Software Architectural Tool Validation for Object-Oriented Testing using with the facilitate Quality Attributes- a Case Study of Snaker Game Project

Lalji Prasad, Sarita Singh Bhadauria

[1] Truba College of Engineering and Technology, Department of Computer science and Engineering, Indore, TCET(RGTU),BHOPAL INDIA, Email: lalji.research@gmail.com ;[2] Department of Electronics ,MITS (RGTU), Madhav Institute of Technology and Science ,Gwalior, INDIA.
Email: saritamits61@yahoo.co.in

Received Month Day, Year (2012).

ABSTRACT

In this research work investigate quality of software architecture testing tool (our architecture testing) model [34] with the help of snaker game project , first draw a architecture of testing method snaker game project based on their attribute nature and shows their relationship, then identify object oriented characteristic (property) for snaker game project at class level architecture, next phase will be quantify testing (based on different software metrics) on each components (class) and after testing we apply different statistical data analysis for validation of our research work and quantify relationship through different software metrics and conclude quality with the help of statistical tool.

Keywords: Architectural Completeness; Architectural Quality Attribute; Architectural Metrics.

1. Introduction

Here we have taken snaker game project for determining quality of software and validation of the software architecture tool , in this game player controls a long, thin creature, resembling a snaker, which roams around on a bordered plane, picking up food (or some other item). Each time the snake eats a piece of food, its tail grows longer, making the game increasingly difficult. The user controls the direction of the snake's head (up, down, left, or right), and the snake's body follows. The player cannot stop the snake from moving while the game is in progress, and cannot make the snake go in reverse. Different researcher work on quality of software architecture and testing for ensuring the quality of software, here discuss only prominence few literature. Bass et al. Articulated importance of software architecture [12] .Soni and et al.“ say , Software architectures describe how a system is decomposed into components, how these components are interconnected, and how they communicate and interact with each other’s” [14]. Perry and Wolf work on Software architecture is concerned with the study of the structure of software, including its topology, properties, constituent components and their relationships and patterns of combination [26]. Gary Chastek and Robert Ferguson enlighten software architectural attributes and quality related issues [1]. Huang and Myers, describe the basic rules for program testing, which provide basic principle for testing [3,10,14,15,16,17]. Poston [26], Williams [27], and Hareton [19] shows, Integration all the data across tools and repositories, Integration of control across the tools and Integration to provide a single graphical interface into the test tool set. Limitation: emphasize only integration tool (usability & portability). Rosenberg [4] provides, the approach to software metric for object oriented programming must be different from the standard metric sets. Some metrics, such as, line of code & cyclomatic complexity, have become accepted as standard for traditional functional / procedural programs, but for an object oriented scenario, there are many proposed object oriented metrics in the literature. Limitation of this work: this provides the only conceptual framework for measurement .Agrawal et al. Colleague [25] cited in this paper the importance of software measurement is increasing leading to the development of new measurement techniques. Limitation: a) It does not provide any relationship between requirement & testing attribute. b) It cannot evaluate for large data sets. Anderson et al.[5] emphasized the software industry has

performed a significant amount of research on improving software quality using software tools & metrics will improve the software quality and reduce the overall development time. Good quality code will also be easier to write, understand, maintain and upgrade. Limitation a) it's not providing any relationship between the requirement testing attribute. b) It does not provide a full featured testing tool (only Complexity & cohesion measure). c) Here provide the only conceptual framework for measurement. Briand et al., and some other researchers [9,11,28,29,30,31] demonstrate aims are that empirically the relationships between most of the existing coupling & Cohesion measures for object oriented (OO) system & fault proneness of object oriented system classes can be studied. Limitation: a) Only emphasis on cohesion & coupling metric. Bitman [6] exhibit key problem in software development of changing software- development complexity and the method to reduce complexity. Limitation: a) It does provide only complexity measurement techniques. Krauskopf et al. [32], and Harrison [8] demonstrate, Coupling is the degree of interdependence between two modules. In a good design, they are kept low. Coupling should be lower in large and complex system. No coupling is highly is desirable but practically it is not possible. The good & bad points of different types of coupling are discussed. The limitation of their work is: a) Only emphasis on cohesion & coupling metrics. Chidambaram [8] and Harrison [7] emphasized the coupling between object (CBO) metric and evaluated for five object oriented systems & compared with alternative design metric called NAS which measure the number of associations between class & its peers (Harrison R.S). NAS metric is directly collectible from design documents such as the object model. Limitation: a) It does not provide any relationship between requirement & testing attribute. b) It does not provide some basic idea for size & effort estimation. c) Measuring complexity of a class is subject to bias. Reiner R. et al., Show How to manage component based software and identify related metrics. [18]

Comprehensive means that it includes all or nearly all features (maintainability, reusability, flexibility and portability) and relationships required for migrating from one testing class to another. It is designed to overcome the limitation of existing software tools by providing a final class level architecture having relationships between various testing classes. Software quality is another focus of our architecture. We wish to achieve good maintainability, reusability, flexibility and portability in the architecture of the software testing tool by validating the architecture using testing algorithms and performing metrics calculation on each relationship existing between the different testing techniques [1, 2, 3].

2. Research Methodology/Experiment

- First establish a requirement specification for qualitative testing tool using formal review specification. Requirement gathering for snaker game project from different literature (research papers, books and technical reports) for the design of comprehensive architecture for a software testing tool. [22,23,24] Create a software architecture testing tool architecture bases on requirement for testing through different literature [33] and identify attributes (data member and member function).Here we take a case study for project snaker game and design relationship class architecture.
- Identify an attribute of the class's architecture and find relationships between different testing classes in the architecture.
- Based attributes and the relationship between function and component we identified different metrics which is supporting our comprehensive architecture. Descriptive Statistics Examine distribution and variance for each measure.
- Validation of our architecture and determines the quality of software products using empirical and comparative analysis of the different case studies. Principal Component Analysis PCA is the standard technique to identify the underlying dimension (class property) that explains the relations between the variation in the data set.
- Finally on the basis of the above study we determine following goals: final architecture of software for testing, determine the quality of software products and study both (Procedural and Component Based) design

An architecture tool (snaker game) is complete if and only if it entirely describes and specifies the system that exactly fulfills all requirements and the model contains all necessary information that is needed to implement that desired model. Increasing the completeness of a requirements specification can decrease its consistency and hence affect the correctness of the final product. Conversely, improving the consistency of the requirements can reduce the completeness, thereby again diminishing correctness [20].Davis states that completeness is the most difficult of the specification attributes to define and

incompleteness of specification is the most difficult violation to detect [31].According to Boehm [22], to be considered complete, the requirements document must

exhibit three fundamental characteristics: (1) No information is left unstated or "to be determined", (2) The information does not contain any undefined objects or entities, (3) No information is missing from this document. The first two properties imply a closure of the existing information and are typically referred to as internal completeness. The third property, however, concerns the external completeness of the document [23]. Architectural Completeness is defined as an architecture including all or nearly all features and relationships required for migrating from one testing class to another.

3. Software Metrics use in Realization for Snaker Game Projects

In this section we try to identify metrics related to architecture. The player controls a long, thin creature, resembling a snaker, which roams around on a bordered plane, picking up food (or some other item). Each time the snake eats a piece of food, its tail grows longer, making the game increasingly difficult. The user controls the direction of the snake's head (up, down, left, or right), and the snake's body follows. The player cannot stop the snake from moving while the game is in progress, and cannot make the snake go in reverse. In snaker game is based on object oriented technology, In this project we have 10 class diagram (figure.1) and each class diagram related with other class diagram with some specific relationship type ,class grafix and snaker interrelated with inheritance property of object oriented system ,similar keyboard, font, balldraw, wormal, levels, plyr, menus, option and master class snaker all interrelated with inheritance property of object oriented system and after analysis of class architecture we find out different architecture related metrics

According above relationship among different testing technique/strategies, we realize the architecture of testing tool using some software metrics and finally determine software quality of software. Chidamber, Agrawal and et al. [4,5,10,12,13,14] proposed twenty two metrics but, here used those metrics which are useful for my research work:

1.Size Metrics:

a) Number of Attributes (NOA)
b) Number of Methods(NOM)
c) Response for a Class(RFC)
d) Number of Children(NOC)

2.Reuse Metrics:

a) Reuse Ratio(U)
b) Specialization Ratio(S)

3. Inheritance Metrics:

a) Method Inheritance Factor (MIF)
b) Attribute Inheritance Factor (AIF)
c) Depth of Inheritance (DIT)

4. Polymorphism Metrics:

a). Number of methods overridden by a subclass (NMO)

b) Polymorphism Factor (PF)

5.Coupling and Cohesion Metrics

a) Coupling Between Object (CBO)

In above metrics some of their values are very low then there impact in data analysis is negligible and others used for providing help to decide the quality of software products (details in table.2).Quality attribute standard of architectural diagram find through metrics analysis in below graphs.

4. Result Analysis and Discussion

Realizing this model through attribute relationship and determine quality of model using measurement of metrics, and graphical representation and realizing this model

DIT:- Inheritance (generalization), is a key concept in the object model.While reuse potential goes up with the number of ancestors, so does design complexity, due to more methods and classes being involved. Studies have found that higher DIT counts correspond to greater error density and lower quality. A class situated too deeply in the inheritance tree will be relatively complex to develop, test and maintain. It is useful, therefore, to know and regulate this depth. A compromise between the high performance power provided by inheritance and the complexity which increases with the depth must be found. A value of between 0 and 4 respects this compromise. RFC:-Larger RFC counts correlate with increased testing requirements. NOA: - A class with too many attributes may indicate the presence of coincidental cohesion and require further decomposition, in order to better manage the complexity of the model. If there are no attributes, then serious attention must be paid to the semantics of the class, if indeed there are any. A high number of attributes (> 10) probably indicate poor design, notably insufficient de-

composition. A value of between 2 and 5 respects this compromise. NOC: -If Values of NOC are larger than reuse of classes also increases, and by this reason increased testing. A class from which several classes inherit is a sensitive class, to which the user must pay great attention. It should, therefore, be limited, notably for reasons of simplicity. A value of between 1 and 4 respects this compromise. NOM: - this would indicate that a class has operations, but not too many. A value greater than 7 may indicate the need for further object-oriented decomposition, or that the class does not have a coherent purpose. This information is useful when identifying a lack of primitiveness in class operations (inhibiting re-use), and in classes which are little more than data types. A value of between 3 and 7 respects this compromise. This metric proved to be the best indicator of the maintenance effort by indicating the class that is more error prone. CBO: - Excessive coupling limits the availability of a class for reuse, and also results in greater testing and maintenance efforts. Use links between classes define the detailed architecture of the application, just as use links between packages define the high level architecture. These use links play a determining role in design quality, notably in the development and maintenance facilities. Value of 0 indicates that a class has no relationship to any other class in the system, and therefore should not be part of the system. A value between 1 and 4 is good, since it indicates that the class is loosely coupled. A number higher than this may indicate that the class if too tightly coupled with other classes in the model, which would complicate testing and modification, and limit the possibilities of re-use.

Result Analysis: In this section the results of PC analysis are presented in figure. 11, figure. 12 and tables. 3. The PC analysis extraction method and varimax rotation method are applied to different class level metrics. PCA is one of the benchmark for dimension reduction technique here first principal components extract a maximum of the variables and second they are uncorrelated .The First one ensures that the minimum of total information will be missed when looking at the first few principal components. The second one ensures that the extracted information will be organized in an optimal way. Numbers of dimensions captured are quite less than the total number of metrics, implying that many metrics are highly related .Here we used normalizes our variable into three dimensions. In appendix section, we discuss details result data analysis using different table and figure show principal component and eigenvalues in the appendix along with variance (standard deviation).

5. Conclusion

In this research work, we identify implements a set of metrics for measurement of architectural testing model, used to evaluate the quality of the architectural models. Certain model characteristics are measured against quality criteria determined by users thereby allowing you to check that your models meet these quality criteria and appraise the overall quality of a project and find out development of different sub-systems is standard or not .This research work used for developing industrial tools for larger data set, and finally most of the values of our architectural model are following standard values .Hence our architecture is useful for any testing process.

.

REFERENCES

[1] Gary Chastek and Robert Ferguson, "Toward Measures for Software Architectures (Software Engineering Measurement and Analysis)," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2006-TN-013, March 2006.

[2] Howden W. E., "Functional Testing and Design Abstractions," *System and Software (Elsevier)*, vol. 1, pp. 307-313, 1980.

[3] J. Huang C., "An Approach to Program Testing," *ACM Computing Surveys*, pp. 113-128, September 1975.

[4] Rosenberg Linda H., "Applying & interpreting object oriented Metrics," Software Assurance Technology Center (SATC) and NASA Goddard Space Flight Center , Utah, Software Technology Conference April 1998.

[5] Anderson John L. Jr., "How to Produce Better Quality Test Software," *IEEE Instrumentation & Measurement Magazine* , vol. 8, no. 3 ISSN : 1094-6969, August 2005.

[6] Bitman William R, *Balancing software composition & inheritance to improve reusability cost, and error rate.*: Johns Hopkins APL Technical Digest Vol. 18(4) , 485–500., November 1997.

[7] Harrison R., Counsell S., and Nithi R., "Coupling metrics for object oriented design," in *Software metrics, symposium*, MD, USA, November 1998, pp. 150-157

[8] Chidamber S. and Kemerer C., "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, pp. 476-493, 1994.

[9] Agarwal k. K., Sinha Y., Kaur A., and Malhotra R., "Exploring Relationships among coupling metrics in object oriented systems," *CSI*, vol. 37 (1), March 2007.

[10] Glenford J. Myers, *The Art of Software Testing*, 2nd ed.: John Wiley & Sons, 2004

[11] Dr. Linda Rosenberg, Ted Hammer, and Jack Shaw, "Software Metrics and Reliability," Software AssuranceTechnology Center (SATC), NASA, 1998.

[12] Bass L., Clements P., and Kazman R., *Software Architecture in Practice*, 2nd ed. Boston: MA: Addison-Wesley, 2003.

[13] Nick Jenkins, "A Software Testing Primer," 2008.

[14] Soni D., Nord R., and Hofmeister C., "Software Architecture in Industrial Applications," in *Proceedings of the 17th International Conference on Software Engineering. Seattle NY: ACM Press*, Washington, New York, April 23-30, 1995.

[15] Hetzel William C., *The Complete Guide to Software Testing*, 2nd ed.: Wellesley, Mass.: ED Information Sciences ISBN:0894352423. Physical description: ix, 280 p.: ill; 24cm, 1988.

[16] Jiantao Pan, *Software Testing 18-849b Dependable Embedded Systems Spring.*, 1999.

[17] Edward Miller, "Introduction to software testing technology. In Tutorial: Software Testing & Validation Techniques," *IEEE Computer Society Press*, pp. 4-16, 1981.

[18] Reiner R. Dumke and Achim S. Winkler, "Managing the component- Based Software Engineering with Metrics," *0-8186-7940-9/97 IEEE*, 1997.

[19] Hareton K.N. Leung, "Test Tools for the Year 2000 Challenges,".

[20] Williams C. T, "The STCL test tools architecture," vol. 41, no. 1

[21] Perry D. E. and Wolf A. L., "Foundations for the study of software architecture," *SIGSOFT Soft. Engg.*, 17 (4), 1992

[22] Boehm BW, "Verifying and validating software requirements and design specifications," *IEEE Software*, vol. 1, no. 1, pp. 75-88, 1984.

[23] Cordes DW and Carver DL., "Evaluation methods for user requirements documents," *Information and system Technology*, vol. 31, no. 4, pp. 181-188, 1989

[24] Davis AM, *Software Requirements: Analysis and Specification*, 2nd ed.: Prentice Hall, 1993.

[25] K. K. Agarwal, Yogesh Sinha, Arvinder Kaur, Ruchika Malhotra " Exploring Relationships among coupling metrics in object oriented systems. Journal of CSI vol. 37, no.1, January March 2007

[26] Robert M. Poston, "Testing tool combine best of new and old," *IEEE Software*. March 2005.

[27] Williams et. Al., "The STCL Test Tool Architecture," *IBM Systems Journal*, Vol 41, No.1, 2002.

[28] Lionel C. Briand, John W. Daly, and Jurgen Wust, "A unified framework for coupling measurement in ob-ject-oriented system", *IEEE transaction on software engineering*, 1996.

[29] Lionel C. Briand, John Daly " A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", Fraunhfer IESE, 1999.

[30] Lionel C. Briand, "Investigating Quality control in object oriented design: an industrial case study" *ACM-1999*

[31] Birand, W. Daly and J. Wust "Exploring the relationship between design measures and software quality.*Journal of systems and software*, 5(2000) 245-273.

[32] Juan Carlos Esteva, "Learning to Recognize" (Krauskopf, 1990) Jan Krauskopf, "The cohesive highs and the coupling lows of good software design", *IEEE*, 1990.

[33] Sun Chong-ai ,Leu Chao, "Architecture Framework for object-oriented Design," *IEEE Transaction on Software Engineering*, 2004.

[34] Lalji Prasad and Sarita singh Bhadauria, A full featured component based architecture testing tool, *International Journals of Computer Science Issues*, Vol. 8, Issue 4, 2011.

Appendix:

1. **Snooker game Class Diagram Architecture:**

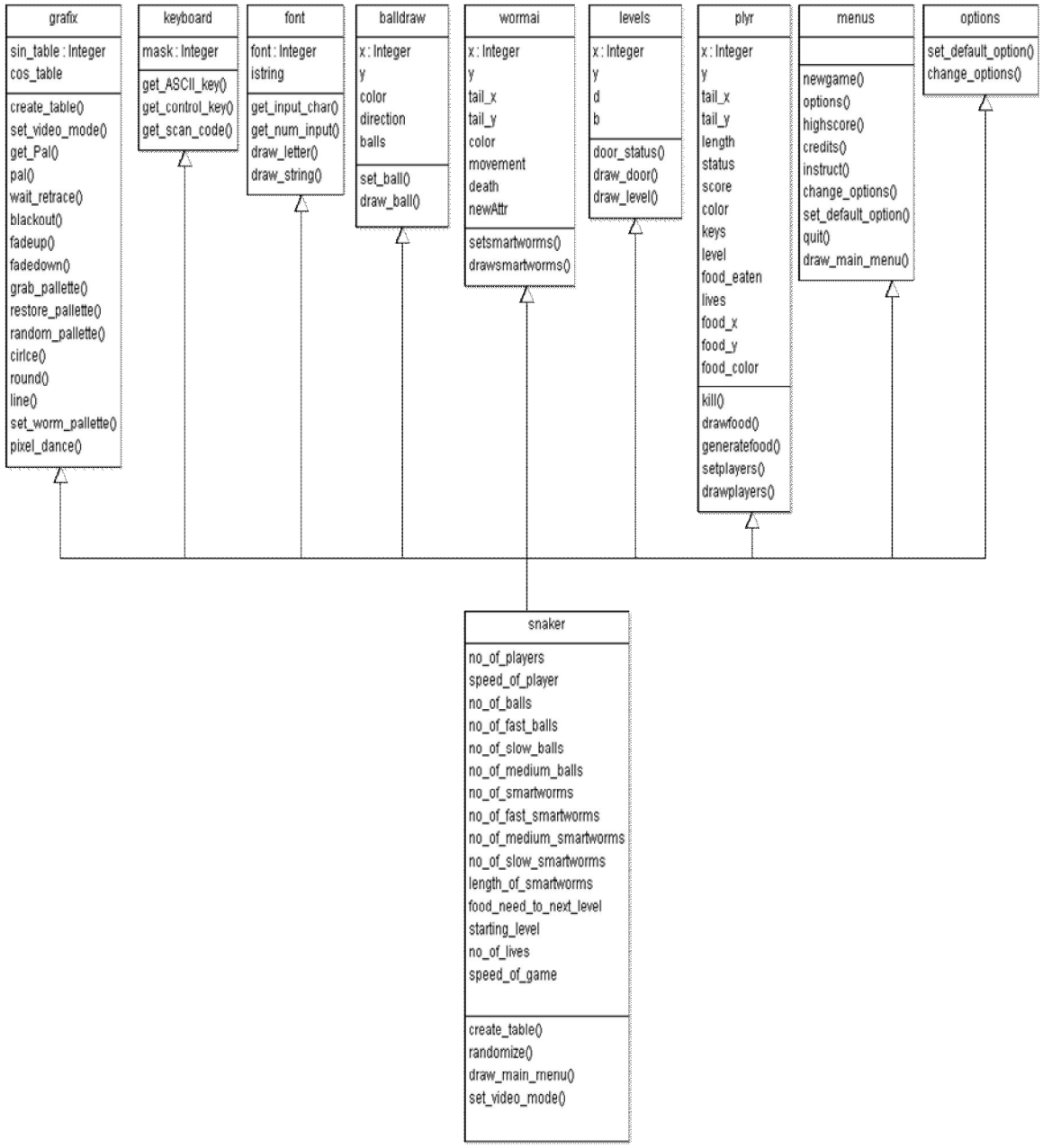

Fig: 1: Class Diagram for Snaker Game Project

2. Metrics Calculation: This table evaluates the values of the different metrics for each class present in the snaker game.

I. Inheritance Metrics:

a) MIF- Method Inheritance Factor

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

Where $M_a(C_i)=M_i(C_i)+M_d(C_i)$
And TC=10
MIF=3/53

b) AIF-Attribute Inheritance Factor

$$AIF = \frac{\sum_{i=1}^{TC} A_d(C)}{\sum_{i=1}^{TC} A_a(C_i)}$$

AIF=52/67

II. Reuse Metrics:

b) Reuse Ratio (U)

U= Number of super classes/Total number of classes
U=9/10

c) Specialization Ratio(S)

S= Number of subclasses/Number of super classes
S=1/9

III. Polymorphism Metrics

a) NMO-Number of methods overridden by a subclass

NMO Snaker=3

b) Polymorphism Factor(PF)

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_j)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$$

Where, $M_n(C_i)$= number of new methods
$M_o(C_i)$ = number of overriding methods
DC (C_i) =Descendant count
PF=3/46

2. **Range table:** The range table evaluates the minimum and maximum ranges for the metrics calculated in below table. On the basis of metrics value and metrics relation to the qualitative property of software we analyzed in below graphs

Classes / Metrics	NOA	NOM	RFC	DIT	NOC	CBO
Snaker class	15	4	15	1	0	4
Grafix Class	2	16	16	0	1	0
Keyboard class	1	3	3	0	1	0
Font class	2	4		0	1	
Balldraw Class	5	2	3	0	1	1
Wormai Class	8	2	2	0	1	0
Levels class	4	3	-	0	1	-
Plyr class	15	5	-	0	1	-
Menus Class	0	9	-	0	1	-
Options Class	0	2	12	0	-	2

Table .1: Metrics calculation table for Snaker game

Size metrics affecting Simplicity:

Number of attributes (NOA): The graph shows the relationship between NOA and simplicity factor which linearly increases until the number of attributes is less and later as NOA increases simplicity reduces.

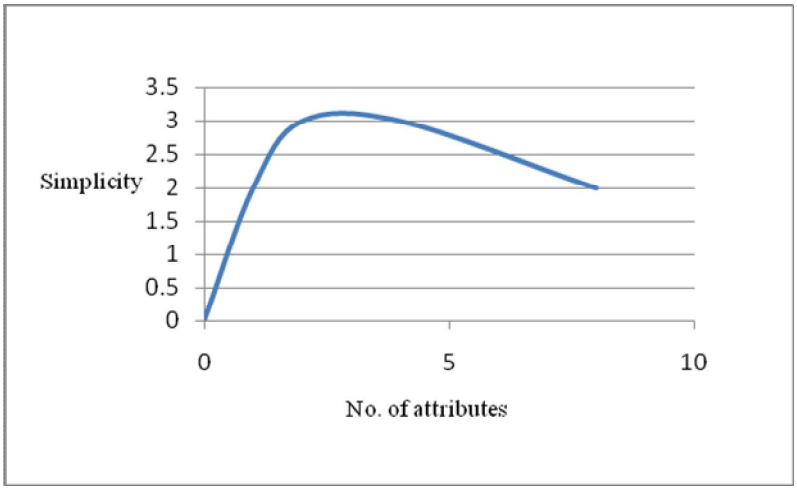

Fig: 2: Graph between simplicity and NOA

Number of methods (NOM): The graph shows the relationship between NOM and simplicity factor. Increment in NOM reduces the simplicity of the program.

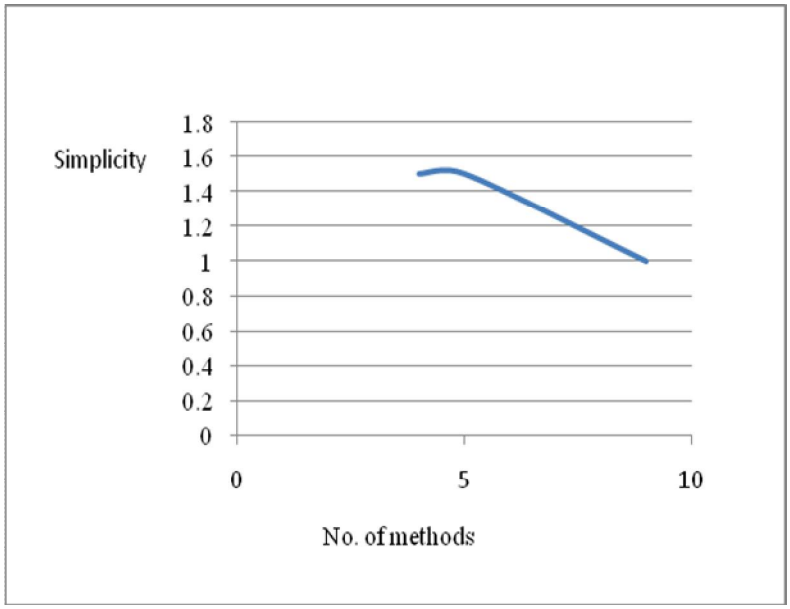

Fig: 3: Graph between simplicity and NOM

Response for a class (RFC): The graph shows the relationship between RFC and simplicity factor. The Response for a class does not affect simplicity after a certain limit and remain constant.

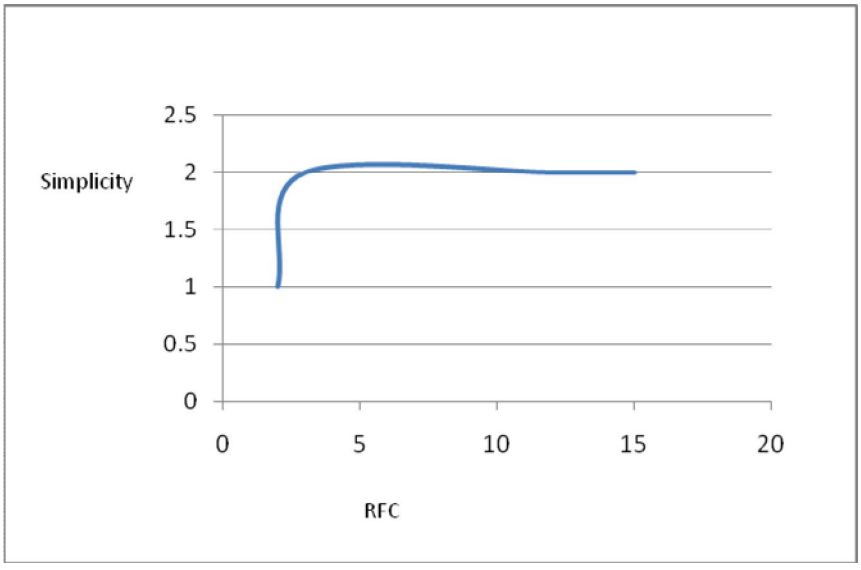

Fig: 4: Graph between simplicity and RFC

Size metrics affecting portability:

Number of attributes (NOA): The graph shows the relationship between NOA and portability factor which linearly increases by the number of attributes is less and later as NOA increases portability reduces.

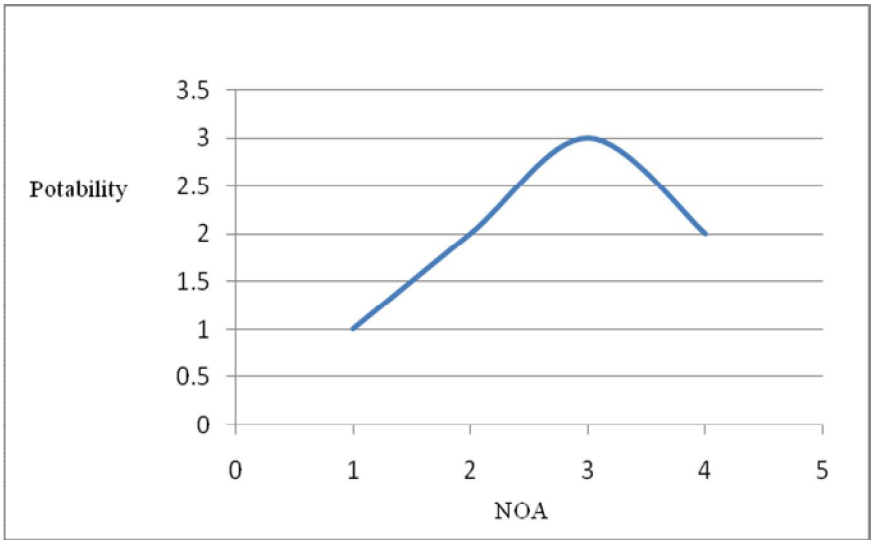

Fig: 5: Graph between portability and NOA

Number of methods (NOM): The graph shows the relationship between NOM and portability factor which linearly increases by the number of attributes is less. Further increment in a number of methods decreases the portability.

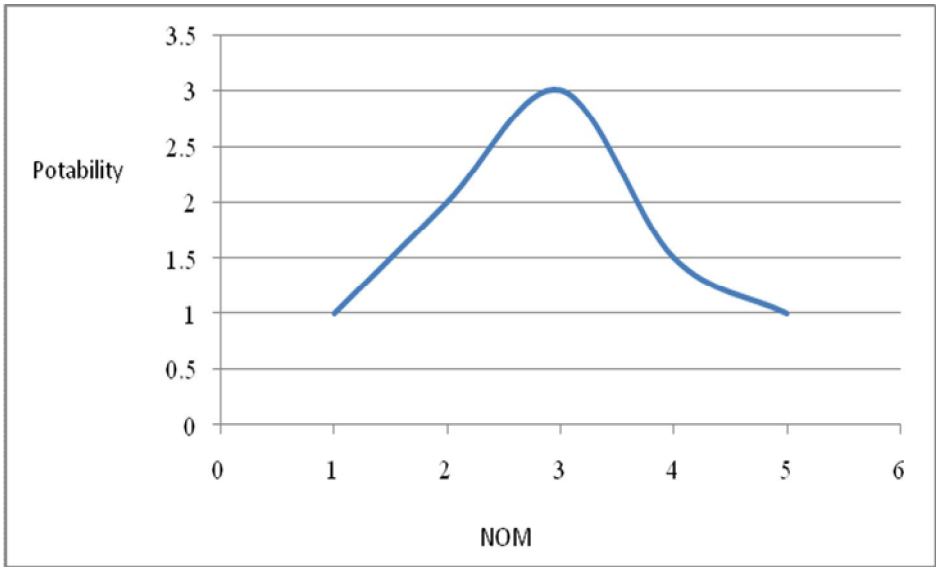

Fig: 6: Graph between portability and NOM

Size metrics affecting user requirements:

Number of attributes (NOA): The more the number of attributes the more requirements of user is satisfied. Hence it depicts a linear relationship.

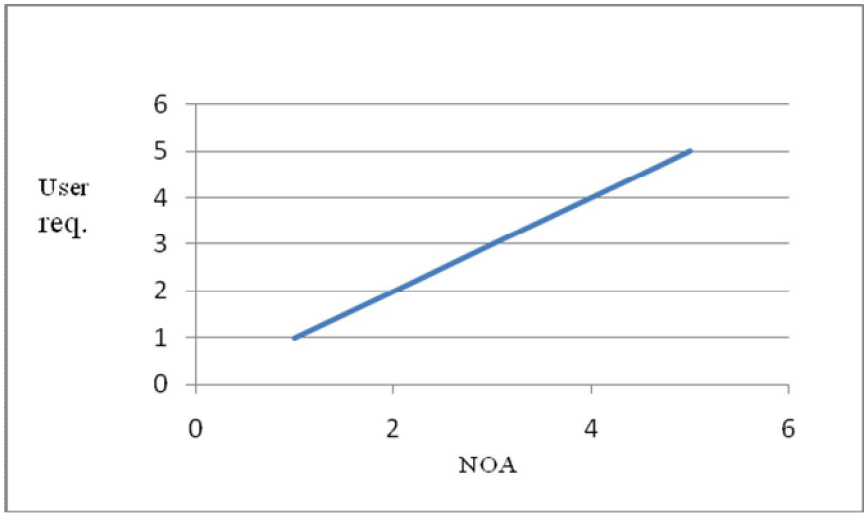

Fig: 7: Graph between user requirements and NOA

Number of methods (NOM): Initially the relationship between the user requirement and NOM is linear, but with further increment is the number of methods the user requirement stabilizes.

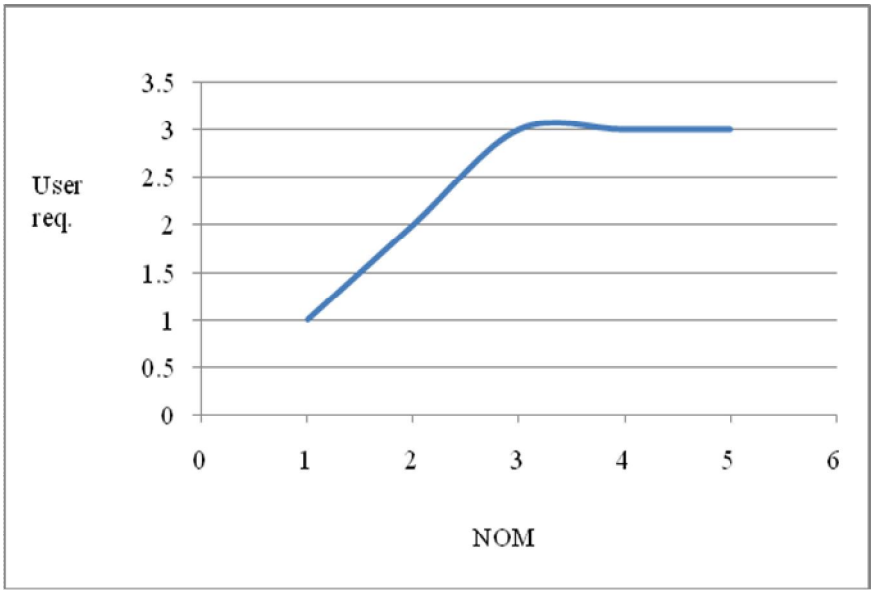

Fig: 8: Graph between user requirements and NOM

Polymorphism metrics affecting high performance:

Number of methods overridden by a subclass (NMO): Overriding of methods increases the performance of the program but further increment of overridden methods decreases the performance as complexity increases.

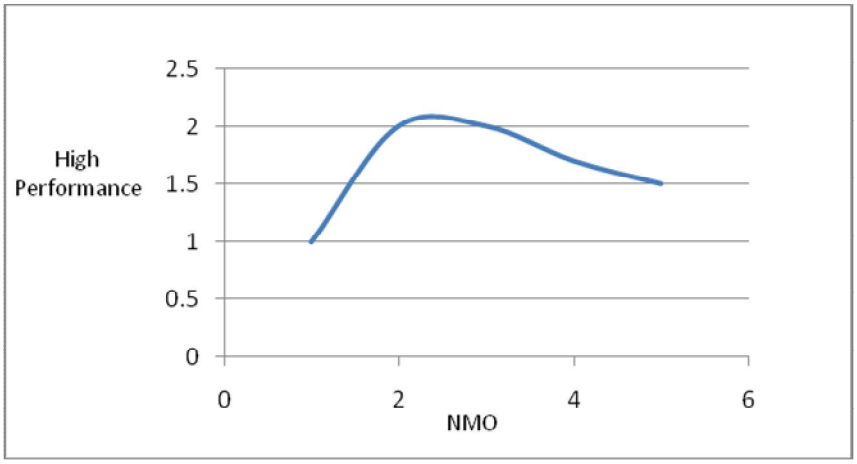

Fig: 9: Graph between high performance and NMO

Polymorphism metrics affecting reusability:

Number of methods overridden by a subclass (NMO): Overriding of methods by subclass reduces the reusability to a greater extent when more methods are overridden.

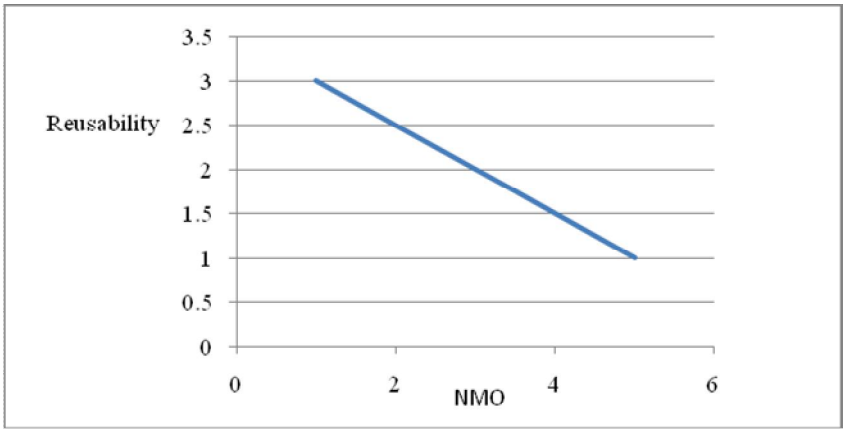

Fig: 10: Graph between reusability and NMO

PCA ANALYSIS: - In this section the results of PC analysis are presented. The PC analysis extraction method and varimax rotation method are applied to different class level metrics. PCA is one of the

benchmark for dimension reduction technique here first principal components extract a maximum of the variables and second they are uncorrelated .The First one ensures that the minimum of total information will be missed when looking at the first few principal components. The second one ensures that the extracted information will be organized in an optimal way. Numbers of dimensions captured are quite less than the total number of metrics, implying that many metrics are highly related .Here we used normalizes our variable into three dimensions. Table 2 shows the value of architectural tool, it shows mean and standard deviation which is help us for deciding our architecture validation.

Table: 2: PCA (Snaker Game)

Metrics	Min	Max.	Mean	Mdn.	S.dev.	PCA_1_Axis_1	PCA_1_Axis_2	PCA_1_Axis_3
NOA	0	15	5.19999981	3	5.71000004	1.33559453	1.15662634	-0.16061185
NOM	2	16	5	8.5	4.38999987	2.8792522	-1.7822665	0.02702049
RFC	0	16	5.0999999	2.5	6.55000019	1.4571259	1.40078652	0.10860458
DIT	0	1	0.1	0	0.31	-2.17278409	-0.35446757	0.06206273
NOC	0	1	0.80000001	1	0.41999999	-1.8494159	-0.40814102	-0.26119137
CBO	0	4	0.69999999	0	1.33000004	-1.64977264	-0.01253791	0.22411549

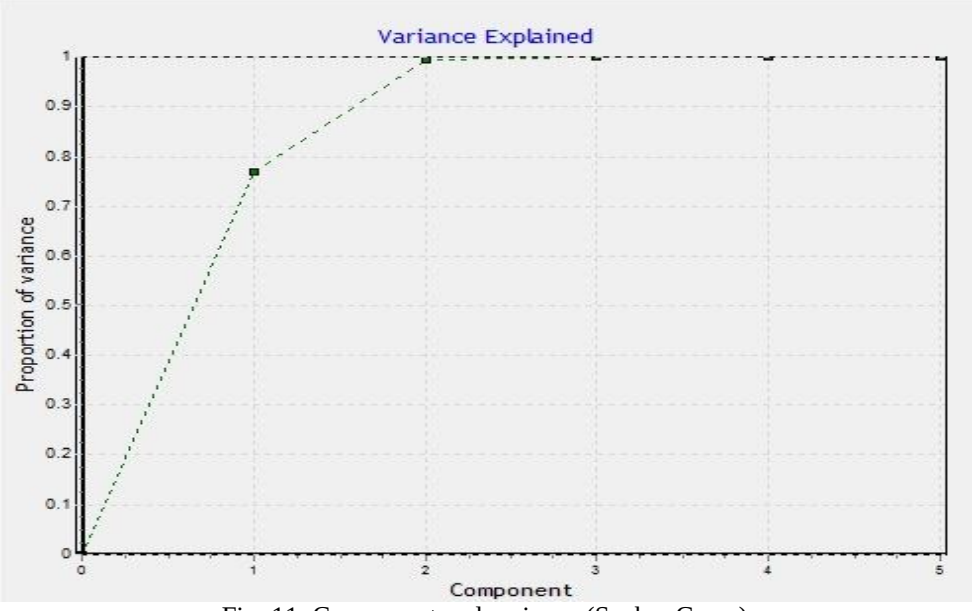

Fig: 11: Component and variance (Snaker Game)

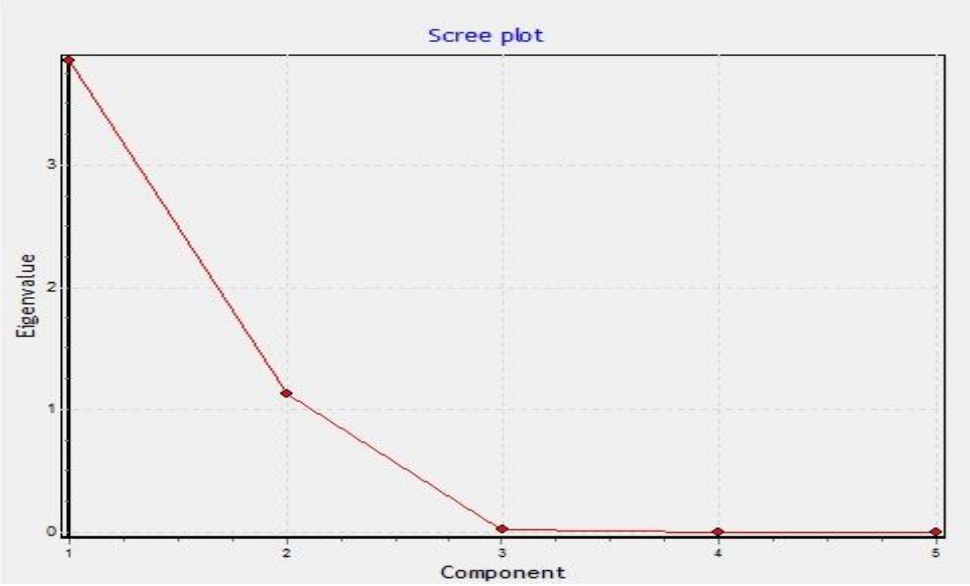

Fig: 12: Eigenvalue with component (Snaker Game)

In above table.2, In first PCA the NOM value higher than others metrics , then its uniquely determine the characteristic, In second PCA axis RFC value is higher than others metric's value ,then its uniquely determines the characteristics .In third PCA axis CBO is higher than others metrics ,then its uniquely determine the characteristic and fig.11 shows relationship of component with variance and fig.12,Eigenvalue with component.