Proposal of Automatic Methods for the Reuse of Software Components in a Library

Koffi Kouakou Ive Arsene[1], Samassi Adama[2], Kimou Kouadio Prosper[3], Brou Konan Marcellin[4]
Ecole Doctorale Polytechnique, Institut National Polytechnique (INP-HB)
Yamoussoukro, Côte d'Ivoire

***Abstract*—The increasing complexity of applications is constraining developers to use reusable components in component markets and mainly free software components. However, the selected components may partially satisfy the requirements of users. In this article, we propose an approach of optimization the selection of software components based on their quality. It consists of: (1) Selecting components that satisfy the customer's non-functional needs; (2) Calculate the quality score of each of these candidate components to select; (3) Select the best component meeting the customer's non-functional needs with linear programming by constraints. Our aim is to maximize this selection for considering financial cost of component and adaptation effort. Yet in the literature review, researchers are unanimous that software components reuse reduces the cost of development, maintenance time and also increases the quality of the software. However, the models already developed to evaluate the quality of the component do not simultaneously take into account financial cost and adaptation effort factors. So, in our research, we established a connection between the financial cost and the adaptation time of the selected component by a linear programming model with constraints. For our work's validation, we propose an algorithm to support the developed theory. User will then be able to choose the relevant software component for his system from the available components.**

Keywords—Method development; reuse; software component; quality of component; functional size; functional processes; financial cost; adaptation effort

I. Introduction

The increasing size of applications and the accretion of their complexity pose enormous challenges for developers. To solve these problems, they must have to recourse to reusable components in their applications. However, selected components may not totally meet the requirements of users. Moreover, there may be functionality defects of these software components or quality services partially rendered by the ones. then, their selection and reuse require the development of appropriate models and methods. In addition, several works relating to the selection of reusable software components have been conducted. And researchers are unanimous on the fact that the reuse of these software components reduces the financial cost, the development time and the effort of adaptation [5], [6], [7]. In [7], the researchers proposed a software component selection model based on integer linear programming. This method makes it possible to measure and evaluate the quality of the software system according to various quality attributes defined in ISO 9126 / IEC and the cost of the components. In [13], the authors worked on the selection of software components based on the attributes or quality criteria most important to practitioners. This survey allowed practitioners to select the most important attributes from a list of factors. The method showed that cost was the most important factor when selecting these components. In [24], based on an exploratory study, researchers have shown that in addition to the cost considered as the most important factor in the selection, other factors such as longevity, compatibility and in charge of the component exist. Their goal is to study the most important factors in a list when selecting components for practitioners. Then to hierarchize them. This study helps companies improve their component selection process. They concluded that small businesses focus on properties associated with ease of use, component development and maintenance, while larger firms and more mature products are more interested in cost-related properties. However, we find that the dependence between financial cost and maintenance time that are the main factors for the selection process, is not considering in the different models of evaluation for denoting the quality of software components. In this research, we will propose automatic methods for:

- Facilitating and accelerating the selection process;
- Evaluate the quality of selected software components according to the criteria and quality indicators desired by the user;
- Selecting the best component satisfying the client's non-functional needs;
- Improving the quality of these softwares to adapt them to the targeted problem.

This work is organized as follows. The first part deals with Section 1. It concerns the state of the art relating to the selection of reusable components, the limits of previous work and research hypotheses. The second part concerns Section 2. It is about different models that we have developed. The third part concerns the validation of the results in Section 3. The last part concerns the conclusion and the perspectives.

II. State of the Art

Several research works relating to the selection of reusable software components have been made. In [1] and [2], the authors have shown that traditional approaches for developing software from scratch are not optimal for building complex software systems. They argue that the use of reusable software components is more efficient and better suited for building complex applications. In [3], the authors proposed the so-called "Storyboard" approach. This method improves and facilitates the choice of customer for appropriate commercial products as their requirements are better understood. His interest is to help the user better understand his requirements. Other selection studies based on surveys and experiments have been conducted. Thus, in [4] an empirical study led on the selection of commercial components. Thus, researchers in [4] led an empirical study on the selection of commercial components. They conducted structured interviews on 16 software projects. This method allowed to customize the development process based of COTS software components. The goal is to know if it is more interesting to build the software components or buy the Cost components for the Norwegian industries. In [8], the research has proposed a method for selecting standard and commercial components. It raises the problem of inadequacy between the software system to be built and the components selected during and after selection. They proposed a decision-support approach aimed at remedying the imbalances noted on the components by estimating the anticipated aptitudes and by suggesting alternative plans for the resolution of the observed disparities. The authors in [9] offer a comparative study of available software before any selection. The goal is to evaluate and select open source software for the management of electronic and digital medical records. This study is carried out with different decision-making techniques multi-criteria. These software systems are selected on the basis of a set of metric results using the AHP technique integrated with different multicriteria decision-making techniques.

In [21], the authors use a software selection approach based on the characteristics of the ISO-9126 standard. The AHP method is used to weight these characteristics of components. Then, the researchers choose the appropriate software component according to the weight evaluation.

In [10], a mechanism allowing the automation of the selection of a software component among a set of candidates according to their functional and non-functional properties was studied. This mechanism permits to facilitate the extraction and the comparison of components. This is after the selection of components, to measure their satisfaction index to find the most relevant. To optimize the quality of selected components, several models and selection methods have been developed and are available. Among these models, some are focused on optimization algorithms. Thus in [11], the researchers proposed a software component selection approach based on the genetic algorithm for optimizing the performance of the software system. Their goal is to maximize the functional performance of the system. This permits to maximize cohesion and to minimize the coupling of software modules for the optimal selection of software components. In [23], the research focused on optimizing the system to build. Researchers have conducted work on selecting optimized software components when user requirements are unclear. it is a question of optimizing the selection in the generic applications unknown to the developers.

The authors in [5] have proposed a model for the selection of components with constraint optimization. The goal is to model the component selection problem as a constraint satisfaction optimization problem. In addition to the quality criteria determining the choice of attributes of quality of the component, other important factors are identified in the literature. These factors can also affluence the quality of the components when selecting. Therefore, authors sustain that the use of reusable software components reduces the time, cost of development and cost of maintenance [5], [6], [7], [20], [22], [25].

In [25], the authors propose in this work, how to select the best component in a repository meeting all functional requirements and user requirements. The best components are recovered in two levels. The first step gives all the components that correspond to the functional requirements, and the second step recommends the components the weighting is the highest to software developer.

In [12], the work focused on the problem of optimizing non-functional attributes when selecting software components. The method consists in choosing software components that provide all the necessary functionalities while optimizing certain non-functional attributes such as the financial cost. In [7], the researchers proposed a software component selection model based on integer linear programming. This so-called flexibility method makes it possible to measure and then evaluate the quality of the software system according to different attributes of quality and the cost of the components. In [13], the authors conducted work on the selection of software components based on the attributes or quality criteria most important to practitioners. This survey allowed practitioners to select the most important attributes from a list of factors. The method showed that cost was the most important factor when selecting these components.

In [14], authors argue that "the quality and cost of a software strongly depend on the quality and cost of the components assembled to produce the product". They proposed a W-shaped model for component selection. This model is a decision support tool for software developers. It permits to obtain data on the stages of component selection and the development process. The article [15] gives different mathematical models of optimization in linear programming. One of these models is a compromise between the minimum monetary cost and the response time in cloud computing. It is formulated below:

$$\begin{cases} minimize(a * T + (1 - a) * C \\ with\ the\ contraints\ defined \\ \quad C{:}\ \ cost\ model \\ T{:} sight\ reponse\ time \end{cases} \quad (1)$$

III. Research Problem

A. Hypotheses

The work that we present treats with the problematic of the evaluation of the quality of the pre-made components. It concerns the maximization of their calculated quality values while optimizing the financial cost and the adaptation time. Our goal is therefore to determine a score based on linear programming with constraints that will maximize the quality of the selected software component. Then we will balance the financial cost and the adaptation time of this component. Finally, we establish a model based on a score to evaluate the quality of the selected software component on the one hand,

and moreover, to predict the adaptation effort of this component.

This leads us to formulate the following hypothesis:

H1: The simultaneous consideration of the financial cost and the adaptation effort makes it possible to better evaluate the quality of the software component,

H2: The selection of reusable and user-friendly software components makes it possible to build quality software.

B. Limit of Methods

Several works relating to the selection of reusable software components have been conducted. Researchers are unanimous that the reuse of these software components can reduce the financial cost, the development time and the effort of adaptation [5], [6], [7], [23]. However, we find that the dependence between the financial cost and maintenance time that are key factors for the selection process, is not taking into account in the different models of quality evaluation of software components. Indeed, the selected components can meet the expectations of the users partially. Faced with failures and user requirements, improvements can be made to correct weaknesses and increase the quality of these components. Indeed, the selected components can partially meet the expectations of users. Faced with failures of certain functionalities and user requirements, improvements can be made to correct weaknesses and increase the quality of these components. This can generate a maintenance effort and a financial cost that can be estimated and predicted. Finally, we can give a model for optimizing parameters.

C. Tool to Predict the Adaptation Time of the Component

To estimate maintenance time and adaptation effort, we will use methods and tools to measure the size of the software component. We used the Cosmic v4.0.1 method and its methods in our work. Below you will find some tools for estimating the development time and their normalization histories in Table 1.

From 1970s, the COCOMO method (Constructive Cost Model) has made it possible to determine the code lines of the programs and to measure the development effort. At present, methods and tools exist to estimate the size of a software and predict the development effort. In [16], the authors gave a summary of these tools with the different standards (see Table 1). The COSMIC method is used to calculate the measurement of the functional size of a software. According to [17], [18] and [19], functional size measurement is a means of determining the size of software, regardless of the technology used to implement it. This size is in units of Cosmic Function Points, noted as PFC. This method also gives the estimate of the adaptation effort. In [16], researchers present measurement aggregation rules. These rules make it possible to calculate.

TABLE I. TIME ESTIMATION TOOL

Sigles	Denominations	ISO standards
FISMA	Finish Software Measurement Association	29881
NESMA	Netherlands Software Metrics Association	24570
Mk II FPA	Function Point Analysis Mk II	20968
COSMIC	COmmon Software Measurement International Consortium	19761

- The functional size of each process i

$$size(functional\ process) = \sum size(Input)_i + \sum size(Output)i + \sum size(Reading)i + \sum size(Writting)i \quad (2)$$

- The size of a software by aggregating the sizes of its functional processes under certain conditions,
- Development effort or adaptation effort

IV. PROPOSED APPROACH

A. Defining the Software Component Quality Model

We are interested in evaluating the selection and integration of software components in a software system. Our main objective is to select the "best software component" according to the defined characteristics. But given the multiplicity of quality indicators and quality sub-indicators according to ISO / IEC 9126, we studied the following characteristics in our work. These characteristics include: functional capability, reliability, ease of use, security and maintainability. This allows us to define the following model[1]:

This model is based on the ISO 9126 quality model and quality representations of literature reviews. It allows to specify the most important characteristics according to the needs of the user. Using the Analytic Hierarchy Process (AHP) method, we define the objective of our project and then construct the hierarchical quality model according to the characteristics and sub-characteristics of the software components (see Fig. 1).

Finally, using the multi-criteria analysis method, we constructed a binary comparison table of characteristics and sub-characteristics. This makes it possible to determine the weights of the various defined quality criteria of the software component. Also, this method makes it possible to evaluate the coherence of our work.

[1]Quality model, inspired by the ISO 9126 model and the software quality defined by Jéremie Grodziski

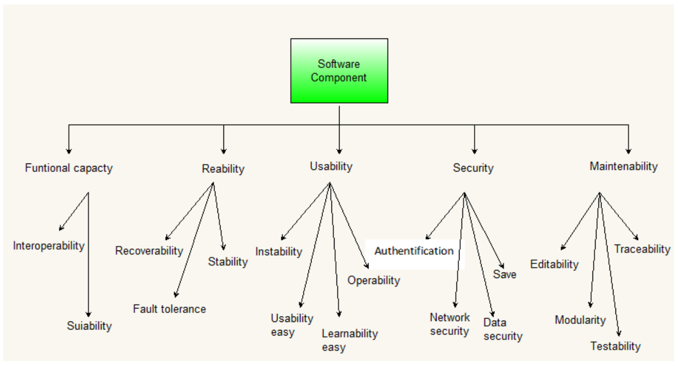

Fig. 1. Hierarchical Structure Indicating the Quality of the Software Component.

B. The Proposed or Software Component Selection Process

We gave a description of the selection process of the selected components and then we evaluated them. This process is modeled in UML by activity diagram as follows according to Fig. 2:

Step 1: The user expresses its functional requirements and quality requirements of the component.

Step 2: A first search consists in considering the functional properties expressing the needs of the user. These needs must be related to the type of software to build. We obtain a set of software components selected functional properties meeting the requirements expressed by the customer. In other words, it is the different services rendered by the software components.

Step 3: This step consists to make selection based on non-functional properties. This is to consider the quality of the software component that is, how the features render the services. This step consists in evaluating the quality of characteristics of the component from defined metrics. This metric will be associated with an ordinal variable of modalities belonging to the set of values:

$$B = \{Bad, Insufficient, Average, Good, Excellent\} \quad (3)$$

Modalities defined in (3) will be associated to following numerical values respectively: 1; 2; 3; 4 and 5.

Step 4: At this step, we observe that the selected components do not fully meet the quality and service requirements. For each component selected *i*, some features make the services perfectly, others do it partially. if we consider that each component contains *p* functionalities. Assuming that the user is satisfied with *k* functionalities (k <p), then we must maintain (*p-k*) functionalities of the component. To predict the adaptation effort of (p-k) functionalities, we used the Cosmic method. It first determines the size of the functional processes of the component. Then we calculate the functional size of the component with defective functionalities. In [16], the authors defined the size of the functional process *i* as follows according to (2). So, for any component *i* of the set of selected components SC having *P* functional processes, we deduce:

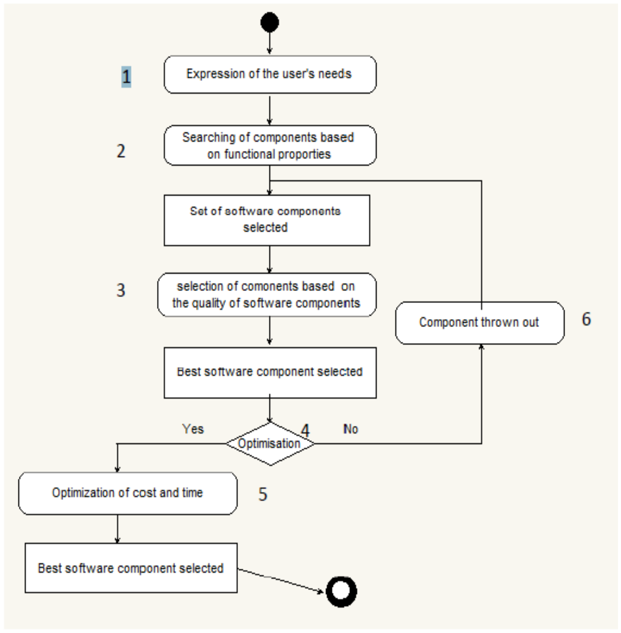
1
Expression of the user's needs
2
Searching of components based on functional proporties
Set of software components selected
3
selection of comonents based on the quality of software components
Best software component selected
Component thrown out
6
Yes
4
No
Optimisation
Optimization of cost and time
5
Best software component selected

Fig. 2. Software Component Selection Process.

$$functional\ size\ of\ a\ component\ (i) = \sum_{j}^{p} size\ of\ functional\ procss(\ \ i, j)$$

$$\forall\ i \in Sc\ and\ 1 \leq j \leq P \quad (4)$$

Then we apply the estimate of the adaptation effort developed according to [19]

$$Estimated\ Development\ effort = Component\ Size * Unit\ Cost \pm Predictive\ interval \quad (5)$$

This phase makes it possible to determine the adaptation time interval of the component to be predicted. This method then evaluates a financial cost and an adaptation time. Finally, with the predicted time, we apply the score that assesses the quality of the component using our objective function.

Step5: In case the cost and time parameters are optimized, then the selected component is retained.

Step 6: If the parameters are not, then the search continues and the process resumes.

C. Our Proposal Model to Maximize the Quality of Software Component

Our model is based on constrained linear programming. It considers the time and the financial cost parameters. Our goal is to define a metric with two parameters: the financial cost and the time. This score serves to optimize the parameters on the one hand and on the other hand to balance the financial cost coupling and the adaptation time.

We define our function as follows:

$$f(c,t) = aC_i + (1-a)t_i \ \forall i \in Sc \tag{6}$$

$$and \ 0 \le a \le 1 \ and$$

$$t_i \in [\![t_{min}; t_{max}]\!] \tag{7}$$

$$with \ constraints$$

$$0 \le a \le 1$$

$$t_i = \frac{t_{i_{rel}}}{T_{max}} \ and \ 0 \le t_i \le 1$$

$$C_i = \frac{C_{i_{rel}}}{C_{max}} \ and \ 0 \le C_i \le 1 \tag{8}$$

Where

Sc: set of available components
C$_i$: Standardized cost of maintenance of the component i
C_{i_rel}: relative cost generated by component i;
C_{max}: maximum cost achieved by one of the selected components;
t$_i$: Standardized adaptation and maintenance time of the component i
t$_{i\_rel}$: Relative time, generated by component i;
$T_{max}$is the maximum time achieved by one of the selected components;
a: Coefficient of adaptation

By taking inspiration from the model (1) and the metric developed in [7], we are able to define a new score to evaluate the quality of the software component. So, our model for any software component i selected will be:

$$S_i = \sum_{h \epsilon A} w_h \, q_{hi} x_i - [ac_i + (1-a)t_i]x_i \tag{9}$$

$$and \ \forall \ i \in Sc \tag{10}$$

$$with \ constraints$$

$$0 \le a \le 1$$

$$t_i = \frac{t_{i_{rel}}}{T_{max}} and \quad 0 \le t_i \le 1 \quad and \tag{11}$$

$$T_{max} = 15 \ jours = 1.296 * 10^3 \text{s}$$

$$C_i = \frac{C_{i_{rel}}}{C_{max}} and \quad 0 \le C_i \le 1$$

$$andC_{max} = 2300 \ \$ \tag{12}$$

Where

A: set of software quality characteristics;
SC: set of available components (candidate components);
q$_{hi}$: the standard level of the quality attribute h∈ A for component *i*;
W$_h$: weight attributed to the quality attribute h∈A;
x$_i$ = 1 if component i is selected, 0 otherwise;
C$_i$: standardized cost of component *i*;
C_{i_rel}: relative cost generated by component *i*;
t$_i$: Standardized component maintenance time;
t$_{i\_rel}$: Relative time, generated by component *i*;
a: Adaptation coefficient to be specified

Model (9) represents the objective function. This function is used to calculate and evaluate the quality of the characteristics of the selected software components. For optimizing the parameters Time and maintenance cost, we maximize the objective function.

For any software component *i* of the library, we obtain the following system:

$$\begin{cases} maximize(\sum_{h \in A} W_h q_{hi} x_i - [aC_i + (1-a)t_i]x_i \ \ \forall i \in Sc \\ 0 \le a \le 1 \\ t_i = \frac{t_{i\_rel}}{T_{max}} \ and \ 0 \le t_{i\_rel} \le T_{max} \\ C_i = \frac{C_{i\_rel}}{C_{max}} \ and \ 0 \le t_{i\_rel} \le C_{max} \\ q_{hi} = \frac{q_{hi\_rel}}{Q_{max}} \ and \ 0 \le q_{hi_{rel}} \le Q_{max} \\ x = 1 \ selected \ else \ x = 0 \end{cases} \tag{13}$$

We will then be able to compare and order the different values designating the quality values of each selected software component.

V. Validation Phase

In the field of research, any theory must go through an experimentation or simulation phase before its validation. To do so, we propose an algorithm to support and validate the developed theory. It evaluates the quality of software component. It is also optimizing the two parameters including the adaptation time and the financial cost. Indeed, we propose the algorithm "SelectCompo" to solve the problem.

A. Presentation of our Algorithm

The algorithm SelectCompo aims to select in a set of available components (C*d*), the optimized and selected component (C*os*). See algorithm Fig. 3.

SelectCompo Algorithm

```
1.  Input: Set of available components (Cd)
2.  Output: Optimized component and selected (Cos)
3.  Begin
4.  While (needs and requirements expressed in Cd) do
5.      For i= 1 to Component (Cd) do
6.          Select (the component Ci)
7.          Put in the list of selected components (Cs)
8.      Endfor
9.  EndWhile
10. If ((conditionsCaracterisks Filled) and (cost and relative
      time in intervals required) then
11.     For i= 1 to ComponentCs do
12.      evaluate (thequality value of the selected
         components)
13.    If (SatisfactionQuality)then
14.       Optimize (the factors of cost and time of adaptation)
15.     Select ( the component(Cos))
16.  else choose another component in the set Cs
17.   end if
18. End
```

Fig. 3. Pseudo Code of SelectCompo.

B. Algorithm Operation

The operation of the algorithm Fig. 3 traces the following steps:

The algorithm takes as input the set of **p** available components (**Cd**) of a library. The user defines his functional requirements and non-functional quality requirements. These requirements are the quality attributes related to the type of software system to be built. The list (**Cs**) of i components verifying the conditions is fulfilled (with k <p). The next step is to evaluate the quality of the components of the list (**Cs**) by binary comparison of their characteristics. Then we maximize their quality value by the linear programming by constraints model that we developed. This step produces two (2) results. An ordered list of components is obtained. We retain the best (**Cos**). The best component is the better optimized. it will be selected. In the opposite case we take back the selection in the list (Cs).

VI. Conclusion and Perspectives

This article presents an automatic method for selecting relevant software components from a library. The methods used are based on an optimization algorithm and a linear programming by constraints. They made it possible to calculate and evaluate the quality of the software components. By maximizing our model, the selected components are ranked. This makes it possible to choose the most relevant component according to the quality criteria of the attributes defined by the customer. This approach is sustained by the SelectCompo algorithm that we defined. In future works, we will do experimentations with the Cplex Studio IDE 12.8.0 optimization tool for selecting the best component in a set of candidate components. Several aspects remain to be developed. This is taking into account the selection of software components in various libraries for any platform. This will solve the problem of interoperability of these components on different platforms.

References

[1] Dellarocas, C., (1997), The SYNTHESIS Environment for Component-Based Software Development‖, Proc. 8th Int. Workshop on Software Technology and Engineering Practice (STEP‟97), London, UK, (July 14-18, 1997), IEEE Computer Society, ISBN 0-8186-7840-2, Washington, DC, USA, Page 434.

[2] Gaurav Kumar, “Optimized Component Development Life Cycle for Optimal Component-Based Software Development », Research Scholar, Punjab Technical University, Kapurthala, India, 2015

[3] Gregor, S., Hutson, J. and Oresky, C “Storyboard process to assist in requirements verification and adaptation to capabilities inherent in cots”. In Proceedings of 1st International Conference on COTS-Based Software Systems, Springer-Verlag Lecture Notes in Computer Science, 132

[4] Li, J. et al. An Empirical Study of Variations in COTS-based Software Development Processes in Norwegian IT Industry. Proc. of the 10th IEEE Intl. Metrics Symposium 72-83, 2004

[5] A. Vescan, H. F. Pop, The Component Selection Problem as a Constraint Optimization Problem, Proceedings of the Work In Progress Session of the 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques (Software Engineering Techniques in Progress), Wroclaw University of Technology, Wroclaw, Poland, 2008, pp. 203-211.

[6] Tom Wanyama, Agnes F. N. Lumala, « Decision Support for the Selection of COTS »,In Proceedings of the Canadian Conference on Electrical and Computer Engineering, 2005

[7] Pande, CJ Garcia, D Pant, “Optimal Component Selection for Component Based Software Development using Pliability Metric”, ACM SIGSOFT Software Engineering Notes, January 2013

[8] Mohamed, A., Ruhe, G. and Eberlein, A. 2007. Decision support for handling mismatches between cots products and system requirements. In Proceedings of the Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (Washington DC, USA,2007

[9] A.A. Zaidan, B.B. Zaidan, Ahmed Al-Haiqi, M.L.M. Kiah, Muzammil Hussain, Mohamed Abdulnabi “Evaluation and selection of open-source EMR software packages based on integrated AHP and TOPSIS , University of Malaya, 50603 Lembah Pantai, Kuala Lumpur, Malaysia ,2015

[10] Bart George, R. Fleurquin, S. Sadou, H. Sahraoui « Un mécanisme de sélection de composants logiciels », juillet 2010

[11] Kwong, C.K., Mu, L.F., Tang, J.F. and Luo, X.G.2010. Optimization of software components selection for component-based software system development. Comput. Ind. Eng., 58,4. (May 2010). 618 – 624

[12] Khan, Ali M. and Mahmood, S., (2010),―Optimal Component Selection for Component-Based System, Innovation in Computer Science and Software Engineering‖, Innovations in Computing Sciences and Software Engineering, DOI 10.1007/978-90-481-9112- 3_79, Sobh, Tarek, Elleithy, Khaled(eds.), Springer

[13] Panagiota Chatzipetrou, Emil Alégroth, Efi Papatheocharous, Markus Borg, Tony Gorschek,Krzysztof Wnuk, « Component selection in Software Engineering - Which attributes are the most important in the decision process?”, 2018

[14] Vinay, Manoj Kumar and Prashant Johri,” W-Shaped Framework for Component Selection and Product”, Development Process SCSE, Galgotias University, Noida, India, 2014

[15] Romain Perriot*, Jérémy Pfeifer*, Laurent d’Orazio*, Bruno Bachelet*, Sandro Bimonte**, Jérôme Darmont***, « Modèles de Coût pour la Sélection de Vues Matérialisées dans le Nuage, Application aux Services Amazon EC2 et S3 », *Clermont Université, CNRS, Université Blaise Pascal, LIMOS UMR 6158, , p.15, archives ouvertes, 2014

[16] Alain Abran, “The COSMIC Functional Size Measurement Method Version 4.0, Measurement Manual”, (The COSMIC Implementation Guide for ISO/IEC 19761: 2011), 2014

[17] C. Gencel, “How to Use COSMIC Functional Size in Effort Estimation Models?”, in Software Process and Product Measurement, Springer Berlin Heidelberg, 2008.

[18] Sylvie Trudel, mesure de la taille fonctionnelle avec la méthode cosmic (iso 19761): recherches récentes et applications industrielles » , conférence du latece 2012

[19] Cosmic : mesure de la taille fonctionnelle avec la méthode, https://info.uqam.ca/midi-confs/2017-02-22-cosmic.pdf

[20] NSadana, S Dhaiya, MS Ahuja, “A Metric for Assessing Reusability of Software Components”, International Journal of Computer Application, Issue 4, Volume 1, February 2014;

[21] Sofiane Batata « Moteur de recherche pour la sélection de composants logiciels » Ecole Nationale Supérieure d’Informatique (Ex. INI), 2011

[22] siham younoussi, ounsa roudies, « all about software reusability: a systematic literature review », Mohammed-V Agdal University, Journal of Theoretical and Applied Information Technology, . Vol.76. No.11, 2015

[23] G. Kumar, « Optimized Component Development Life Cycle for Optimal Component-Based Software Development”, International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) ,Vol 2, Issue 12, December 2015

[24] P. Chatzipetrou, E. Alégroth, E Papatheocharous, M. Borg, Tony Gorschek, Krzysztof Wnuk,” Component selection in Software Engineering - Which attributes are the most important in the decision process?”, 2018

[25] Sumit Sharma Upinder Kaur[b], Pawanpreet Kaur[c,] , a.b,c Chandigarh University,Mohali, India, “Component Recommender System Based on Collaborative Approach in Incremental Development”, 2017.