A Systematic Approach to Stable Components Synthesis from Legacy Applications

Bassey Asuquo Ekanem

Delta State Polytechnic, Ozoro, Nigeria

Abstract— *Component-based software modernization involves the restructuring of a legacy application into its modernized version with better qualities and maintainability attributes using stable components extracted primarily from the legacy system itself. At the time of modernization, some of the legacy components may not be suitable for reuse due to their poor reusability attributes, especially component instability, yet their services and roles must be provided in the modernized version of the software. The option of creating stable components from the unstable counterparts through a systematic approach, though not widely practiced remains the best option over the development of new components from the scratch or use of COTs with full or partial experience to replace the unstable components considering the risk associated with each of these options. This research therefore, presents a technique called stable components synthesis technique aimed at supporting the process of creating stable components from unstable legacy components and reused in modernization.*

Keywords— Components Stability Assessment, Unstable Components, Stable components Synthesis, Software Modernization.

I. INTRODUCTION

The frequent technological changes and the dynamism of the environment in which businesses operate usually demand for regular modifications to the application software that run such business so as to extend their usable life. For a class of software called legacy applications which usage has spanned decades and is now aged usually presents some modification challenges due to their unique characteristics of 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 rely on these applications for their routine business operations and will not easily abandon or replace them; rather, will modernize them to remove the maintenance impediments and share in the benefits of running their businesses with modern applications (Mishra, 2009). In fact as reported by (Malinova, 2010), Comella-Dorda et al (2010); Saarelainen et al (2006) and Khadka et al (2010), legacy modernization is more profitable than outright replacement and application should only be replaced when it can no longer be evolved.

Legacy modernization is a process geared towards transforming aged application into its modernized version with better qualities and maintainability attributes. It has become a burning issue in the software industry due to the fact that the industry experiences at least 10% annual increase in legacy code (Denoncourt, 2011); a situation which demands for a systematic approach to dealing with such annual increase in other not to escalate the problems associated this class of system. Gartner (2012) CIOs survey report further affirms the critical nature of legacy modernization, with a report that places application modernization amongst the top 10 IT technology priorities of CIOs globally. Amongst all application modernization techniques namely, wrapping, migration, reengineering, and component-oriented reengineering, it is a known fact that component-oriented reengineering technique has 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, a complete legacy system will have both stable and unstable components where unstable components are considered unsuitable for reuse and will not be selected for modernization. Now the big question is, how do will make up for the unstable components that are not suitable for reuse in the modernized version of the application? Many options exist: development of new components with modern tools to replace the unstable components, use of COTs with full or partial experience, and creating of stable components from the unstable ones. Considering the high risk associated with the first two options, most organizations usually go with the third option since they are already familiar with those components save that they are unstable.

The task of creating stable components from unstable components is not an easy one. Furthermore, there are no specific models or techniques for stable components synthesis from their unstable counterpart, hence efforts in this regards most times are unsuccessful. Existing models mainly focus on components extraction and measures to ensure well-planned and

controlled reuse-oriented software development process in organizations (Caldiera and Basil, 1991; Misra, Kushawa and Mishra, 2009; Fazal-e-Amin et al, 2011; Jasmine and Vasantha, 2010, Younoussi and Roudies, 2015).

In view of the above, this research presents a technique on how stable components could be synthesized from the unstable legacy components and reused in modernization through a technique called stable components synthesis.

II. REVIEW OF RELATED RESEARCH WORKS

This section presents a review of literature related to this research. Caldiera and Basil (1991) proposed a clustering technique for identifying reusable components with cost, quality and usefulness as reusable factors which should be addressed by cyclomatic complexity, regularity, reuse frequency and code value matric. 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. In Inoue et al (2004) component rank model is presented using digraph-based system for computing and ranking software components selected for reuse.

Misra, Kushawa and Mishra (2009) presented a five step component identification method designed specifically for legacy software to include identification of systems components, construction of CRUD Matrix, Construction of message-call information graph of the classes, identification of clustering classes and identification of reusable components. In Garcia et al (2007) the RISE Maturity Model (RISE) is presented as a model designed to support organizations in improving their software development process with respect to components reusability assessment. The model 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 also specified.

Furthermore, in Fazal-e-Amin et al (2011), a review of software components reusability assessment approaches is 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.

Other relevant research works reviewed include the following: 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.

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. In Ekanem and Woherem (2015), a technique for the assessment of legacy components stability and components ranking is presented as a veritable tool to guide professionals on what components to select for reuse in modernization.

III. FINDINGS FROM THE REVIEW

The review of the related works clearly indicate that there exist numerous research works, techniques and models to support the process of components extracting from legacy applications and using same in build reusable components. Also, there exist some models and techniques to support components reusability assessment, classification and ranking for software development projects whereas there exist little or none for same processes in relation to legacy modernization. Simply put, most of the techniques and models reviewed focuses primarily on supporting reuse-oriented software development projects and ensuring that such projects are well-planned and controlled in organizations where they are applied.

The review also reveals a big gap in the research area in terms of lack of techniques, methods and models for dealing with the unstable components and their roles in the legacy application as the application evolves into its modernized version, Clearly, there exist techniques and methods for reusing stable components in modernization whereas there are no formal methods for dealing with the unstable counterparts; what happens to their roles in the modernized version of the software? How best could those roles be integrated into the modernized system assuming they are critical to routine business operations?

The absent of formal methods and techniques for dealing with unstable components during modernization is the reason why attempts to develop replacement for such components usually result in undesirable outcome. Moreover, the use of COTs for such projects are usually very risky since the stability of such components could not be guaranteed. However, for the unstable components being that the organization is already familiar with them, measures could be put in place to leverage on their strength while the weakness are addressed to restructure them into modernized components that are stable and suitable for reuse. There is a need to adapt existing models to address this gap or more appropriately new techniques and models could be evolved specifically to deal with unstable components in a more beneficial manner.

In view of the above, this research presents a technique on how stable components could be synthesized from the unstable legacy components and reused in modernization through a technique called stable components synthesis Technique.

IV. RESEARCH METHODOLOGY

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

i. Review of relevant documentations

ii. Randomization of legacy application maintenance 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 (legacy maintenance records) for the research were generated randomly using RANDBETWEEN function in spreadsheet program based on operands of Software Maturity Index (SMI), the model used in the research. 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 generated data were coded and analyzed using statistical package to generate results that were further interpreted and reported accordingly. The analysis was directed towards results that support the classification of legacy components into stable and unstable components. Thereafter, the proposed stable components synthesis techniqueis presented with discussions on how stable components could be synthesized from the unstable components.

V. DESCRIPTION OF SOFTWARE MATURITY INDEX (SMI) MODEL

Software Maturity Index (SMI) is needed to compute the stability and rankings of the legacy components; hence a brief description of the model is given in this section. Software Maturity Index, a metric in IEEE (1988), specifically IEEE 982.1-1988 was introduced to measure the maturity of software systems as 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. In this case, 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. A slight modification to the SMI model to fit into legacy component stability assessment follows thus:

i. modules as used in the model are replaced by legacy components of the application

ii. Maintenance data on each component in the recent versions of a legacy application used to compute the respective SMIs of each component.

iii. The result of such computation is further interpreted and used to determine components stability.

Legacy components assessment and ranking is based on this concept and uses maintenance data generated on a legacy application to demonstrate this concept.

VI. LEGACY COMPONENTS RANKING SCHEME

In Ekanem and Woherem (2015) legacy component ranking scheme is presented with the following items on the scale: Highly Stable, Fairly Stable, Stable, Unstable, Fairly Unstable and Highly Unstable. The criteria for this ranking are 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 itis characterized by regular SMI increases

In the last three application versions with the last two SMIs tending to 1

Stable: A component is said to be Stable if it is characterized by regular SMI increases in the last three application versions with the SMI 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.

VII. DATA SHEET AND DATA RECORDING CONCLUSION

Data needed to demonstrate components stability assessment and ranking are the maintenance data of legacy application software for some versions, at least the last four versions. The required data were generated randomly for the legacy application coded as legacy application A using 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 RECORDING

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

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 15based 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.

VIII. DATA PRESENTATION AND ANALYSIS

The maintenance data generated from the RANDBETWEEN function on a 15 component legacy application coded as Legacy Application Aare given below with the components labeled A1 to A15:

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

IX. SMI COMPUTATION AND ANALYSIS

The SMIs of each component in the application generated for the last four versions using appropriate formulae (i.e. SMI model) entered into the spreadsheet package as shown below:

TABLE 3
SOFTWARE MATURITY INDEX OF LEGACY 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

X. COMPONENTS ASSESSMENT AND RANKING

The 15 legacy components are assessed and ranked accordingly using the components assessment and ranking scheme explained earlier. 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 the benchmark for SMI tending to 1. This characteristic clearly fits the Fairly Stable rank hence component A1 can be said to be fairly stable. Similarly, for 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 said to be highly stable because the component is characterized by regular SMI increases where the SMIs of the last three versions tend to 1.

Furthermore, component A3 which SMIs are 0.58, 0.42, 0.73 and 0.87 is characterized by regular increases which do not tend to 1, hence A3 is said to be unstable. Similarly, for A10 with irregular SMI increases given as 0.73, 0.64, 0.45 and 0.69 is characterized by regular increases which do not tend to 1, hence A10 is also unstable. The table below shows the status and rank of all components in legacy application A obtained by applying the assessment and ranking scheme:

TABLE 4
STATUS AND 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 receding from 1	**Fairly Unstable**
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**

XI. RESULTS AND DISCUSSIONS

Table 4 clearly shows the following categories of components and ranks: highly stable component – A7; Fairly stable component – A1;stable components - A4, A6, A8, A9, A12, A13, A15; Unstable components – A3, A10 and Fairly Unstable Components – A2, A5, A11and A14.

The implication of the components assessment and ranking given above is that, components A1, A4, A6, A7, A8, A9, A12, A13 and A15 being components in the stable categories, although with variable degrees of stability could be selected for reuse in legacy modernization whereas componentsA2, A3, A5, A10, A11 and A14 being components with variable degrees of instability are not good candidates for reuse in modernization hence should not be selected. However, to complete the modernization process, stable components could be synthesized from the six unstable components through component-based reengineering process and incorporated with others in the modernized version of the software. The subsequent section presents the proposed Stable Components Synthesis Technique that could be used to create stable components from the unstable components.

XII. STABLE COMPONENTS SYNTHESIS TECHNIQUE

This section presents the Stable Components Synthesis Technique being proposed for used in creating stable components from their unstable counter parts in legacy applications. It is an attempt to fill the gap of lack of formal methods and techniques to create stable components from their unstable counter parts. The technique has the following steps:

i. **Components evaluation and Classification:** this has to do with the assessment of legacy components stability and their classification into stable and unstable components based on their SMI values computed from the legacy maintenance records. Also, the classified components are further ranked thus: stable components - highly stable, fairly stable, stable while for unstable components - unstable, fairly unstable and highly unstable using the component ranking scheme described earlier.

ii. **Unstable Components Evaluation:** this stage is key in the stable components synthesis technique as it is undertaken to identify the weak links in the unstable components and to propose measures need to address them. The weak links in this case are attributes of the unstable components that qualify them as such. In doing this, legacy maintenance records like faults reporting records, error detection and correction records must be reviewed. Code area and execution paths with frequent reported faults in the records are prime suspects.

iii. **Reverse Engineering of Unstable Components:** This has to do with the analysis of each of the legacy components to identify their roles/services. This could be supported by automated reverse engineering tool say Rational Rose which takes the legacy code as input and perform reverse engineering to generate its class diagrams.

iv. **Components Architecture Restructuring:** At this stage, the class diagrams generated during reverse engineering are understudied to further identify the use cases (functional dependencies), sequence diagrams (behavioral relationships) and class diagrams (structural relationships) which are essential ingredients in the forward engineering. To achieve good success, architectures of the prime suspects must critically examine to identify and remove or modify the offensive architectures.

v. **Forward Engineering:** this has to do with components code generation and testing. At this point, the program code for each of components is generated based on the restructured architecture generated earlier in step (iv) above. Modern tools like EJB, CORBA, COM+, .Net tools and the like could be used as applicable.

vi. **Components Integration and Testing:** at this stage, the stable components created from the unstable counterparts are integrated with other components and tested using testing tools like **Nunit** – a tool with graphical user interface application to execute all the test scripts and show their results as a success/failure ratio thereby making components testing easier.

vii. **Deployment of the Modernized System**: with the successful completion of the components integration process, the modernized application is deployed to the organization for use.

Relating this approach to the earlier demonstration with legacy application A, where the unstable components A2, A3, A5, A10, A11 and A14 from the application are subjected to the process of stable component synthesis presented above, they be transformed into stable components suitable for reuse in the modernization.

XIII. CONCLUSION

This research was designed to demonstrate how stable components could be synthesized from unstable components of a legacy application. It also illustrates how legacy components could be assessed and ranked into highly stable, fairly stable, stable, unstable, fairly unstable and highly unstable using the components assessment and ranking Scheme. The research

further emphasizes the fact that, where some or most of the extracted components from a legacy application are unstable (i.e. unstable, fairly unstable and highly unstable) such cannot hinder the modernization process from proceeding to its logical completion since such components could be synthesized into stable components suitable for reuse in the modernization.

To this end, a technique called stable components synthesis technique is presented for used in creating stable components from their unstable counterparts in a legacy application. With a careful and proper application of this technique, the quality of modernized application could be enhanced and the risk usually associated with the use of new components or COTs to replace the unstable components could also be minimized.

XIV. RECOMMENDATIONS

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

a) Software professionals in charge of application modernization should always adopt a systematic approach to creating stable components from the unstable counterparts rather than build new components from the scratch or use COTs which are highly risky. The stable components synthesis technique presented in this article is highly recommended for this purpose.

b) Also, legacy components reusability assessment and ranking should be performed to guide their professionals' decisions on choice of components for reuse in modernization.

c) Legacy software maintenance data should be properly kept as the software progresses from one version to another since this a major requirement for components stability assessment and ranking as well as stable components synthesis from their unstable counterparts.

d) There should be deliberate efforts by researchers to conduct researches aimed at evolving models, tools and techniques to support the creation of stable components from their unstable counterparts.

REFERENCES

[1] Caldiera, A. and Basil, C. 1991. Structural Analysis of Reusable Components, Proceedings of the Conference of the Future of Science and Engineering.

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

[3] Comella-Dorda, S., Wallnau, K., Seacord, R. and Robert, J. 2010. A Survey of Black-Box Modernization Approachesfor Information Systems, Proceedings of International Conference on Software Maintenance pp. 173

[4] Denoncourt, D. 2011. Approaches to Application Modernization. Scandinavian Developer Conference 2011 (SDC2011),Goteborg. Available at: www.scandevconf.se/2011 accessed on: April 6, 2014.

[5] Ekanem, B. A. and Woherem, E. (2015). Assessment of Components Stability for Modernization Using Software Maturity Index; International Journal of Scientific Research & Engineering Studies, West Bengal, India.

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

[7] 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)

[8] IEEE 1988. Description of Software Maturity Index. IEEE Standards.www.standards,ieee.org/reading/ieee/std_public/description/982.1-1988_desc.htmlRetrieved on: September 10, 2015

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

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

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

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

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

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

[15] Subedha, V. and Sridhar, S. 2012. Design of Dynamic Component Reuse and Reusability Metrics Library for Reusable Software Components in Context Level. International Journal of Computer Applications 40(9): 30-34 Available at: www.ijcaonline.org Retrieved November, 2,2015 Malinova, A. 2010. Approaches and Techniques for Legacy Software Modernization , Bulgaria Scientific Works, 37 (2), University of Plovdiv, Plovdiv, Bulgaria. www.fmi.uni-plovdiv.bg/Get_Resource?id=402Retrieved on: February 15, 2013

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

[17] Younoussi, S. and Roudies, O. 2015. All About Software Reusability: A systematic Literature Review; Journal of Theoretical and Applied Information Technology. www.jatit.orgRetrieved on: September 10, 2015.