise.

Software maintenance prices result from modifying your application to either support new use cases or update existing ones, at the side of the continual bug fixing when readying. The maximum amount as 70-80% of the entire possession value (TCO) of the software system may be attributed to maintenance prices alone! Software maintenance activities may be classified as:

? Corrective maintenance -prices thanks to modifying software system to correct problems discovered when initial readying (generally two hundredth of software system maintenance costs) ? Adaptive maintenance -prices thanks to modifying a software system resolution to permit it to stay effective in a very ever-changing business surroundings (25% of software system maintenance costs) ? Perfective maintenance -prices thanks to up or enhancing a software system resolution to boost overall performance (generally five-hitter of software system maintenance costs) ? Enhancements-prices thanks to continued innovations (generally five hundredth or additional of software system maintenance costs) ? Since maintenance prices eclipse alternative software system engineering activities by great deal, it's imperative to answer the subsequent question:

Measuring software system maintainability is non-trivial as there's no single metric to state if one application is additional rectifiable than the opposite associated there's no single tool which will analyze your code repository and supply you with a correct answer either. There's no substitute for a personality's reviewer, however even humans can't analyze the complete code repositories to grant a definitive answer. Some quantity of automation is critical.

So, however are you able to live the maintainability of your application? To answer this question let's dissect the definition of maintainability additional. Imagine you have got access to the ASCII text file of 2 applications -A and B. Let's say you furthermore may have the super human ability to match each of them in a very little span of your time. Are you able to tell, albeit subjectively, whether or not you think that one is additional rectifiable than the other? What will the adjective rectifiable imply for you once creating this comparison -suppose this for a second before we have a tendency to move?

Most software system engineers would think about some combination of testability, perceive ability and modifiability of code, as measures of maintainability. Another facet that's equally vital is that the ability to grasp the need, the "what" that's enforced by the code, the "how". These core aspects may be lessened additional, to achieve additional insight into the maintainability of the application: 1) Testability -the presence of an efficient takes a look at harness; what proportion of the applying is being tested, the categories of tests (unit, integration, situation etc.,) and therefore the quality of the take a look at case themselves?

2) Understandability -the readability of the code; are naming conventions followed? Is it self-descriptive and/or well commented? Are things (e.g., classes) doing just one factor or several things at once? Are the ways extremely long or short and might their intent be understood in a very single pass of reading or will it take an honest deal of screen staring and whiteboard analysis? 3) Modifiability -structural and style simplicity. 4) Requirement to implementation mapping and contrariwise: This data is preponderant for maintenance efforts and it should or might not exist for the applying into consideration, forcing you to reverse engineer the code and fathom the 'what' yourself.

Those are the four major dimensions on that one will measure maintainability. Every of the aspects will (and is) lessened additional for an additional granular comparison. These might or might not be the precise same ones that you simply thought of; however there'll be a good deal of overlap. Also, not each criterion is equally vital. For a few groups, testability might trump structural/design simplicity. That is, they'll care lots additional regarding the presence of take a look at cases (depth and breadth) than deep inheritance trees or a rather additional tightly coupled style. It's therefore important to understand that dimension of maintainability is additional important for your maintenance team once menstruation the standard of your application and perform the reviews and refactoring with those in mind.

The table below, towards the top of the article, shows a close breakdown of the on top of dimensions of maintainability and elaborates on their connectedness to menstruation the standard of the ASCII text file [2]: Correlation with quality: what proportion will the metric relate with our notion of software system quality? It implies that almost all programs with the same price of the metric can possess the same level of quality. Importance: however vital is that the metric and are low or high values desirable once menstruation them? The scales, in declivitous order of priority are: very vital, vital and sensible to possess Feasibility of automatic evaluation: are things absolutely or partly automatic and what types of metrics are obtainable? Ease of automatic evaluation: just in case of automation however simple is it to cipher the metric? Will it involve mammoth effort to line up or will or not it's plug-and-play or will it has to be developed from scratch. Completeness of automatic evaluation: will the automation utterly capture the metric price or is it inconclusive, requiring manual intervention? Do we have a tendency to ought to verify things manually or will we directly deem the metric reportable by the tool? Units: units/measures accustomed quantify the metric. They provide an extremely effective structure at intervals that you'll lay out choices and investigate the doable outcomes of selecting those choices. They additionally assist you to create a balanced image of the risks and rewards related to every doable course of action.

14 f) Drawing a Decision Tree

You start a choice Tree with a choice that you simply ought to create. Draw a tiny low sq. to represent this towards the left of an outsized piece of paper.

From this box put off lines towards the correct for every doable resolution, and write that resolution on the road. Keep the lines apart as way as doable so you'll expand your thoughts.

At the top of every line, think about the results. If the results of taking that call are unsure, draw a tiny low circle. If the result's another call that you simply ought to create, draw another sq.. Squares represent choices, and circles represent unsure outcomes. Write the choice or issue on top of the sq. or circle. If you have got completed the answer at the top of the road, simply leave it blank.

Starting from the new node on your diagram, put off lines representing the decisions you want to choose. From the circles draw lines representing doable outcomes. Once more create a short note on the road expression what it suggests that. Keep it up doing this till you have got drawn out as several of the doable outcomes and choices as you'll see leading on from the first choices.

Once you have got done this, review your tree. Challenge every sq. and circle to visualize if there are any solutions or outcomes you have got not thought of. If there are, draw them in. If necessary, draft your tree if components of it are too full or untidy. You ought to currently have an honest understanding of the doable outcomes of your choices.

V.

15 Result and Discussion

Results of the Proposed UML Diagram Based Metric Calculation & Count of Operations in Actual Software. These are number of operations required in the complete package and are an indicator of the number of functions required in the project. This value is a measure of the work done and found to be accurate for both the case studies. purpose analysis (FPA) are applied to live the computer code size metrics. From the results obtained from the output of American state Metric Tool, LOC and FPA, it's found that the results obtained from the inclusion of the various UML diagrams and most correct and matches with the particular computer code ASCII text file. [1] [2]

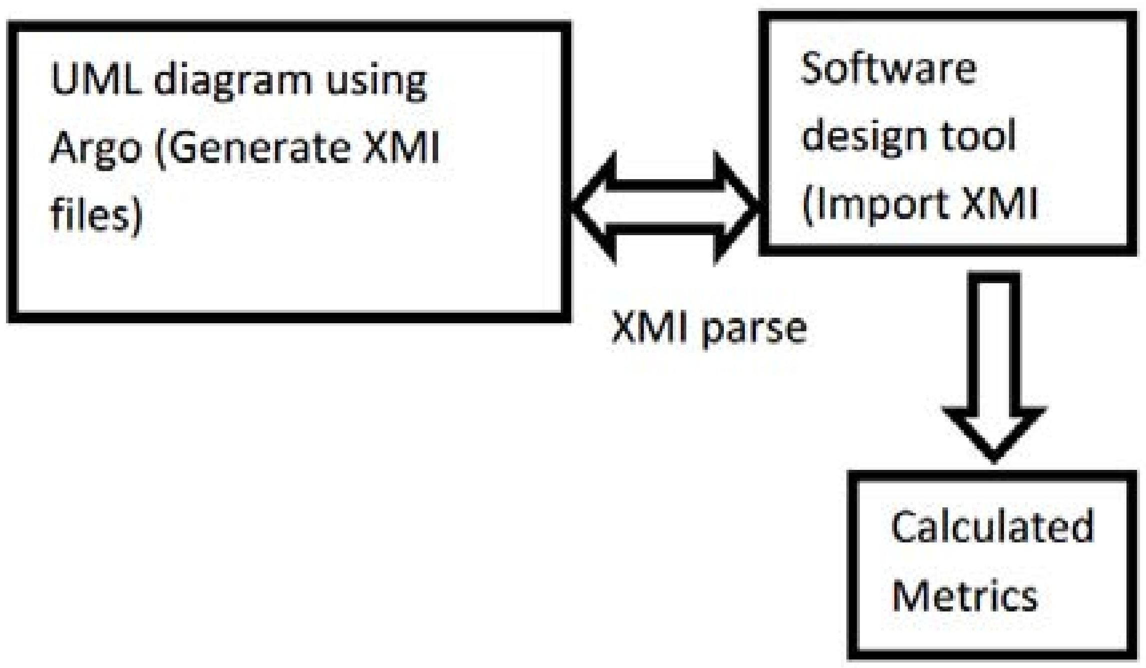
[1] © 2015 Global Journals Inc. (US)

[2] © 2015 Global Journals Inc. (US) 1

1 Open Association of Research Society, USA

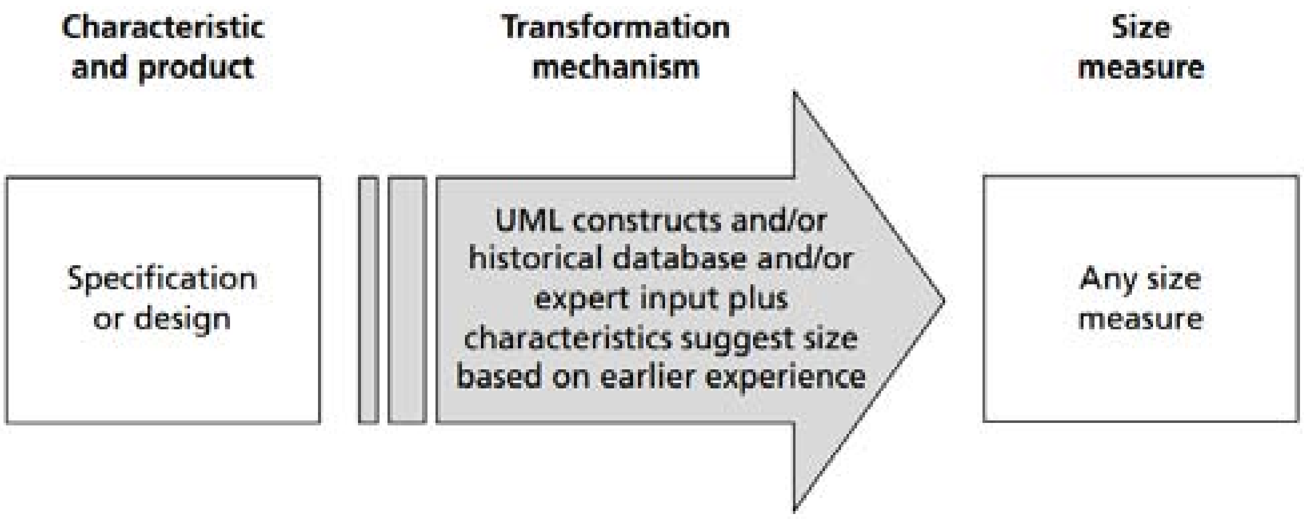
Research | Diversity | Ethics

Figure 1: Figure 1 :



UML diagram using Argo (Generate XMI files)
XMI parse
Software design tool (Import XMI
Calculated Metrics

Figure 2: Global


Characteristic and product
Transformation mechanism
Size measure
Specification or design
UML constructs and/or historical database and/or expert input plus characteristics suggest size based on earlier experience
Any size measure

2

Figure 3: Figure 2 :

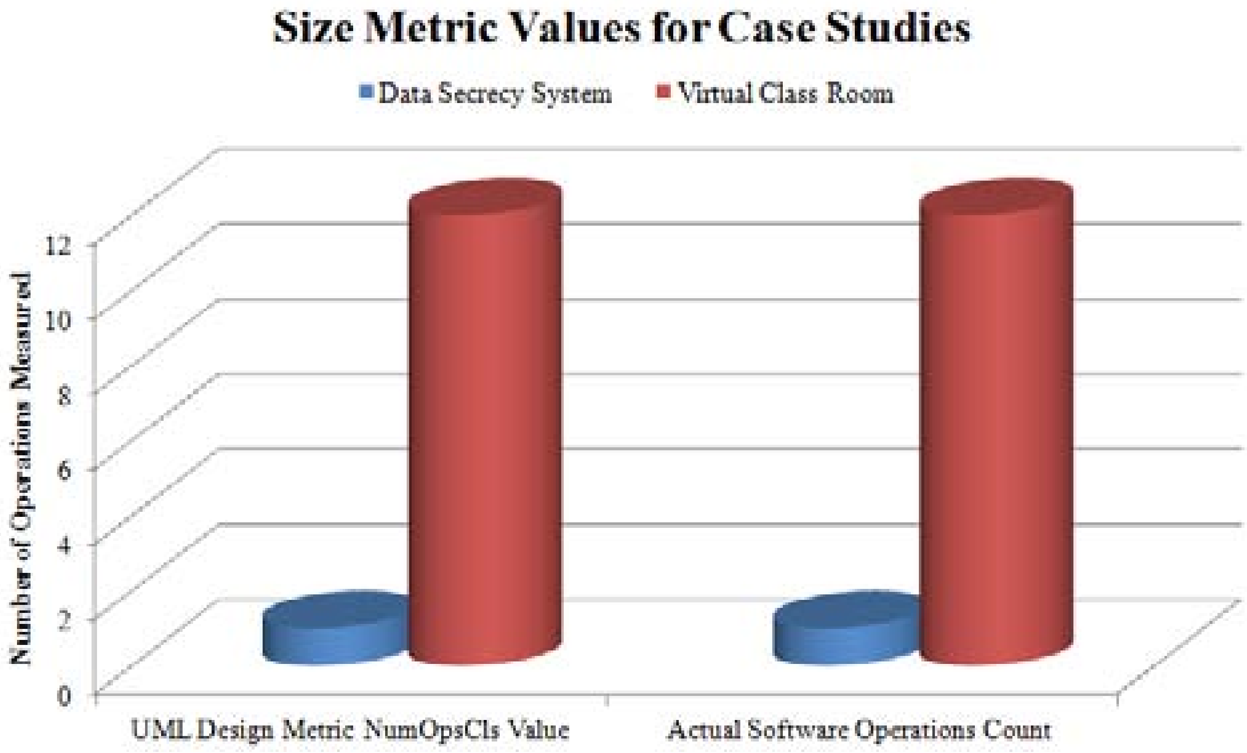


Figure 4:

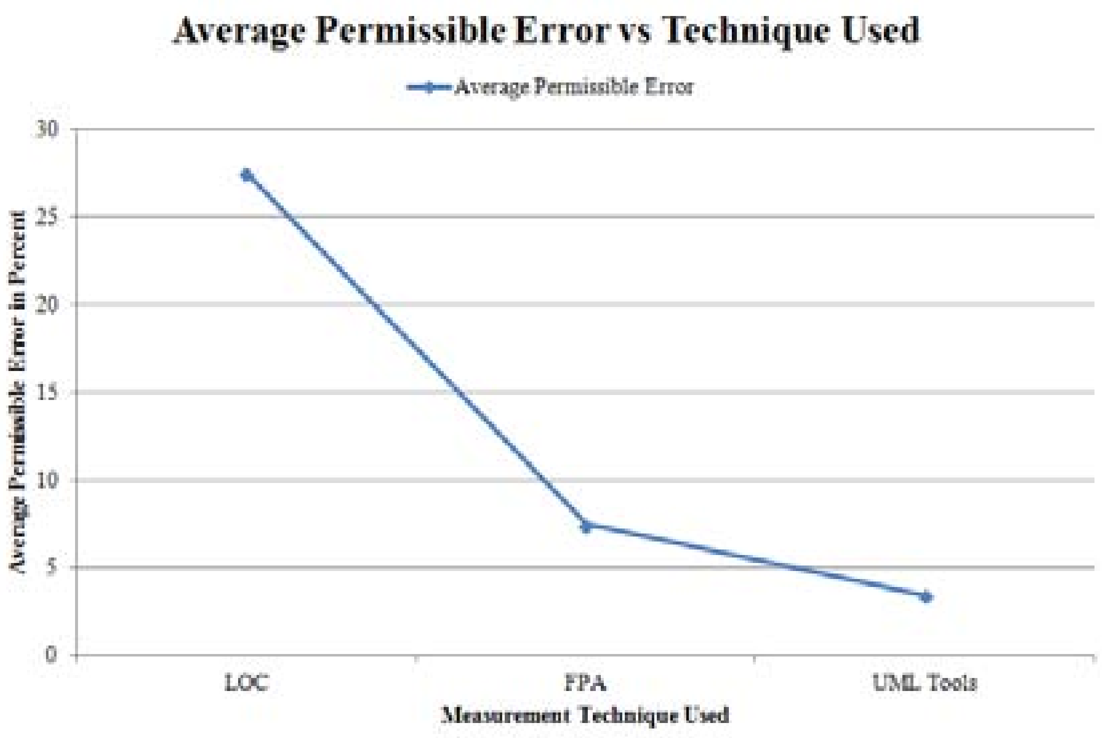


Figure 5: 4

Year 2015

Figure 6: T

1

FTR's	1-4	DATA ELEMENTS 5-15	>15
0-1	LOW	Low	Average
2	LOW	Average	High
3 or More	Average	High	High

Figure 7: Table 1 :

2

FTR's	1-5	DATA ELEMENTS 6-19	>19
0-1	LOW	Low	Average
2-3	LOW	Average	High
> 3	Average	High	High

Figure 8: Table 2 :

3

Rating	EO	VALUES EQ	EI
Low	4	3	3
Average	5	4	4
High	7	6	6

Figure 9: Table 3 :

RET's	1-19	DATA ELEMENTS 20-50	> 50
1	Low	Low	Average
2-5	Low	Average	High
> 5	Average	High	High

Figure 10: :

5

Rating	ILF	VALUES	EIF
Low	4		3
Average	5		4
High	7		6

Figure 11: Table 5 :

6

F1. Does the system require reliable backup
and recovery?
F2. Are data communications required?
F3.

Figure 12: Table 6 :

7

CASE STUDY	UML DESIGN METRIC NUMOPSCLS VALUE	ACTUAL SOFTWARE OPERATIONS COUNT
DSS	1	1
VCR	12	12

Figure 13: Table 7 :

8

ALGORITHM	AVERAGE PERMISSIBLE ERROR
LOC	27.5
FPA	7.5
UML TOOLS	3.5

Figure 14: Table 8 :

[Tong (2004)] ‘A Comparison of Metrics for UML Class Diagrams’. Yi Tong . *ACM SIGSOFT Software Engineering Notes Page* September 2004. 1.

[Vafaei] *A New Method Software Size Estimation based on UML Metrics*, Jahan Vafaei .

[Yue and Barry] *An Empirical Study of eServices Product UML Sizing Metrics*, Chen Yue , Boehm Barry .

[Wei ()] ‘An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes’. Li Wei . *IEEE Transactions On Software Engineering* November 2003. 1043. 29 (11) .

[Ramanath (2003)] ‘Empirical Analysis of CK Metrics for Object-oriented Design Complexity: Implications for Software Defects’. Subramanyam Ramanath . *IEEE Transactions on Software Engineering* April 2003. 29 (4) .

[Arasimhan Lakshmi ()] ‘Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE)’. . V Arasimhan Lakshmi . *Issues in Informing Science and Information Technology* 2009. 6.

[James et al.] Rumbaugh James , Ivar Jacobson , Booch Grady . *The Unified Modeling Language User Guide” Second Edition2008, pg 5, chap1*, ISBN p. .

[Chidamber] *Managerial use of metrics for Object-oriented software: an exploratory analysis*, Chidamber . IEEE.

[Edith (2011)] ‘Metrics for Component based Measurement Tools’. Linda Edith , P . *International Journal of Science & Engineering* May -2011. 2 (5) .

[Shaik Amjan ()] ‘Metrics for Object Oriented Design Software Systems: A Survey’. Shaik Amjan . *Journal of Emerging Trends in Engineering and Applied Sciences* 2010. 1 (2) p. . (JETEAS))

[Xenos ()] ‘Object-oriented metrics -a survey’. M Xenos . *Federation of European Software Measurement Associations*, (Madrid, Spain) 2000. 2000.

[Jacobson Magnus Christerson et al. ()] *Object-oriented Software Engineering*, Patrick Jacobson Magnus Christerson , Gunnar Jonsson , Overgaard . 2008. 66 p. .

[Orysolya] Doban Orysolya . *Cost Estimation Driven Software Development Process*,

[Pressman and Roger ()] *Software Engineering*, S Pressman , Roger . 2005. 22 p. . (Sixth Edition)

[Tegarden and David] P Tegarden , David . *Effectiveness of Traditional Software Metrics for Object-Oriented Systems*,

[James et al. ()] *The Unified Modeling Language User Guide*, Rumbaugh James , Jacobson Ivar , Booch Grady . 2008. ISBN p. . (Second Edition)

[James et al.] *The Unified Modeling Language User Guide” Second Edition2008, pg 6, chap1*, Rumbaugh James , Jacobson Ivar , Booch Grady . ISBN p. .

[Aine ()] ‘Toward a definition of run-time object-oriented metrics’. Mitchell Aine . *7TH ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2003.

[Luigi ()] ‘Using Function Point in the Estimation of Real-Time Software: an Experience’. Lavazza Luigi . *Proceedings 5 th Software Measurement European Forum*, (5 th Software Measurement European ForumMilan) 2008.