International Journal of Intelligent Computing and Information Science

Assessing the stability of selected software components for reusability

Ajayi, Olusola O.
Department of Computer Science Adekunle Ajasin University Akungba Akoko, Ondo State Nigeria
ajayicomputer@gmail.com

Chiemeke, Stella Chinye
Department of Computer Science University of Benin Benin City, Edo State, Nigeria
schiemeke@uniben.edu

Kingsley Chiwuike Ukaoha
Department of Computer Science University of Benin Benin City, Nigeria.
kingsley.ukaoha@uniben.edu

Abstract -The need to develop software of great quality with timely delivery and tested components gave birth to reuse. Component reusability entails the use (re-use) of existing artifacts to improve the quality and functionalities of software. Many researches have considered and justified common reusability factors such as customizability, portability, interface complexities, understandability/Documentability etc. but with limited works on stability as a factor. The need to experiment stability (in the context of volatility) as a factor for determining component reusability, is an attempt to lend our voice to the domain of component reusability. This study introduces and justifies stability, in the context of volatility of software component, as a factor that determines the reusability of software components. As part of the study's methodology, sixty-nine (69) software components were collected from third party, and data extracted from their features were used to compute the metric values of stability. The experimented conducted proved the stability status of the various component types considered.

Keywords **- software component, reusability, soft-computing, adaptive neuro-fuzzy, stability, interdependency, afferent, efferent, coupling**

I. INTRODUCTION

Reusability is the degree to which a software component can be reused [1][2]. This consequently leads to reduced software development cost and less development time as it enables less writing but more of assembly. Reusability plays an important role in

CBSD and also acts as the basic criterion for evaluating component. [3] asserted that, reusability of a component is an important aspect, which gives the assessment to reuse the existing developed component, thereby reducing the risk, cost and time of software development. If a component is not reusable, then the whole concept of component-based software development fails [4]. Reusability is one of the quality attributes of CBSD. It can measure the degree of features/components that are reused in building similar or different new software with minimal change [5]. To realize the reuse of components effectively, reusability estimation has to be carried out. For systematic reuse process, the use of metrics is very germane. Without metrics, evaluating the quality and qualification of the selected components for reuse becomes an uphill task [5].

[6] defined reusability as the quality of any software component to be used again with slight or no modification. Software reuse is the process of creating software systems from existing software assets rather than building them from scratch. The author also viewed Reusability as the quality factor of software that qualifies it to be used again in another application, be it partially modified or completely modified. In other words, software reusability is a measure of the ease with which previously acquired concepts and objects can be used in new contexts. [7] sees reusability of a component as an important aspect, which gives the assessment to reuse the existing developed component.

Thinking mathematically, reusability could be described thus:

Reusability = Usability + Usefulness …(1)

where

usability describes the degree to which an asset (component) is easily usable (reusable), while

usefulness implies the degree of suitability (relevancy) for use (reuse).

According to [8], there are several factors which influence component reusability. The following factors may be needed to check the 'Reusability Level' of a software element. They are: Reliability, Customizability, Interface Complexity, Adaptability, Portability, Understandability, Stability etc.

Reusability is hard to quantify because of numerous factors influencing it, component stability inclusive. The complexity of the situation contributes the fact that it is often not clear at which extent some factors influence reusability.

II. RELATED WORKS

[9] viewed stability as the life time of a component that satisfies the system requirement through its services. The study viewed that if any of the changes happened in that reusable component, immediately the stability factor of that component has to be measured for keeping the same component as a 'Reusable' one. The authors considered stability as one of the four factors (Coupling, Complexity, Stability, and Quality) needed to check the reusability level of software element. However, 'Import' and 'Export' coupling of a component was used to calculate the stability value of that corresponding component. Kamalraj's work centered on reuse clustering for classifying reusable elements. Stability was only introduced to track the type of dependency among components, communication among them and their interior elements.

[10] posit stability as a factor relevant in determining reusability of a component as he states: 'the reusable component should be stable in any environment as the components are portable to the system where workload is varying and many processes are depending on it'. According to the authors, Stability involves achieving consistent and higher process yields. However, stability was used as a fuzzy input with variables such as Low, Medium and High in the ANFIS structure developed by the authors, without reference to porting of the components as suggested in the definition.

In [11], the study established in a preliminary way, the extent to which the software architecture of a software project is stable, with reference to its core components. The work was devoted to the study of the stability of the architectural core of a software project. Authors evaluated the architectural stability of a set of open source software projects with the aim of understanding the potential reusability of their software components. In presenting the results, the various projects were catalogued in a set of stability trends preliminarily defined. The obtained results showed that projects of the internet category were generally more stable than those ones belonging to the software development category. It should be stated however that Aversano's work is tailored towards Architectural Level Components and not Application Level Components, which this study is tailored to. According to [5] and Figure 1, components at application level have more reusability than those at the rest levels.

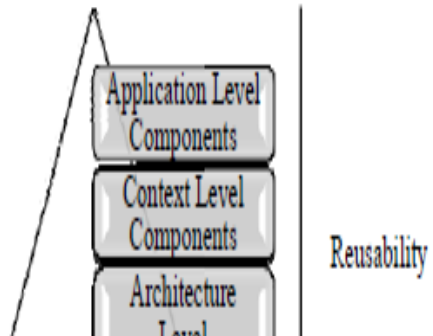

Figure 1: Reusability Hierarchy (Singh et al., 2014)

[12] present techniques for assessing the stability of components extracted from legacy applications using software maturity index. The research presents a technique for assessing the stability of components extracted from legacy applications using software maturity index. The research proposes components reusability assessment technique designed specifically for components stability assessment and possible ranking using Software Maturity Index (SMI). The practical demonstration of the approach was based on maintenance data generated with RANDBETWEEN function of spreadsheet package on three legacy applications used in the demonstration. The study through a careful analysis of the representative legacy maintenance data randomly generated with RANDBETWEEN function from a spreadsheet package yielded some results that led to components ranking technique which could be used to assess and rank legacy components to guide their choice for reuse in modernization. The ranking scheme comprises of the following ordered items, highly stable, fairly stable, stable, unstable, fairly unstable and highly unstable. However, the author measured stability of legacy components using maturity index but with no recourse to the reusability of the component.

III. PROBLEM STATEMENT

The three studies ([9],[10],[11]) established the need for stability in measuring reusability level of a software element. However, their areas of application and context differ from the intended focus of this study, which is, measuring the level to which stability affect component reusability and introducing stability as a factor in the context of volatility.

IV. METHODOLOGY

This study adopts:

i. Component-based development approach. This methodology helps to build component analysis tool for accessing common software components;
ii. Metric-based approach. This methodology aids to measure the degree to which a component is reusable;
iii. Soft-computing approach. This methodology predicts the certainty for reusability.

The following procedures are followed in ensuring a successful implementation of the work:

i. Sixty-nine (69) Commercial Off-The Shelve Software (COTS) Components were be collected from third party organization. According to [13], the key to the success of Component-Based Software Development (CBSD) is its ability to use software components that are often developed by and purchased from third party.
ii. Appropriate stability metrics, in the context of volatility, was applied.
iii. Adaptive Neuro-Fuzzy Inference System (ANFIS) was deployed for evaluating the level of reusability of the selected components based on their types and level of stability.

V. MATERIALS

With established facts that components could be purchased and extracted from third party rather than built ([13]; [14]; [15]; [3]), sixty-nine (69) software components were gotten from four (4) different third-party component development organizations (www.elegantjbeans.com, www.jidesoft.com, www.math.hws.edu, and www.codeproject.com) . Table 1 shows the sources, nature and numbers of the components used while Table 2 shows the features and data extracted from the components. The extracted features were used in the computations of the metrics.

Table 1: Components Used

S/N	Component Source	Nature of Components	Number of Components
1.	www.elegantjbeans.com	Java Components	48

2.	www.jidesoft.com	Java Components	4
3.	www.math.hws.edu	Web Components	13
4.	www.codeproject.com	.Net Components	4

Table 2: Extracted Components' Features and Data

S/N	Component Name	No of methods	No of property	No of writable property	No of readable property
1.	ftptextdataprovider	23	5	5	5
2.	httptextdataprovider	35	2	2	2
3.	Sqldataprovider	37	27	27	22
4.	textdataprovider	19	8	8	5
5.	xmldataprovider	4	1	0	0
6.	awtdatagrid	34	2	1	1
7.	datagrid	31	2	1	1
8.	Tablesawtapp	11	7	7	6
9.	tablejfcapp	4	1	1	1
10.	rowcolumnwiseeditor	9	44	30	20
11.	awtdatatreeviewer	58	11	11	6
12.	datatreeviewer	49	10	10	10
13.	treeawtapplet	8	20	10	10
14.	treesawtapp	7	21	10	10
15.	datebox	48	21	21	20
16.	datemask	43	30	30	17
17.	editmask	90	43	43	43
18.	timemask	34	18	18	14
19.	masking_demo_applet	8	47	35	30
20.	entry_form_app	7	47	32	28
21.	ftpclientapp	4	37	37	30
22.	httpclientgetapp	4	37	32	31
23.	httpclientheadapp	4	12	11	10
24.	httpclientpostapp	4	14	13	13
25.	pop3clientapp	5	25	24	22
26.	smtpclientapp	6	36	34	30
27.	datagrampacket	22	10	10	9
28.	datagramsocket	40	12	12	9
29.	ftpclient	29	11	11	10
…	…	…	…	…	…
58.	GenericGraphApplet	270	251	117	134
59.	IntegralCurves	12	4	2	2
60.	MultiGraph	9	0	0	0
61.	Parametric	7	1	1	0
62.	Riemannsums	5	1	0	1
63.	ScatterPlotApplet	256	211	106	105
64.	SecantTangent	4	0	0	0
65.	SimpleGraph	3	0	0	0
66.	CurrencyCoverter	14	9	3	6
67.	ShapeControl	77	58	18	40
68.	TimePicker	35	24	8	16
69.	MathEx	24	20	18	6

VI. EXPERIMENTATION AND FINDINGS

One of the characteristics of component is interdependency. Interdependency is necessary if the packages of the design are to be collaborated. Thus, some forms of dependency must be desirable, and other forms must be undesirable.

Stability is at the heart of all software design. While designing software, every software practitioners strive to make the product stable in the presence of change. A component is difficult to reuse when the desirable parts of the component are highly dependent upon other components/details which are not desirable. Lack of interdependencies can bring about reusability. One factor that can be used to measure stability is volatility. According to [16], stability is defined as the capability of a software system or component to evolve while preserving its design. He sees stability as the ability of a software item/component to evolve without violating the compatibility among versions. He posits stability as a pointer to the volatility of a component. A highly volatile component will yield instability, while a lowly volatile component is stable and hence, highly reusable.

The concept of interdependency and volatility is associated with the coupling of a component. To this end, this study sees stability as a factor of the volatility of a software component, which is computed as:

Component Stability (COST) = $C_e / (C_a + C_e)$ …(2)

where:

C_a is Afferent Coupling of the component, which implies the number of classes outside the component that depend upon classes within the component.

C_e is Efferent Coupling of the component, which implies the number of classes inside the component that depend upon classes outside the component.

The metric has the range [0,1]. Where COST = 0 (<1), the component is adjudged to be stable, while it is seen as unstable if equals 1. A highly stable software component is highly reusable.

Table 3 shows the values computed for the component stability (COST).

Table 3: Stability Values

Component ID	Afferent Couplings (C_a)	Efferent Couplings (C_e)	Stability = ($C_e / (C_a + C_e)$)	COST
1.	2	1	0.33	0

2.	8	3	0.27	0
3.	5	4	0.44	0
4.	4	3	0.43	0
5.	7	6	0.46	0
6.	5	8	0.62	0
7.	4	2	0.30	0
8.	9	8	0.47	0
9.	7	11	0.61	0
10.	7	11	0.61	0
11.	4	3	0.43	0
12.	9	25	0.74	0
13.	7	7	0.50	0
14.	7	6	0.46	0
15.	5	4	0.44	0
16.	4	3	0.43	0
17.	7	8	0.53	0
18.	9	5	0.36	0
19.	2	4	0.67	0
20.	8	6	0.43	0
21.	8	4	0.33	0
22.	9	2	0.18	0
23.	10	2	0.17	0
24.	15	2	0.12	0
25.	10	3	0.23	0
26.	2	4	0.67	0
27.	6	3	0.33	0
28.	7	5	0.42	0
29.	10	4	0.29	0
30.	8	7	0.47	0
31.	4	6	0.60	0
32.	6	5	0.45	0
33.	3	4	0.57	0
34.	1	2	0.67	0
35.	3	7	0.70	0
36.	4	8	0.67	0
37.	6	5	0.45	0
38.	29	17	0.37	0
39.	2	9	0.82	0
40.	31	17	0.35	0
41.	4	2	0.33	0
42.	5	1	0.17	0
43.	7	8	0.53	0
44.	4	2	0.33	0
45.	3	9	0.75	0
46.	7	8	0.53	0
47.	6	2	0.25	0
48.	7	5	0.42	0
49.	4	3	0.43	0
50.	0	1	1.00	1
51.	0	1	1.00	1
52.	3	7	0.70	0
53.	5	1	0.17	0
54.	3	1	0.25	0
55.	2	3	0.60	0
56.	5	1	0.17	0
57.	1	3	0.75	0
58.	19	6	0.24	0
59.	8	4	0.33	0
60.	9	1	0.10	0
61.	6	1	0.14	0
62.	4	1	0.20	0
63.	4	3	0.43	0
64.	4	1	0.20	0
65.	3	1	0.25	0
66.	2	1	0.33	0
67.	1	1	0.50	0
68.	4	2	0.33	0
69.	3	2	0.40	0

A. Parameter Specifications

The followings are the specifications for the parameters used, both for the Fuzzy Inference System (FIS) and the Adaptive Neuro-Fuzzy Inference System (ANFIS) using MATLAB 2017

Table 4: FIS Structure/Properties

Parameter	**FIS Name**	**Parameter Type/Range**
Input Parameter	COST	[0 1]
	Input FIS Type:	Sugeno
	MF Type:	Triangular
	Output Name:	CompoStability
	Output Type:	Linear
	Input Parameters:	Low [1 10 100] Medium [1 1 1] High [0.1 0.75 0.99] Very High [0 0 0]
	Output Parameters:	HighlyStable [0 0] Stable [0.1 0.99] Unstable [1 1] HighlyUnstable [1 100]

Table 5: ANFIS Specifications

S/N	**Parameters**	**Main Attribute**	**Others**
1.	**Testing Data**	20 data	29% of the entire data used
2.	**Training Data**	49 data	71% of the entire data used
3.	**No of Epoch**	50	
4.	**Error Tolerance**	0	
5.	**Rules**		
6.	**Logical Operator**	AND	
7.	**Inputs**	1	COST
8.	**Input MF**	3	Low, Medium and High
9.	**Output**	1	Reusability
10.	**Output MF**	3	Low, Medium and High
11.	**Optimization Method**	Hybrid	

B. Results

Having four (4) input variables (Low, Medium, High and Very High) and one (1) quality factor (Stability), 4^1 (4) rules were formed. The rule base was constructed to control the output variable, using the simple IF-THEN rule with a condition/antecedence and a conclusion/consequence. The rules formed are as presented below and in Figures 2 and 3:

1. (COST==Low)=>(CompoStability=HighlyUnstable)
2. (COST==Medium)=>(CompoStability=Unstable)
3. (COST==High)=>(CompoStability=Stable)
4. (COST=VeryHigh)=>(CompoStability=HighlyStable)

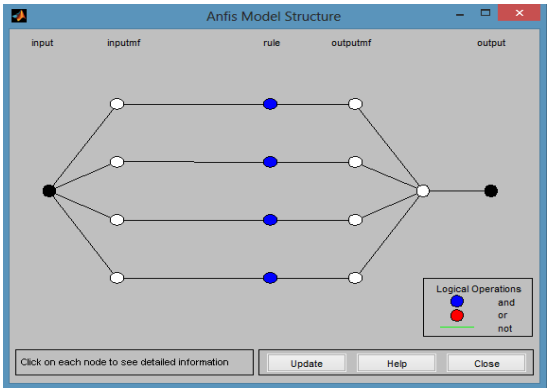

Figure 2: ANFIS Model

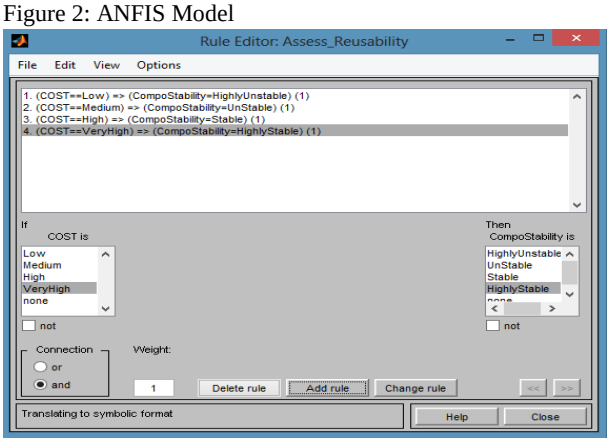

Figure 3: Rule Editor

For this study, Sugeno was selected as the Fuzzy Inference System (FIS) type and Triangular MF chosen as the Membership Function (MF) Type (Figure 4).

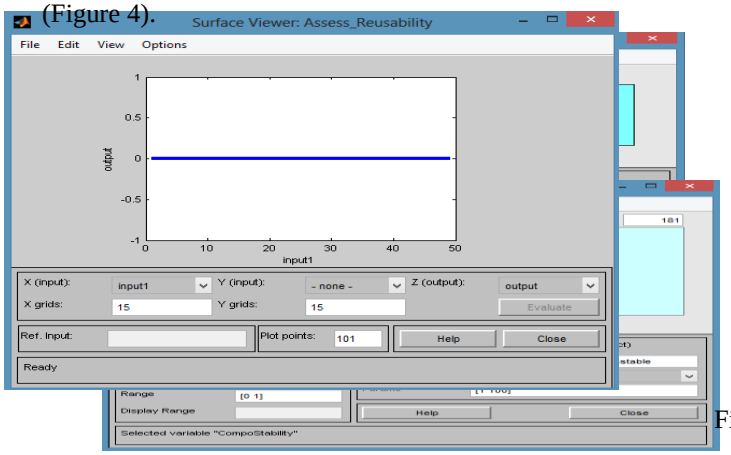

Figure 4: FIS and MF Editors

29% of the data were used as the testing data, while 71% was for the training data. The system reported an average training error of 0.47735 and 0.49134 as the testing error (Figures 5 and 6). This implies a considerable level of prediction accuracy, as most components show high stability posture (Figure 7).

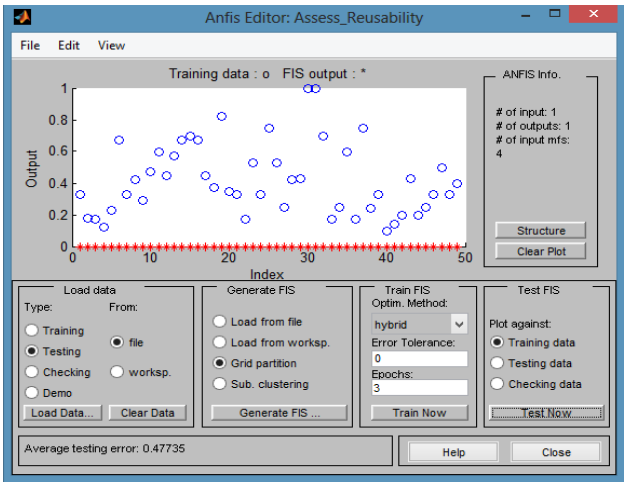

Figure 5: Training Error

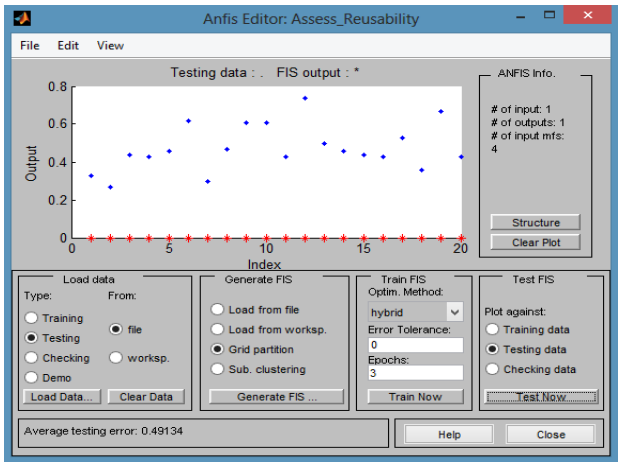

Figure 6: Testing Error

Figure 7: Stability Output

C. Findings

The result of the study's evaluation shows most components highly stable and hence highly reusable.

The result also justifies stability, in the context of volatility as a factor to be consider while measuring the reusability of software components.

VII. CONCLUSION AND RESEARCH DIRECTION

This work has been able to prove that stability is a necessary factor to be consider in the assessment of software component's reusability. The work which made use of three different component types (Java, .Net and Web Components), totaling 69 in number, can be improved upon by increasing the tally to further validate our result. Also, stability can be viewed and defined in another context with a view to further establish its place in software components reusability assessment.

REFERENCES

[1] Washizaki, H., Yamamoto, H., and Fukazawa, Y. (2003). A metrics suite for measuring reusability of software components. Proceedings of the 9th International Symposium on Software Metrics. Sept 3-5, Sydney, Australia, pp. 201-211

[2] Fazal-e- Amin, Mahmood, A. K., and Oxley, A. (2011). A Review of Software Component Reusability Assessment Approaches. *Research Journal of Information Technology*, 3(1):1-11.

[3] Kumar, V., Kumar, R., and Sharma, A. (2013). Applying Neuro-Fuzzy Approach to build the Reusability Assessment Framework across Software Component Releases – An Empirical Evaluation. International Journal of Computer Applications. 70(15): 41-47

[4] Thakral, S., Sagar, S., and Vinay (2014). Reusability in Component Based Software Development – A Review. World Applied Sciences Journal. 31(12):2068-2072.

[5] Singh, A. P. and Tomar, P. (2014). Estimation of Component Reusability through Reusability Metrics. International Journal of Computer, Control, Quantum and Information Engineering. 8(11):1865-1872

[6] Goel, S., and Sharma, A. (2014). Neuro-Fuzzy based Approach to Predict Component's Reusability. International Journal of Computer Applications, 106(5)

[7] Kumar, A., Chaudhary, D., and Kumar, A. (2014). Empirical Evaluation of Software Component Metrics. International Journal of Scientific and Engineering Research. 5(5):814-820

[8] Hristov, D., Hummel, O., Huq, M., & Janjic, W. (2012). Structuring Software Reusability Metrics. The Seventh International Conference on Software Engineering Advances (pp. 422-429). IARIA.

[9] Kamalraj, R., Kannan, A. R., Ranjani, P. (2011). Stability-Based Component Clustering for Designing Software Reuse Repository. International Journal of Computer Applications. Vol. 27, No. 3, pgs. 33-36

[10] Ravichandran, K., Suresh, P., and Sekr, K. R. (2012). ANFIS Approach for Optimal Selection of Reusable Components. Research Journal of Applied Sciences, Engineering and Technology, 4(24): 5304-5312

[11] Aversano, L., Molfetta, M., Tortorella, M. (2013). Evaluating Architecture Stability of Software Projects. IEEE. Pgs. 417-424

[12] Ekanem, B.A., and Woherem, E. (2016). Legacy Components Stability Assessment and Ranking using Software Maturity Index, International Journal of Computer Applications (0975 – 8887) Volume 134 – No.13, January 2016.

[13] Sharma, A., Kumar R., Grover P. S. (2006). Investigation of reusability, complexity and customisability for component-based systems", ICFAI Journal of IT, 2(1).

[14] Sharma, A., Kumar, R. and Grover, P. S. (2009). Reusability assessment for software components. ACM SIGSOFT Software Engineering Notes. 34(2):1-6.

[15] Bhardwaj, V. (2010). Estimating Reusability of Software Components Using Fuzzy Logic. A Master of Science in Mathematics and Computing Thesis, School of Mathematics and Computer Applications, Thapar University, India.

[16] Grosser, D., Sahraoui, H. and Valtchev, P. (2003): An analogy-based approach for predicting design stability of java classes. In *Proceedings of the Ninth International Software Metrics Symposium*, 252–262.