Legacy Components Stability Assessment and Ranking using Software Maturity Index

Bassey Asuquo Ekanem
Institute of Engineering,
Technology & Innovation Management
University of Port Harcourt, Nigeria

Evans Woherem
Compumetrics Solutions
Abuja, Nigeria

ABSTRACT
Component-based software modernization is technique that is widely accepted to have the greatest potentials in restructuring legacy applications into modernized versions with best qualities and maintainability attributes amongst other modernization techniques. This technique relies greatly on stable components extracted from the legacy system and selected for reuse. In selecting the components, some reusability attributes are usually considered of which components stability is one of such. However, the task of selecting stable components for reuse especially from legacy applications is a very difficult one due to inadequate techniques and models specifically designed for this. This research therefore, presents a technique for assessing the stability of components extracted from legacy applications using software maturity index. It also provides a means of ranking the assessed components with a scale comprising of Highly Stable, Fairly Stable, Stable, Unstable, Fairly Unstable and Highly Unstable to guide the choice of quality and stable components for reuse in modernization. The research further emphasizes the importance of proper software maintenance data recording from one version to another as such is a major requirement for legacy components reusability assessment.

General Terms
Legacy Components Reusability Assessment

Keywords
Components Stability Assessment, Components Ranking, Software Maturity Index, Software Modernization

1. INTRODUCTION
Legacy applications in recent past have drawn so much attention from software professionals due to their critical nature to routine business operations and the difficulties associated with their maintenance. The fact that software industry experiences at least 10% annual increase in legacy code (Denoncourt, 2011); raises further concern about organization's capacity to cope with their challenges.

The frequent technological changes and the dynamism of the environment in which legacy applications operate usually demand for regular maintenance as a requirement to extend their usable life. However, legacy modifications are usually difficult and expensive due to some unique characteristics of these set of applications, namely language obsoleteness, poor data abstraction, poor code structure, lack of qualified engineers with experience in the obsolete tools, and incomplete documentation (Cipresso, 2010).

Despite these challenges, organizations still find it difficult to abandon or replace them; rather, legacy modernization aimed at transformed the legacy application into modernized versions with features that address the fundamental challenges and possibly drive down maintenance cost is usually considered as the best option (Mishra, 2009). This fact is further affirmed in (Malinova, 2010), Comella-Dorda et al (2010); Saarelainen et al (2006) and Khadka et al (2010) where legacy modernization is reported as being more profitable than outright replacement with a caution that application must be replaced only when it can no longer be evolved.

According to Gartner (2012) CIOs survey report, application modernization is presently one of the top 10 IT technology priorities of CIOs globally. This is because year-in year-out, some application software used in organizations usually mature into legacy application category thereby prompting the need for their modernization to address their maintenance challenges. Legacy modernization could take any of the following forms, namely wrapping, migration, reengineering, and component-oriented reengineering. However, component-oriented reengineering technique is believed to have the greatest potentials in restructuring legacy applications into modernized versions with best qualities and maintainability attributes (Mishra, 2009); Cipresso, 2010).

Successful component-oriented reengineering of legacy applications requires the use of stable components extracted from the legacy application (Younoussi and Roudies, 2015). However, the task of selecting stable components from a host of components in a legacy application could be very difficult, hence the need for a systematic approach in this regards. There is no doubt the fact that, information from such assessment would serve as a guide to professionals in making proper choice of components and providing some levels of confidence in the components being selected for reuse.

Furthermore, existing models for components assessment, mainly focus on measures to ensure well-planned and controlled reuse-oriented software development process in organizations. In other words, majority of these models are designed mainly to support reusability assessment of components built for software development projects with little or no emphasis on reusability assessment of components extracted from legacy systems for reuse in modernization (Younoussi and Roudies, 2015; Fazal-e-Amin et al, 2011; Jasmine and Vasantha, 2010).

In view of the above, this research proposes components reusability assessment technique designed specifically for components stability assessment and possible ranking using Software Maturity Index (SMI). The practical demonstration of this approach is based on maintenance data generated with RANDBETWEEN function of spreadsheet package on three legacy applications used in the demonstration.

2. REVIEW OF RELATED RESEARCH WORKS
For a proper understanding of the research area, a review of related research works particularly in components reusability

assessment was made and reported thus. In Rine and Nada (2000), Reuse Reference Model (RRM) is presented with both technical and organizational elements needed to establish a successful practice of software reuse in organizations. The level of reuse as defined in RRM, determines the capability of improvements in the productivity, quality and time-to-market of the organization. Inoue et al (2004) presents the component rank model using digraph-based system for computing and ranking software components selected for reuse.

Garcia et al (2007) presents the RISE Maturity Model (RISE) designed mainly to support organizations in improving their software development process with respect to components reusability assessment. It also serves as a roadmap for software reuse adoption and implementation where reusability attributes like stability, adaptability, completeness, maintainability and understandability were considered.

In Jasmine and Vasantha (2010), a model called Reuse Capability Maturity Model (RCMM) is presented with emphasis on measures needed to ensure a well-planned and controlled reuse oriented software development. The model is structured into five levels with each level representing a stage in the evolution to a mature reuse process. A set of maturity goals for each level and the activities, task and responsibilities are specified.

Furthermore, in Fazal-e-Amin et al (2011), a review of software components reusability assessment approaches was made, with the research results indicating that the majority of the approaches (i.e. 70%) are based on metrics, and applicable to the object oriented development projects using Java as the target language. The research further emphasizes the need for other approaches particularly experimental based approaches for comparison and better results. In Subedha and Sridhar (2012) a technique for measuring the quality of components for reuse with functional coverage report, software metrics and minimum extraction time is presented. The technique also provides a means of classifying the identified set of components into qualified and not qualified components.

Other relevant research works reviewed include the following: In Younoussi and Roudies (2015), a detailed literature review of recent research works in software reusability is presented with stability, understandability, portability, maintainability, flexibility, independence, documentation, adaptability and interface complexity identified as attributes that influence software components reusability. The research further reports that studies on maturity models of software reuse are limited and more was needed to be done in this area to help organizations in proper auditing of their maturity reuse level.

Kessel and Atkinson (2015) discuss some of the main issues involved in improving the selection support for pragmatic reuse provided by test-driven search engines. It also describes some new metrics that could help address the issues and presents an approach for ranking components in search results.

3. FINDINGS FROM THE REVIEW

The reviewed works clearly indicate that existing techniques and models were designed mainly to support reusability assessment of components built for software development projects with little or no emphasis on reusability assessment of components extracted from legacy systems for reuse in modernization. In other words, these techniques and models were designed primarily to support and ensure that reuse-oriented software development are well-planned and controlled for successful software reuse practice in organizations where they are applied. With this, existing assessment techniques rely on software development data from integration testing for the measurement (Fazal-e-Amin, 2011); whereas maintenance data of legacy application from one version to another are needed in the case of legacy components reusability assessment.

Considering the need to cope with annual increase in legacy code in the software industry where today's modern software are tomorrow's legacy applications and candidates for modernization, there is a need to fill the gap of inadequate techniques for legacy components reusability assessment if the present gains in component-oriented modernization as applicable to legacy system are to be consolidated. This could be addressed by adapting existing models primarily designed for components assessment in development projects to utilize legacy maintenance data in assessing legacy components reusability. More appropriately, new techniques and models could be evolved specifically for legacy reusability assessment to fill the gap.

4. DESCRIPTION OF SOFTWARE MATURITY INDEX (SMI) MODEL

This research relies of Software Maturity Index (SMI) for components stability computation and ranking. Software Maturity Index, a metric in IEEE (1988), specifically IEEE 982.1-1988 was introduced to measure the maturity of software systems as a software evolves from one version to another. The metric is represented below:

$$SMI = (M - (A + C + D))/M$$

where

M = number of modules in current version
A = number of added modules in current version
C = number of changed modules in current version
D = number of deleted modules in current version

More precisely, $SMI = 1 - N/M$, where

M is the total number of modules in the current version of the system and
N is the number of modules added, changed or deleted between the previous version and the subsequent version.

Accordingly, SMI can be used as a measure of product stability. Therefore, when SMI approaches 1.0 the product is said to be stable. Also, when this is correlated with the time it takes to complete a version of the software, an indicator of the maintenance effort needed in maintenance is obtained.

However, a closer examination of this model reveals possible adaption to fit into legacy component stability assessment. In this case, the maintenance data on each component in the recent versions of a legacy application could be collected and used to compute the respective SMIs of each components; in which case modules as used in the model are replaced by legacy components in the application under study. The result of such computation is further interpreted and used to determine components stability. This research is based on this concept and uses maintenance data on three legacy applications from three organizations.

5. RESEARCH METHODOLOGY

The research work was designed as experimental research with the following processes:

i. Review of relevant documentations

ii. Randomization of the needed research data using RANDBETWEEN function in spreadsheet Program

iii. Data Coding and Analysis

iv. Results Interpretation and discussions

A review of relevant literature was made to establish the level of achievements in the research area and identify research gaps. Furthermore, the needed data for the research were generated randomly using RANDBETWEEN function in spreadsheet program. Data generated include number of components in current version (M), number of added components in current version (A), number of changed components in current version (C), and number of deleted components in current version (D). The data were coded and analyzed using statistical package to generate results that were further interpreted and reported accordingly.

6. DATA SHEET AND DATA RECORDING

Data needed in this research are the maintenance data of some legacy application software for some versions, at least the last four versions of the legacy application. The required data were generated randomly on three legacy applications coded as Legacy Applications A, B and C from the RANDBETWEEN function in Microsoft Excel. Also, a datasheet designed specifically for the research (see Table 1) was used to record the generated data for further processing.

Table 1: Datasheet for Legacy Maintenance Data Collection

Component Id Id	Version N-3				Version N-2				Version N-1				Version N			
	M	A	C	D	M	A	C	D	M	A	C	D	M	A	C	D

where M = number of modules in current version

A = number of added components in current version

C = number of changed components in current version

D = number of deleted components in current version

The datasheet has sections for recording the maintenance data of the last four versions of the legacy application which are denoted as Version N-3, Version N-2, Version N-1, and Version N, where Version N is the most recent version.

In other to generate realistic data with the RANDBETWEEN function, the following assumptions were made: For version N-3, the range of values for M were specified as between 6 and 15 based on the assumption that the number of modules in a component at the point of initial deployment will not be below 6 and not above 15. Similarly, the range of values for other operands (i.e. A, C, and D) in all versions were specified as between 0 and 5 with the assumption that modifications to components (i.e. addition, deletion or change) will be between 0 and 5.

7. DATA PRESENTATION AND ANALYSIS

The maintenance data generated on the three legacy applications labelled Legacy Applications A, B and C from the RANDBETWEEN function are given below with the number of components for A, B and C randomly fixed at 15, 10 and 7 respectively:

Table 2: Maintenance Record of Legacy Application A

Component Id	Version N-3				Version N-2				Version N-1				Version N			
	M	A	C	D	M	A	C	D	M	A	C	D	M	A	C	D
A1	11	3	3	1	14	1	1	0	15	1	0	0	16	0	1	0
A2	9	1	1	0	10	0	4	3	10	1	2	0	11	0	5	2
A3	12	0	3	2	12	3	3	1	15	0	2	2	15	1	1	0
A4	10	2	2	0	12	2	3	1	14	2	0	2	16	0	1	0
A5	7	1	1	1	8	4	2	0	12	1	1	1	13	1	5	2
A6	9	2	1	0	11	3	4	1	14	3	3	1	17	0	1	0
A7	12	3	1	0	15	1	0	0	16	0	0	1	16	0	1	0
A8	9	2	3	0	11	4	2	0	15	1	1	1	16	0	1	0
A9	8	1	1	1	9	2	3	1	11	2	2	1	13	1	0	0
A10	11	0	2	1	11	0	1	3	11	2	2	2	13	3	1	0
A11	7	1	1	0	8	1	3	2	9	1	1	0	10	1	3	1
A12	8	2	3	1	10	2	4	0	12	2	1	1	14	0	0	1
A13	10	1	4	3	11	2	2	1	13	0	0	3	13	1	0	0
A14	8	2	2	2	10	3	2	1	13	1	1	1	14	2	0	2
A15	10	1	2	1	11	4	0	3	15	2	1	0	17	0	1	0

Table 3: Maintenance Record of Legacy Application B

Component Id	Version N-3				Version N-2				Version N-1				Version N			
	M	A	C	D	M	A	C	D	M	A	C	D	M	A	C	D
B1	11	1	2	1	12	1	2	1	13	2	2	2	15	1	2	0
B2	8	3	2	0	11	1	0	0	12	1	0	0	13	0	1	0
B3	10	0	2	2	10	2	2	1	12	0	2	1	12	1	0	0
B4	9	1	1	1	10	2	1	1	12	1	1	1	13	0	0	2
B5	10	1	1	0	11	1	4	1	12	1	2	0	13	0	1	0
B6	8	1	3	0	9	3	1	2	12	2	1	0	14	0	1	0
B7	10	2	2	1	12	1	2	1	13	1	2	1	14	1	0	1
B8	6	1	2	2	7	2	1	0	9	1	0	0	10	1	0	0
B9	7	0	1	0	7	1	0	0	8	1	0	0	9	0	1	0
B10	6	3	2	1	9	2	2	2	11	1	1	2	12	0	2	1

Table 4: Maintenance Record of Legacy Application C

Component Id	Version N-3				Version N-2				Version N-1				Version N			
	M	A	C	D	M	A	C	D	M	A	C	D	M	A	C	D
C1	9	1	3	0	10	1	3	0	11	2	4	2	13	3	3	2
C2	11	0	2	1	11	2	2	0	13	0	2	0	13	0	1	0
C3	10	2	2	0	12	4	4	0	16	3	3	1	19	1	0	0
C4	11	4	1	1	15	2	0	1	17	1	0	2	18	0	0	1
C5	8	3	1	1	11	3	3	2	14	0	1	1	14	0	1	0
C6	10	2	3	0	12	4	1	3	16	0	0	1	16	0	1	0
C7	6	2	2	0	8	4	0	3	12	3	0	3	15	0	1	0

In other to generate the components SMIs, appropriate formulae, (in this case the SMI model SMI = (M – (A + C + D))/M described earlier) was entered into the statistical package.

8. RESULTS AND DISCUSSIONS

The results of the analysis of the maintenance data using the statistical package with appropriate formulae (i.e. SMI model) yielded the following:

a) Software Maturity Index of Application Components

The table below presents the SMIs of legacy components for each application:

Table 5: Software Maturity Index ofLegacy Application A

Id	Ver N-3	Ver N- 2	Ver N-1	Ver N
A1	0.36	0.86	0.93	0.94
A2	0.78	0.30	0.70	0.36
A3	0.58	0.42	0.73	0.87
A4	0.60	0.50	0.71	0.94
A5	0.57	0.25	0.75	0.38
A6	0.67	0.27	0.50	0.94
A7	0.67	0.93	0.94	0.94
A8	0.44	0.45	0.8	0.94
A9	0.63	0.33	0.55	0.92
A10	0.73	0.64	0.45	0.69
A11	0.71	0.25	0.78	0.5
A12	0.25	0.40	0.67	0.93
A13	0.20	0.55	0.77	0.92
A14	0.25	0.40	0.77	0.71
A15	0.60	0.36	0.80	0.94

Table 6: Software Maturity Index of Legacy Application B

Id	Ver. N-3	Ver. N-2	Ver. N-1	Ver. N
B1	0.64	0.67	0.54	0.80
B2	0.38	0.91	0.92	0.92
B3	0.60	0.50	0.75	0.92
B4	0.67	0.60	0.75	0.85
B5	0.80	0.45	0.75	0.92
B6	0.50	0.33	0.75	0.93
B7	0.50	0.67	0.69	0.86
B8	0.17	0.57	0.89	0.90
B9	0.86	0.86	0.88	0.89
B10	0.00	0.33	0.64	0.75

Table 7: Software Maturity Index of Legacy Application C

Id	Ver N-3	Ver N-2	Ver N-1	Ver N
C1	0.56	0.6	0.27	0.38
C2	0.73	0.64	0.85	0.92
C3	0.60	0.33	0.56	0.95
C4	0.45	0.80	0.82	0.94
C5	0.38	0.27	0.86	0.93
C6	0.50	0.33	0.94	0.94
C7	0.33	0.13	0.50	0.93

b) Components Stability Assessment and Ranking

For better understanding of the results, graphical representation of the components SMIs were obtained. Figures 1, 2 and 3 are the graphical representation of the SMIs of the components in Legacy Applications A, B and C respectively.

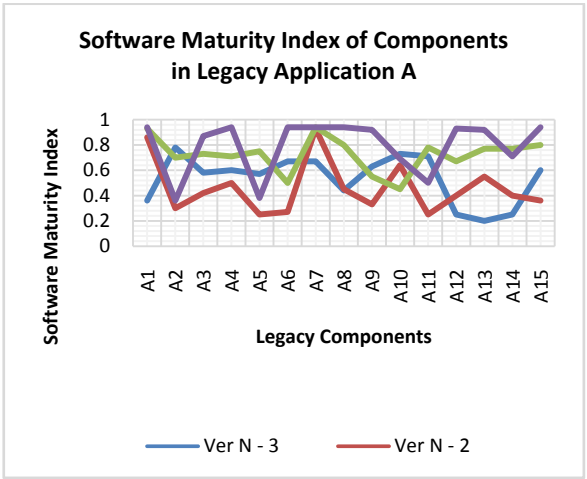

Fig. 2: Software Maturity Index of Components in Legacy Application A

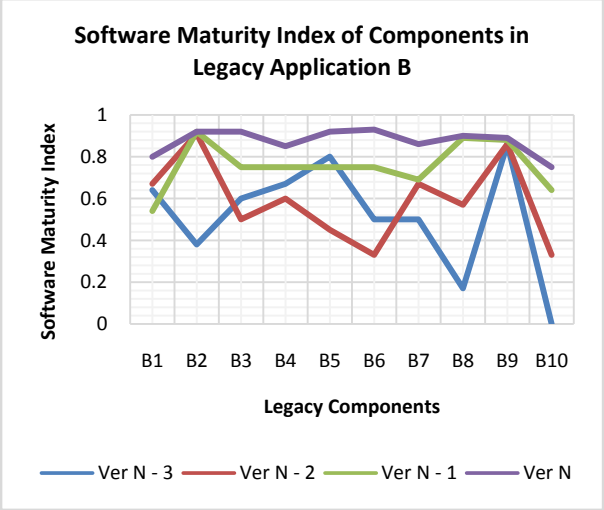

Fig. 3: Software Maturity Index of Components in Legacy Application B

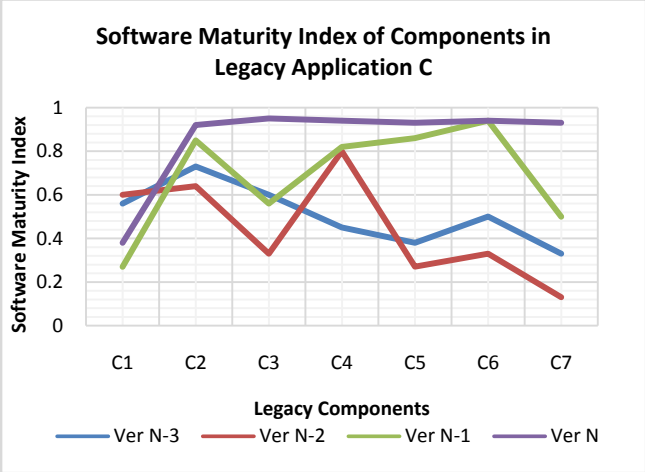

Fig. 4: Software Maturity Index of Components in Legacy Application C

A closer study of the data and the charts reveals some important characteristics of the components:

i) regular and irregular SMIs increases for some components tending to 1or not tending to 1

ii) regular and irregular SMIs decreases for some components receding from 1

A further examination of these characteristics and careful interpretation lead to the following ranking of components being proposed: Highly Stable, Fairly Stable, Stable, Unstable, Fairly Unstable and Highly Unstable. The criteria for this ranking is given thus:

Highly Stable: A component is said to be Highly Stable if it is characterized by regular SMI increases in the last three application versions with all three SMIs tending to 1

Fairly Stable:A component is said to be Fairly Stable if it is characterized by regular SMI increases inthe last three application versions with the last two SMIs tending to 1

Stable:A component is said to be Stable if it ischaracterized by regular SMI increases in the lastthreeapplicationversionswiththeSMI of the most recent version tending to 1

Unstable: A component is said to be Unstable if it is characterized by regular/irregular SMI increases in the last three application versions with the SMIs not tending to 1

Fairly Unstable: A component is said to be Fairly Unstable if it is characterized by regular/irregular SMI decreases in the last three software Versions with the last two SMIs receding from 1

Highly Unstable: A component is said to be Highly Unstable if it is characterized by regular/irregular SMI decreases in the last three software Versions with the SMIs receding from 1

For the purpose of clarity, 0.9 is fixed as a benchmark for SMI tending to 1.

c) Application of the Proposed Assessment and Ranking Scheme

To illustrate how the Assessment and ranking scheme could be applied, legacy application A is used. This application has a total of 15 components to be assessed, coded as A1 to A15. The SMIs of component A1 for instance from version N-3 to Version N are given as 0.36, 0.86, 0.93 and 0.94. This presents a characteristic of a components with regular SMI increases where the SMIs of the last two versions tend to 1. Recall, 0.9 is fixed as the benchmark for SMI tending to 1. This characteristics clearly fits the Fairly Stable rank hence component A1 can be said to be fairly stable. Similarly, applying the scheme to component A7 which SMIs are given as 0.67, 0.93, 0.94 and 0.94 from version N-3 to version N, it could be clearly seen that the component is characterized by regular SMI increases where the SMIs of the last three versions tend to 1, hence the component can be said to be highly stable.

An interesting characteristic is observed with components A3 and A10 where SMIs of A3 indicate regular increases (0.58, 0.42, 0.73 and 0.87) while that of A10 are irregular increases (0.73, 0.64, 0.45 and 0.69). Despite these increases (regular or irregular) the SMIs are still far from 1; a characteristic that fits unstable components, hence A3 and A10 can be said to be unstable. Applying the technique for other components yields the table given below for all components of Application A:

Table 8: Ranking of Legacy Application A Components

Component Id	Software Maturity Index (SMI)				Component Status	Rank
	Ver. N-3	Ver. N-2	Ver. N-1	Ver. N		
A1	0.36	0.86	0.93	0.94	Regular SMI increases with the SMIs of the last two versions tending to 1	**Fairly Stable**
A2	0.78	0.30	0.70	0.36	Irregular SMI decreases with the last two	**Fairly Unstable**

					receding from 1	
A3	0.58	0.42	0.73	0.87	Regular SMI increases in the last three versions with the SMIs not tending to 1	**Unstable**
A4	0.60	0.50	0.71	0.94	Regular SMI increases with the SMI of the most recent version tending 1	**Stable**
A5	0.57	0.25	0.75	0.38	Irregular SMI decreases with the SMI of the last two receding from 1	**Fairly Unstable**
A6	0.67	0.27	0.50	0.94	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
A7	0.67	0.93	0.94	0.94	Regular SMI increases with the SMI of the last three versions tending to 1	**Highly Stable**
A8	0.44	0.45	0.8	0.94	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
A9	0.63	0.33	0.55	0.92	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
A10	0.73	0.64	0.45	0.69	Irregular SMI increase in the last three versions with the SMIs not tending to 1	**Unstable**
A11	0.71	0.25	0.78	0.5	Irregular SMI decreases in the last three versions with the SMIs of the last two receding from 1	**Fairly Unstable**
A12	0.25	0.40	0.67	0.93	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
A13	0.20	0.55	0.77	0.92	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
A14	0.25	0.40	0.77	0.71	Irregular SMI decreases in the last three versions with the SMIs of the last two receding from 1	**Fairly Unstable**
A15	0.60	0.36	0.80	0.94	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**

From table 8, it can be seen that component A7 is highly stable; A1 is fairly stable while A4, A6, A8, A9, A12, A13 and A15 are stable components. The implication is that, these nine components though with variable degrees of stability could be selected for reuse in legacy modernization. In contrast, the remaining six components, namely A2, A3, A5, A10, A11 and A14 with variable degrees of instability are not good candidates for reuse in modernization hence should not be selected. However, to complete the modernization process, the six components could be redeveloped and incorporated with others in the modernized version of the software.

Also, applying the technique to components in Applications B and C yields the component status and ranks as presented in tables 9 and 10 respectively.

Table 9: Ranking of Legacy Application B Components

Component Id	**Software Maturity Index (SMI)**				**Component Status**	**Rank**
	Ver. N-3	**Ver. N-2**	**Ver. N-1**	**Ver. N**		
B1	0.64	0.67	0.54	0.80	Irregular SMI increase in the last three	**Unstable**

					versions with the SMIs not tending to 1	
B2	0.38	0.91	0.92	0.92	Regular SMI increases with the SMI of the last three versions tending to 1	**Highly Stable**
B3	0.60	0.50	0.75	0.92	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
B4	0.67	0.60	0.75	0.85	Regular SMI increases with the SMIs not tending to 1	**Unstable**
B5	0.80	0.45	0.75	0.92	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
B6	0.50	0.33	0.75	0.93	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
B7	0.50	0.67	0.69	0.86	Regular SMI increases with the SMIs not tending to 1	**Unstable**
B8	0.17	0.57	0.89	0.90	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
B9	0.86	0.86	0.88	0.89	Regular SMI increases with the SMIs not tending to 1	**Unstable**
B10	0.00	0.33	0.64	0.75	Regular SMI increases with the SMIs not tending to 1	**Unstable**

From the above, A2 is highly stable; B3, B5, B6, B8 are stable while B1, B4, B7, B9 and B10 are unstable. For this legacy application B, component A2 which is highly stable together with B3, B5, B6 and B8 which are ranked as stable components could be selected for reuse in modernization because of their appreciable stability status. On the other hand, components B1, B4, B7, B9 and B10 being unstable are not good candidates for reuse, hence should not be selected rather they could be redeveloped with modern tools and incorporated with others.

Table 10: Ranking of Legacy Application C Components

Component Id	Software Maturity Index (SMI)				Components Status	Rank
	Ver. N-3	Ver. N-2	Ver. N-1	Ver. N		
C1	0.56	0.6	0.27	0.38	Irregular SMI increases with the SMIs not tending to 1	**Unstable**
C2	0.73	0.64	0.85	0.92	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
C3	0.60	0.33	0.56	0.95	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
C4	0.45	0.80	0.82	0.94	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**
C5	0.38	0.27	0.86	0.93	Regular SMI increases with the SMI of the most recent version	**Stable**

					tending to 1	
C6	0.50	0.33	0.94	0.94	Regular SMI increases with the SMI of the last two versions tending to 1	**Fairly Stable**
C7	0.33	0.13	0.50	0.93	Regular SMI increases with the SMI of the most recent version tending to 1	**Stable**

For legacy application C as indicated in Table 10, components C6 is fairly stable; C2, C3, C4, C5 and C7 are stable while C1 is unstable. Therefore, for the purpose of reuse in modernization, all components could be selected except C1 that is unstable. To this effect, Component C1 could be redeveloped with modern tools and incorporated into the system with others since it is not reusable.

9. CONCLUSION

In applying component-oriented reengineering in software modernization, the challenging issue has always been, how to identify and select stable components for reuse from a host of components that make up a legacy application. To this effect, this research was undertaken to understudy the characteristics of legacy components as they progress from one version to another in order to determine their level of stability and by extension suitability for reuse in modernization. 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.

With this technique, software professionals will be able to determine the stability status of components extracted from legacy applications and use same to guide their decision on whether a component should be reused in the application modernization or not. Moreover, components ranking of this manner is capable of providing the software engineer with some levels of confidence in the legacy components selected for reuse. Where some of the legacy components cannot be selected for reuse due to their low ranking in the scale, specifically those below the stable rank, such components could be redeveloped with modern tools and incorporated with others in the modernized software version.

10. RECOMMENDATIONS

Based on the findings of this research, the following recommendations are necessary:

a) Software professionals in charge of application modernization should always perform components reusability assessment on legacy components and rank them accordingly before they are selected for reuse. For legacy components stability assessment, the technique presented in this article is highly recommended.

b) Since software maintenance data is the major input for this technique, organizations should keep proper records of their software maintenance and provide same for components assessment whenever the need arises.

c) There should be deliberate efforts by researchers to conduct researches aimed at evolving models, tools and techniques suitable for legacy components assessment and selection for reuse in modernization, knowing fully well that legacy modernization will continue to remain a common phenomenon in the software industry, as today's modern application is tomorrow's legacy application and candidate for modernization.

11. REFERENCES

[1] Denoncourt, D. 2011. Approaches to ApplicationModernization. Scandinavian Developer Conference 2011 (SDC2011),Goteborg.Availableat:www.scandevconf.se/ 2011 accessed on: April 6, 2014.

[2] Cipresso, T. 2010. Software Reverse Engineering Education. Master's Theses and Graduate Research, San Jose State University. USA [online] Available http://scholarworks.sjsu.edu/etd_theses/3734 Retrieved on: March 5, 2012

[3] Mishra, S. K., Kushwaha, D. S., Misra, A. K. 2009. Creating Reusable Software Components from Object-orientedLegacy System through Reverse Engineering. Journal ofObject Technology, ETH Zurich. www.jot.fm/issues/issue_2009_07/article3.pdf Retrieved on: April 17, 2011

[4] Malinova, A. 2010. Approaches and Techniques forLegacy Software Modernization , Bulgaria ScientificWorks,37(2),UniversityofPlovdiv,Plovdiv,Bulgaria.www.fmi.uniplovdiv.bg/GetResource?id=402 Retrieved on: February 15, 2013

[5] Comella-Dorda, S., Wallnau, K., Seacord, R. and Robert, J2010. A Survey of Black-Box Modernization Approaches for Information Systems, Proceedings of InternationalConference on Software Maintenance pp. 173

[6] Saarelainen, M., Ahonen, J. J., Lintinen, H., Koskinen, J., Kankaanpaa, I., Sivula, H., Juutilainen, P. and Tilus, T. 2006. Software Modernization and Replacement DecisionMaking in Industry: A Qualitative Study. Available At:www.bcs.org/upload/pdf/ewic-ea06-paper.pdf Retrieved on: August 26, 2014

[7] Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., andHage, J. 2010. How Do Professionals Perceived Legacy Systems and Software Modernization? Utrecht University, Utrecht, The Netherlands. www.servicifi.files.wordpress.com/2010/06/icse.pdf Retrieved on: August 1, 2014.

[8] Gartner 2013. Gartner Survey Shows 75 Percent ofGovernment CIO Budgets Flat or Increase in 2013; Gartner ewsroom, 2013. Available at:

www.gartner.com/newsroom/id/2572815 accessed on: April 6, 2014

[9] Younoussi, S. and Roudies, O. 2015. All About Software Reusability: A systematic Literature Review; Journal of Theoretical and Applied Information Technologywww.jatit.org Retrieved on: September 10, 2015

[10] Fazal-e-Amin, Mahmood, A. K. and Oxley, A. 2011. AReview of Software Component Reusability Assessment Approaches; Research Journal of Information Technology3(1) pp. 1-11

[11] Jasmine, K. S. and Vasantha, R. 2010. A New CapabilityMaturity Model for Reuse Based Software Development Process; IACSIT International Journal of Engineering and Technology 2(1)

[12] Rine, D. C. and Nada, N. 2000. An Empirical Study of a Software Reuse Reference Model, Information and Software Technology Journal 42(1)

[13] Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M. and Kusumoto, S. 2004. Component Rank: Relative Significance Rank for Software ComponentsSearch. Available at;http://sel.ist.osaka-u.ac.jp/lap-db/betuzuri/archive/391.pdfRetrieved on: September 10, 2015

[14] Garcia, V,. Lucredio, D. and Alvaro, A. 2007. Towards a Maturity Model for a Reuse Incremental Adoption, Proceedings of Simposio Brasileiro de Componentes, Arguitetura e Reutilizacao de Software (SBCARS)

[15] Subedha, V. and Sridhar, S. 2012. Design of DynamicComponent Reuse and Reusability Metrics Library for Reusable Software Components in Context Level.International Journal of Computer Applications 40(9):30-34Available at: www.ijcaonline.org Retrieved November, 2, 2015

[16] Kessel, M. and Atkinson, C. 2015. Ranking Software Components for Pragmatic Reuse. 2015 IEEE/ACM 6th International Workshop. Available atwww.ieeexplore.ieee,org/xpl/articleDetails.jspRetrieved on: November 2, 2015

[17] IEEE 1988. Description of Software Maturity Index. IEEEStandards.www.standards,ieee.org/reading/ieee/std_public/description/982.1-1988_desc.html Retrieved on: September 10, 2015