International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:11, 2014

Estimation of Component Reusability through Reusability Metrics

Aditya Pratap Singh, Pradeep Tomar

***Abstract*—**Software reusability is an essential characteristic of Component-Based Software (CBS). The component reusability is an important assess for the effective reuse of components in CBS. The attributes of reusability proposed by various researchers are studied and four of them are identified as potential factors affecting reusability. This paper proposes metric for reusability estimation of black-box software component along with metrics for Interface Complexity, Understandability, Customizability and Reliability. An experiment is performed for estimation of reusability through a case study on a sample web application using a real world component.

***Keywords*—**Component-based software, component reusability, customizability, interface complexity, reliability, understandability.

I. Introduction

THE software reuse through components has been present in software engineering for several decades. It has been started way back in 1969 with the work presented by McIlroy on reusable components [1]. According to reuse the component belonging to one application can be used in development of other application having different functionality. The reusability is one of the effective ways to improve productivity. Reusable software components are intended to apply the power and benefit of reusable, interchangeable parts from other applications to the field of software development. Other industries have also profited from reusable components like reusable electronic components are found on circuit boards. The main objective of reuse in context of Component-Based Software (CBS) is to reduce various overheads like cost, duplication of work, time of implementation, efforts and to enhance standard compliance and reliability of the system [2] [3]. There are some issues [4] which are associated with reuse of components like increased maintenance costs, not-invented-here syndrome, creating and maintaining a component library, finding, understanding and adapting reusable components [5]. In spite of these issues, component reuse attracts due to various benefits in the development of new application by using reusable components. Reusability has an important role in CBS Architecture. Reusability can measure the degree of features/components that are reused in building similar or different new software with minimal change.

For Component-Based Software Development (CBSD), there are two broad reuse development approaches. One is the development of systems with reuse and another is development of components for reuse [6]. In the first approach, the application is developed by reusing a number of already built in components. Such components are already tested thoroughly and enhance the quality of the concerned product and will save time and cost. In the other approach, the components are developed in a way to keep them more reusable. This paper considers first approach and studies the methods of finding reusability of target components before integrating with the new application.

Reusability is one of the quality attributes of CBS. It is not easy to measure quality attributes of software directly because various quality-attributes may be affected by many factors and there is no standard method to weigh them. This paper proposes metrics for estimation of one of the quality attributes of black-box component i.e. reusability.

In the literature, many metrics are available to measure the quality of component, but there is very less work on the framework that makes use of these metrics to find reusability of software components [5]. The paper is focused on estimating the reusability of black-box component using metrics.

The paper is organized as follows. In Section II we briefly surveyed a substantial literature on component reusability. Some of the approaches of component reusability estimation and assessment are reviewed. The paper then identifies four attributes for reusability estimation and metrics for identified attributes are proposed that are to be used for reusability estimation. In the next section, a reusability estimation process is presented. A metric for reusability estimation is also proposed using metrics for identified attributes. In Section IV a case study is presented to illustrate the use of metrics on a real world component used in a sample web application. This section describes the process of estimating reusability of a component using the proposed model. Section V contains the conclusion and future work.

II. Component Reusability

A physical replaceable part of a system that adds functionality to the system, through the realization of a set of interfaces is called reusable component. The components having well defined interfaces can be considered good for reuse. The interfaces have strong significance in context of reusability of components. An interface contains a collection of operations, which are used to access a service of the component. All platforms supporting CBS architecture like COM+, CORBA and EJB use interface as the glue that binds component together [7].

The components are being reused at different design levels having different reusability probabilities in an application and

Aditya Pratap Singh and Pradeep Tomar are with the School of Information and Communication Technology of Guatam Buddha University, Greater Noida, Uttar Pradesh, India 201308 (e-mail: adityapsingh@gmail.com, parry.tomar@gmail.com).

they can be classified according to reusability levels as shown in Fig. 1. The components at application level have more reusability than a context/Domain level component. The component at foundation/Base level has least reusability. This paper refers to application level components.

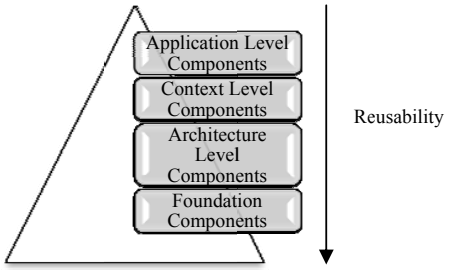

Fig. 1 Reusability Hierarchy

Component reusability is a quality attribute that refers to the expected reuse potential of a software component. Component reuse not only improves productivity, but it also puts a positive effect on quality and maintainability of software products [8]. Reusability minimizes the redundant component development and also the implementation time. In systematic reuse the components are particularly constructed to be reused in future applications and are more likely to be easy and safe for reuse [9].

With the consideration of the presence of software components, there have been several efforts undertaken to discuss the topic of component reusability e.g. [3], [5], [8], [10], [11] with some research direction [12].

A. Approaches of Component Reusability Estimation

Component reusability is the degree to which a component can be reused [11], and reduces the software development cost by enabling less coding and more assembly. The key issues with respect to developers are the selection of component with higher reusability and which one is most reusable among components with similar specifications. The reusability estimation becomes necessary in order to realize the reuse of components effectively.

For systematic process of software reuse, the use of metrics is essential. Without metrics, it is difficult to evaluate the quality of components selected for reuse and cost saving due to reuse [13]. Various models and metrics [6], [11], [14], [15] are documented in the literature that try to assess the degree of component reuse in a CBS. Some of the measures are concerned with estimating the actual financial benefits due to reuse. Such as Barns and Bollinger [16] suggested analytical approaches to making good reuse investments, based on cost-benefit analysis. The programmers find software reusability a useful approach. The experimental results from [17] confirm that the reusability prediction is possible, but it involves more than the set of metrics being used. Proposed reusability metrics, as well as the empirical study, are concerned with measures of reusability potential of black-box components. In this section we will review some of the available reusability metrics.

Washizaki et al. [11] proposes a metrics suite to measure various non-functional attributes like understandability, adaptability, portability and reusability of components. The proposed metrics suite is based on confidence intervals. These interval values were set with the help of statistical analysis of Java Bean components. Authors normalized all the measurement values in the metrics to a number between 0 and 1. The metrics they proposed are:

- EMI: Existence of Meta-Information, its confidence interval is between 0.5 and 1.

$$EMI = \begin{cases} 1, & (If\ BeanInfo\ class\ exists) \\ 0, & (Otherwise) \end{cases} \quad (1)$$

- RCO: Rate of Component Observability; its confidence interval is between 0.17 and 0.42.

$$RCO(c) = \begin{cases} \frac{P_r(c)}{A(c)}, & A(c) > 0 \\ 0, & (Otherwise) \end{cases} \quad (2)$$

where: $P_r(c)$: number of readable properties in component c; $A(c)$: number of fields in c's Facade class

- RCC: Rate of Component Customizability, adaptability is considered as an important factor that depends on customizability. Its confidence interval is between 0.17 and 0.34:

$$RCC(c) = \begin{cases} \frac{P_w(c)}{A(c)}, & A(c) > 0 \\ 0, & (Otherwise) \end{cases} \quad (3)$$

where: $P_w(c)$: number of writable properties in component c.

- SCCr: Self-Completeness of Component's Return Value, confidence interval is between 0.61 and 1.

$$SCCr(c) = \begin{cases} \frac{B_v(c)}{B(c)}, & B(c) > 0 \\ 1, & (Otherwise) \end{cases} \quad (4)$$

where: $B_v(c)$: number of business methods without return value in component c; $B(c)$: number of business methods in component c.

- SCCp: Self-Completeness of Component's Parameter, confidence interval is between 0.42 and 0.77.

$$SCCp(c) = \begin{cases} \frac{B_p(c)}{B(c)}, & B(c) > 0 \\ 1, & (Otherwise) \end{cases} \quad (5)$$

where: $B_p(c)$: number of business methods without parameters in component c.

Reusability metrics were based on a reusability model. However, the architectural and system domain constrains were not considered while these constraints may affect overall measurements.

Boxall and Araban [18] have proposed that software component's interface is a major quality factor for determining reusability. To better understand Interface they have defined a set of metrics including size of interface and argument count.

- APP: Arguments per Procedure, The average number of arguments in publicly declared procedures (within the interface).

$$APP = \frac{n_a}{n_p} \quad (6)$$

where: n_a : total count of arguments of the publicly declared procedures; n_p : total count of publicly declared procedures.

- DAC: Distinct Argument Count, the number of distinct arguments in publicly declared procedures

$$DAC = |A| \quad (7)$$

where: A: set of <*name, type*> pairs representing arguments in the publicly declared procedures; |A|: number of elements in the set A.

- DAR: Distinct Arguments Ratio, the percentage of DAC in the component interface.

$$DAR = \frac{DAC}{n_a} \quad (8)$$

where: n_a: total count of arguments of the publicly declared procedures.

- ARS: Argument Repetition Scale, it aims to account for the repetitiveness of arguments in a component's interface.

$$ARS = \frac{\sum_{a \in A} |a|^2}{n_a} \quad (9)$$

where: A: set of the <*name, type*> pairs representing arguments in the publicly declared procedures; |a|: count of procedures in which argument name-type 'a' is used in the interface; n_a: argument count in the interface.

They validated their metrics on the selected components and develop a tool to automatically calculate them.

Rotaru and Dobre [19] studied Adaptability, Composability and Complexity of individual components as determinants for their measure of reusability. The multiplicity of a software component can be used to measure composability. The multiplicity of a component C (μ_c) is defined as:

$$\mu_c = \sum_{i=1}^{n} m_i \quad (10)$$

where: n is the number of interface methods in C.

The multiplicity of an interface method M:

$$m_M = C_r \times m_r + C_s \times m_s \quad (11)$$

where: m_r : return multiplicity; m_s : signature multiplicity; C_r and C_s : constants.

They stated that adaptability of a component is not only influenced by internal factors but also by the adaptability of the architecture.

$$A = A_C + A_F \quad (12)$$

where: A_C: component's adaptability; A_F: adaptability of the framework.

The complexity k of a software component C can be expressed based on its multiplicity (10):

$$k_C = \lim_{x \to \mu_c} \left(\frac{\varepsilon . x}{x+1} \right) \quad (13)$$

where: ε is a constant (0,1].

Gill [8] discussed the variety of issues regarding component reusability. Author listed some important guidelines to improve the level of software reusability in CBSD. Author suggested to software reuse practicing organizations for conducting thorough and detailed assessment of software reuse to get maximum benefit in terms of cost and time.

Sandhu and Singh [15] propose an approach based on metric for identification of a reusable software module. The reusability estimation was done with the help of Fuzzy Logic and Neuro-Fuzzy technique. The study carried out by the authors shows the use of metrics for identification of quality of a software component.

Kumar [20] used Support Vector Machine (SVM) for classification of reusability of software components. The various available metrics like Cyclomatic Complexity Using McCabe's Measure, Halstead Software Science Indicator, Regularity Metric, Reuse-Frequency Metric, and Coupling Metric were used for identification of reusable software modules.

Gui and Scott [21] measured the reusability of Java components retrieved from the internet by using a set of new proposed static metrics for coupling and cohesion.

Yingmei et al. [24] considered reusability as a factor that depends on functionality, reliability, utilizability, maintainability and portability. For component reusability assessment they proposed Reusability Measure Value (RMV) metric:

$$RMV = W_1 * F + W_2 * R + W_3 * U + W_4 * M + W_5 * P \quad (14)$$

where: W_i(i=1,...,5): weights; F: functionality; R: reliability; U: utilizability; M: maintainability; P: portability.

Koteska and Velinov [25] performed a critical review on various existing component reusability metrics. The authors suggested two new attributes security and installability to be included as additional conditions when evaluating component reusability. According to the authors the final score of the security class of a component can be calculated using these equations:

$$F_{i,j=MIN(D_{i,j,k}),\ k \geq 1} \quad (15)$$

$$O_{i=MIN(F_{i,j}),\ j \geq 1} \quad (16)$$

$$C = \sum_{i=1}^{N} W_i O_i \quad (17)$$

where: C: A class; O: A security objective; F: A security function; W: the percentage weight of an objective; Di, j, k: the scores of the dependencies k of the security function j of security objective i.

Installation flexibility metric as follows:

$$X = \frac{A}{B} \tag{18}$$

where: A: number of implemented customizable installation operation; B: number of customizable installation operation required.

The authors expressed component reusability as:

$$CR = \sum_{i=1}^{N} W_i \, M_i \tag{19}$$

where: N: number of attributes that will be considered as important in the measurement process; W_i : weight factor for i-th attribute; M_i: metric that measures i-th attribute.

B. Identified Factors Affecting Software Reusability

The literature lists various characteristics of software components, which are believed to determine reusability and are therefore repeatedly referenced in literature [6] [10], [11], [18], [19], [22], [24], [25]. Some of the factors are: adaptability, complexity, composability, maintainability, modularity, portability, programming language, quality, reliability, retrievability, utilizability, size, documentation quality, understandability, security and installability. Based on the researchers and practitioners view, we have identified following four factors for the reusability estimation of black-box components (Fig. 2):

- Interface Complexity
- Understandability
- Customizability
- Reliability

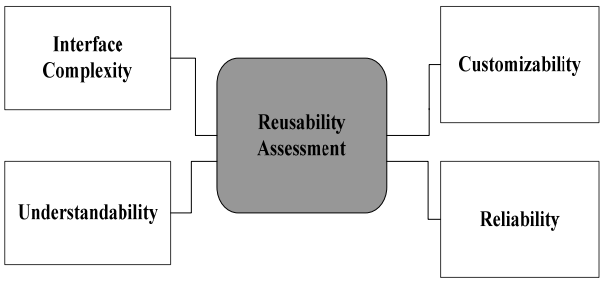

Fig. 2 Identified factors for reusability estimation

1) Interface Complexity

Components interact with other components through their well-defined interfaces. The interfaces act as a primary source of information to understand, use, implementation and maintenance to the component. Interfaces may contain information like inputs, outputs, operations and exceptions. The component interface complexity provides an estimate of the complexity due to interfaces of components. The lower value of interface complexity leads to better reusability of component.

Interface complexity of a component can be qualitatively defined by analyzing the parameters and return values of its interface methods [19]. The interface methods with no parameters and no return value have least complexity because it does not have any external data dependencies. The interface methods having some return value, but no parameters can be considered as middle level complexity. Whereas interface methods with both parameters and return values have highest complexity.

Interface complexity (IC) of a component can be calculated as:

$$IC = \frac{\sum_{i=1}^{n}(rw_i + \sum_{j=0}^{k} aw_{ij})}{n} \tag{20}$$

where: n: number of interface methods available for the component; k: number of arguments in ith method; rw_i: weight for return value type of ith method; aw_{ij} : weight for jth argument type of ith method.

The weight values can be assigned on the basis of the data type of return value and formal arguments of the component interface method. The weight for no or void type has been assumed 0.01. All other weight values depending on data types are represented in Table I.

TABLE I
WEIGHT VALUE ASSIGNED TO DIFFERENT CATEGORIES OF DATA TYPES

Category of data type	Weight value
Primitive data type	0.10
Derived data type	0.20
User defined type/ object type	0.30

2) Understandability

According to ISO/IEC 9126 [26] understandability can be defined as the capability of the component to enable the user to understand whether it is suitable and how it can be used for particular tasks and conditions of use. Component understandability depends on how much component information is provided for functional description and how well it is documented [27]. The cohesiveness between component document and component functionality is important for understandability. If the design of the component and language of the documents is closely related then understandability is high and the user will make fewer efforts to know the functionality of directly used the services of component. For better reusability, understandability of a software component should be as high as possible.

The component documentation contains component descriptions, demos, API's, test procedures and tutorials. These documentation attributes have direct impact on component understandability.

For documentation, practitioners also use Component Registry i.e. a fully searchable XML (Extensible Markup Language) document for component documentation. Such XML document covers a range of reusable JavaBean, Enterprise JavaBean and Component Object Model (COM) components.

A presence type metric can be used to measure such attributes. The metrics EMI (1) and RCO (2) proposed by Washizaki et al. [11] can be adapted to assess understandability of the component. These two metrics may help component

users to understand its behavior. The EMI (1) is only concerned with Java Beans, which can be changed to check also for other UML or other metadata objects for component service and context.

$$EMI = \begin{cases} 1, & (If\ BeanInfo\ or \\ & other\ metadata) \\ 0, & (Otherwise) \end{cases} \quad (21)$$

The simplest way to combine two metrics is to calculate the average of two. The Understandability of Component (UC) can be calculated as:

$$UC = \frac{\text{EMI+RCO}}{2} \quad (22)$$

3) Customizability

The ability to be customized is called Customizability. The customizability of a component represents the available writable properties within exterior side classes of a component. The components can provide customizable features to enhance reuse spectrum. A component should be customizable during the integration to adjust itself into specific requirements. But more writable properties can be used wrongly. Component's customizability effects on reusability of components in CBSD [10].

The component customizability can be evaluated by adapting Rate of Component Customizability (RCC) metric (3) defined by Washizaki et al. [11].

4) Reliability

The reliability is the ability of a system or a component to perform its required functions under stated conditions for a specified period of time [23]. Reliability is the probability that the system will perform as intended over a specified time interval. The high reliability of a component does not guarantee the high reliability of CBS. The CBS reliability is estimated using the reliability of the individual components and their interconnection mechanisms [2]. It is still argued that the reliability plays an important role when reuse of pre-existing software component is performed.

In CBS the components may be black-box, independently deployable components. To evaluate the reliability of a component, there could be some reliability results from the component developer. But, that result is obtained under component developer's environment and assumptions. Therefore, the results may not match those for the component user's environment. Before integration the components are tested separately. During or after this testing of component, its reliability can be measured.

The black box component reliability can be estimated using [28]:

$$R = 1 - \lim_{n\to\infty} \frac{f_i}{n_i} \quad (23)$$

where: R: reliability of component; f_i: number of failures of component i; n_i: number of executions of component i in N randomly generated test cases.

The next section contains proposed metric for reusability estimation using metrics for identified attributes.

III. Estimation of Reusability

In order to improve quality, flexibility and development productivity of a software application, practitioners relies on reusable software components. The reusability is one of the nonfunctional requirements of CBSD. Assessing nonfunctional requirement is always a tedious task.

To measure reusability, the relative significance of individual identified attributes that influence component reusability is weighted proportionally.

To calculate Component Reusability (CR) of a black-box software component, first the individual identified attributes of measurement model have to be quantified through metrics specified in section II then these metrics are aggregated to estimate black-box component. Based on metrics for identified attributes in section II, the Component Reusability of a black-box component can be estimated as follows:

$$CR = w_1 \times (1 - IC) + w_2 \times UC + w_3 \times RCC + w_4 \times R \quad (24)$$

where: w_1- w_4 are weights and others are metrics for identified attributes for reusability estimation.

The equation uses (1-IC) as the interface complexity should have lower value for higher reusability. To facilitate the comparison of different black-box component reusability these values should be normalized to a specific range [0...1]. The weights are used for the relative importance of attributes for measurement of component reusability and can be decided empirically. The weights can be influenced by the domain constraints and may have a relative importance in different application domain. Due to the normalization, the sum of weights has to be 1.

IV. A Case Study for Estimation of Reusability

This section applies the metrics presented above using one case study. The case study presents a real world component, Apache.Commons.FileUpload [29]. It is an independently released component as a part of the Apache Commons project [29]. The FileUpload component provides robust, high performance form based file upload facility to web applications. The FileUpload component has totally 41 classes. This component has six interface methods out of which four are constructors. Total ten fields are in the ServerFileUpload and DiskFileItemFactory classes. Out of these ten properties five are readable and three are writable properties. To test this component a very simple JAVA web application is created using CodeEnvy [30] (shown in appendix A). Various types of files including multiple compressed files were uploaded using the application to test the component. All the files were successfully uploaded.

A. Measurement of IC

The component FileUpload has ServletFileUpload and DiskFileItemFactory as its façade classes. Two methods parseParameterMap() and parseRequest() are available as

interface methods. Two constructors per class are also used to create objects of ServletFileUpload and DiskFileItemFactory classes. The interface methods return a List object and have HttpServletRequest object as argument. Based on this dataset the Interface Complexity (IC) of FileUpload component is calculated using (20) and the weights as per Table I.

$$IC = ((0+0.01) + (0+0.30) + (0 + 0.01) + (0+ (0.10 + 0.30)) + (0.30 + 0.30) + (0.30 + 0.30)) / 6 = 0.32$$

B. Measurement of UC

The FileUpload component has javadoc and other proper documentation attached to it. Therefore, the value of EMI metric should be 1. The ServletFileUpload class has total nine fields inherited from its parent classes. The four properties are readable in this class. The DiskFileItemFactory has only one property which is readable as well as writable. As per (2) the value of RCO metric will be 0.5. With values of EMI and RCO the understandability of FileUpload Component (UC) is calculated using (22).

$$UC= (1+0.5) / 2 = 0.75$$

C. Measurement of RCC

The facade classes of FileUpload component have total ten properties out of which three are writable. The value of RCC metric is calculated using (3).

$$RCC= 3/10= 0.30$$

D. Measurement of Reliability

The reliability of this component depends on the test results in user environment. For testing of FileUpload component ten test cases are prepared for various aspects like file size, file name, file type, file location.

The developed sample web application for testing of this component successfully passed all the test cases. Hence the reliability for FileUpload in the context of sample web application is 1.

The calculated metric values for FileUpload component are given in Table II.

TABLE II
CALCULATED VALUES OF METRICS FOR FILEUPLOAD

Metrics	Value
IC	0.32
UC	0.75
RCC	0.30
R	1

E. Measurement of Reusability:

To calculate reusability of FileUpload component, the relative weight value for each attribute is required. The weight values are determined based on researchers and practitioner's accumulated knowledge about the relative importance of identified attributes. A set of weight values decided empirically for some of the situations is given in Table III.

The component reusability for each weight value set is calculated using (24).

CR for weight value set S1 is:

$$CR_{S1} = (0.3 * 0.68) + (0.1 * 0.75) + (0.2 * 0.3) + (0.4 * 1) = 0.739$$

TABLE III
DATASET OF WEIGHT VALUES

Weight value set	Weight Value
S1	(0.3,0.1,0.2,0.4)
S2	(0.4,0.1,0.2,0.3)
S3	(0.5,0.1,0.2,0.2)
S4	(0.3,0.2,0.2,0.3)

Similarly the value of CR metric for other weight value sets S2, S3 and S4 can be calculated. The results are given in Table IV.

TABLE IV
COMPONENT REUSABILITY VALUES

Weight value set	Component Reusability (CR)
S1	0.739
S2	0.707
S3	0.675
S4	0.714

In this way the reusability of a component can be measured based on identified attributes. The weight values for the metric (24) can be adjusted as per the component type and the context of its usage.

V. CONCLUSION

The paper has surveyed current approaches of component reusability estimation and assessment. Some of the available approaches were presented. Based on these approaches, the paper identified four attributes as a part of presented reusability metric to estimate the reusability of a black-box component. The reusability metric is parameterized by following attributes: component interface complexity, understandability, customizability and reliability.

The paper presents metrics for calculating values for identified attributes. A proposed metric for component interface complexity is presented and validated along with other attribute metrics by calculating their values for the FileUpload component of the Apache Commons project. The metric for reusability is a composition of these four sub metrics. The proposed reusability metric is used to estimate reusability value of FileUpload component. However, this work further requires validation. In future the weight values for the estimation of reusability can be adjusted using neural network.

APPENDIX

A: Partial Code for JAVA Web Application

This code is a part of sample web application to test FileUpload component of Apache Commons project. The code contains a standard way of using a component in a servlet and is developed with the help of Codenvy [30] an online developer environment.

FileUploadServlet.java

```
import java.io.File;
```

```
import java.io.IOException;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
public class FileUploadServlet extends HttpServlet {
private static final long serialVersionUID = 1L;
private static final int THRESHOLD_SIZE = 1024 * 1024 * 3; //
3MB
private static final int MAX_FILE_SIZE = 1024 * 1024 * 40; //
40MB
private static final int REQUEST_SIZE = 1024 * 1024 * 50; //
50MB
private List<FileItem> fileItem = null;
private String __filePath =
this.getClass().getClassLoader().getResource("../../").getFile();
protected List<FileItem> initRequest(HttpServletRequest req) {
    boolean isMultipart =
ServletFileUpload.isMultipartContent(req);
    if(!isMultipart) throw new UnsupportedOperationException();
    DiskFileItemFactory factory = new DiskFileItemFactory();
    factory.setSizeThreshold(THRESHOLD_SIZE);
    factory.setRepository(new
File(System.getProperty("java.io.tmpdir")));
    ServletFileUpload upload = new ServletFileUpload(factory);
    upload.setFileSizeMax(MAX_FILE_SIZE);
    upload.setSizeMax(REQUEST_SIZE);
    List<FileItem> formItems = null;
    try {
        formItems = upload.parseRequest(req);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return formItems;
}

protected File uploadFile(List<FileItem> formItems, String
destFolder)
{
    String uploadPath = __filePath+destFolder;
    File uploadDir = new File(uploadPath);
    System.out.println(uploadDir.getAbsolutePath());
    if (!uploadDir.exists()) {
        uploadDir.mkdir();
    }
    File uploadedFile = null;
    try {
        for(FileItem fi : formItems ){
            if (!fi.isFormField()) {
                String fileName = new File(fi.getName()).getName();
                String filePath = uploadPath + File.separator +
fileName;
                uploadedFile = new File(filePath);
                fi.write(uploadedFile);
            }
        }
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    return uploadedFile;
}
protected String getFieldValue(List<FileItem> formItems, String
fieldName)
{
    String value = null;
    try {
        for(FileItem fi : formItems )
        {
            if (fi.isFormField())
            {
                if(fi.getFieldName().equals(fieldName))
                {
                    value = fi.getString();
                }
            }
        }
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    return value;
}
protected void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException
{
    fileItem = initRequest(request);
    String description = getFieldValue(fileItem,
"inputDescription");
    File file = uploadFile(fileItem, "uploads");
 request.setAttribute("path",file.getAbsolutePath());
    request.setAttribute("description", description);
    RequestDispatcher rd =
request.getRequestDispatcher("/success.jsp");
    rd.forward(request, response);
}
}
```

REFERENCES

[1] Mojica, I. J., Adams, B., Nagappan, M., Dienst, S., Berger, T., and Hassan, A. E. “A Large-Scale Empirical Study on Software Reuse in Mobile Apps”. *IEEE Software*, vol. 31, no. 2, pp 78-86, 2014.

[2] Singh, A. P. and Tomar, P., “A new model for Reliability Estimation of Component-Based Software”, in *Proc. IEEE 3rd International Advance Computing Conference* (IACC), pp. 1431-1436, Feb. 2013.

[3] Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B., and Irlbeck, M. “On the extent and nature of software reuse in open source Java projects”. *In Top productivity through software reuse Springer Berlin Heidelberg*. pp. 207-222, 2011.

[4] Sommerville I., *Software Engineering*, 9th edition, Pearson Education; 2013. ISBN: 978-9332518858

[5] Sandhu, P.S., Kakkar, P. and Sharma, S., “A survey on Software Reusability”, in *Proc. 2nd International Conference on Mechanical and Electrical Technology (ICMET)*, pp.769-773, 2001.

[6] Sharma, A., Grover, P. S., and Kumar, R., “Reusability assessment for software components”. *ACM SIGSOFT Software Engineering Notes*, vol.34, no. 2, pp. 1-6, 2009.

[7] Booch, G., Rumbaugh, J., and Jacobson, I., *The unified modeling language user guide*. Pearson Education India, 2011. ISBN: 978-81-7758-372-4

[8] Gill, Nasib S., "Importance of Software Component Characterization for Better Software Reusability", *ACM SIGSOFT Software Engineering Notes*, vol. 31 No. I, pp. 1-3, 2006.
[9] Schach, S. R., *Object-oriented and classical software engineering*, ed. 8. McGraw-Hill, 2011. ISBN: 0071081712, 9780071081719
[10] Sharma, A., Kumar R. and Grover, P.S., "A critical Survey of Reusability Aspects for Component- Based Systems", *World Academy of Science, Engineering and Technology*, Vol.19, pp. 411-415, 2007.
[11] Washizaki, H., Yamamoto, H., and Fukazawa, Y., "A metrics suite for measuring reusability of software components", in *Proceedings of IEEE Ninth International Software Metrics Symposium*, pp. 211-223, 2003.
[12] Frakes, W.B. and Kyo K., "Software Reuse Research: Status and Future", *IEEE Trans. Software Eng*., vol. 31, issue 7, pp. 529 – 536, 2005.
[13] Leach, Ronald J. *Software Reuse: Methods, Models, Costs*. AfterMath, 2012. ISBN: 0985368519, 9780985368517
[14] Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., and Schach, S. R. "Evaluating software reuse alternatives: a model and its application to an industrial case study", *IEEE Trans. Software Eng*, vol. 30, no 9, pp. 601-612, 2004.
[15] Sandhu, P. S., and Singh, H., "Automatic reusability appraisal of software components using neuro-fuzzy approach", *International Journal Of Information Technology*, vol. 3, no. 3, pp. 209-214, 2006.
[16] Barns, B. H., and Bollinger, T. B., "Making reuse cost-effective", IEEE *Trans. Software*., vol. 8, issue 1, pp. 13-24, 1991.
[17] Schach S. R. and Yang X., "Metrics for Targeting Candidates for Reuse: An Experimental Approach", *ACM, SAC*, pp 379-383, 1995.
[18] Boxall, M. A. and Araban, S., "Interface metrics for reusability analysis of components", in *Proceedings of IEEE Australian Software Engineering Conference*, pp. 40-51, 2004.
[19] Rotaru, O. P. and Dobre, M., "Reusability metrics for software components", in *3rd ACS/IEEE International Conference on Computer Systems and Applications*, pp. 24, 2005.
[20] Kumar, A., "Measuring Software Reusability using SVM based Classifier Approach". *International Journal of Information Technology and Knowledge Management*, vol. 5 no. 1, pp. 205-209, 2012.
[21] Gui, G. and Scott, P. D., "Coupling and cohesion measures for evaluation of component reusability", in *Proceedings of ACM International workshop on Mining software repositories*, pp. 18-21, 2006.
[22] Gill, N. S., "Reusability issues in component-based development", *ACM SIGSOFT Software Engineering Notes*, vol. 28 no. 6, pp. 30, 2003.
[23] *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard glossaries.* IEEE Press, 1991.
[24] Yingmei, L., Jingbo, and S., Weining, X., "On Reusability Metric Model for Software Component", in: Wu, Y. (eds.) Software Engineering and Knowledge Engineering. LNCS, Vol. 114, pp. 865-870. Springer, Heidelberg, 2012.
[25] Koteska, B. and Velinov, G., "Component-Based Development: A Unified Model of Reusability Metrics", in *ICT Innovations*, pp. 335-344. Springer Berlin Heidelberg, 2013.
[26] *ISO/IEC 9126: Software engineering - Product quality - Part 1: Quality model. International Organization for Standardization and International Electrotechnical Commission,* 2001.
[27] Gao, J., "Component Testability and Component Testing Challenges", in *Proceedings of International Workshop on Component-based Software Engineering* (ICSE2000), 2000.
[28] Goseva-Popstojanova, K., Mathur, A. P. and Trivedi, K. S., "Comparison of architecture-based software reliability models", in *Proceedings of IEEE 12th International Symposium on Software Reliability Engineering* (ISSRE2001). pp. 22-31, 2001.
[29] Apache Commons FileUpload library (online), (http://commons.apache.org/fileupload/)
[30] Codenvy integrated development environment (https://codenvy.com/ide)