International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(3): 519-533
The Society of Digital Information and Wireless Communications, 2011 (ISSN: 2220-9085)

# Reusability Assessment of Open Source Components for Software Product Lines

Fazal-e-Amin, Ahmad Kamil Mahmood, Alan Oxley
Computer and Information Sciences Department, Universiti Teknologi PETRONAS,
Bandar Seri Iskandar, 31750 Tronoh, Perak, Malaysia.
fazal.e.amin@gmail.com, {kamilmh, alanoxley}@petronas.com.my

## ABSTRACT

Software product lines and open source software are two emerging paradigms in software engineering. A common theme in both of these paradigms is 'reuse'. Software product lines are a reuse centered approach that makes use of existing assets to develop new products. At the moment, a motivation for using open source software is so as to gain access to source code, which can then be reused. The product line community is being attracted to open source components. The use of open source software in software product lines is not for one time reuse but, being a core asset, the component is intended to be used repeatedly for the development of other products in the family. In this paper the results of an exploratory study is presented; it was conducted to explore the factors affecting the reusability of open source components. On the basis of the results of the exploratory study a reusability attribute model is presented which makes use of established object oriented metrics accompanied with newly defined metrics. The assessment using the proposed metrics is compared with the rankings assigned by human evaluators.

## KEYWORDS

Software metrics, measurement, reusability, mixed method, interview

## 1 INTRODUCTION

Software reuse reduces development time, effort, cost and increases productivity and quality. Studies in software engineering confirm these benefits[1] and [2]. Software reuse in its most common form can be seen in component based software development. Software product lines (SPLs) are a systematic way of using components. An SPL can be defined as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"[3]. An SPL provides an infrastructure for systematic software reuse. The software development scene has been greatly influenced by the emergence of open source software (OSS) components. The availability of OSS is far better today than it was in the past; this is because of component search engines. Furthermore, the Code Conjurer tool, as described in [4], has elevated code reuse to a high level.

The motivation behind the two emerging paradigms of SPLs and OSS is reuse. SPL development can benefit from OSS. It is more usual to start an SPL with some assets already in place, in other words SPLs are seldom started

from scratch. These initial assets could be OSS.

The form of reuse in an SPL differs from that in traditional reuse. It is systematic reuse, whereas traditional reuse is ad hoc. In an SPL an asset is developed by reuse and is developed for reuse. The latter concept of development for reuse sets out new requirements for the asset.

The Software Engineering Institute has defined a framework for SPLs that states that software enters the organization in three ways: built in-house; commissioned from a third party; purchased from a vendor by having licensed user rights, as in the case of open source or web services. The inclusion of open source as an asset and part of product line infrastructure is already envisioned by the community, for instance a model for an open source based product line is presented in [5] and a COTS based product line concept can be found in [6-7]. In line with this vision, while including an open source component in a product line asset base it is necessary to measure its reusability. The measurement of reusability helps to make a decision and a comparison of the different components providing the desired functionality.

This paper has three contributions; first is the report on the partial results and process of an exploratory study conducted to explore the factors affecting the reusability of open source components in a product line environment. Second, is the proposal of a reusability attribute model and third is the implementation/validation of the results obtained by using the model and metrics. In terms of methodology this paper is based on mixed methods, both a qualitative research method (interview) and quantitative research methods (survey; experiment) are used during the research.

## 2 SOFTWARE REUSABILITY

Software reusability refers to the probability of reuse of software [8]. In [9] software reusability is defined as the "characteristics of an asset that make it easy to use in different contexts, software systems, or in building different assets". The potential benefits of software reuse and the maturity of reusability concepts leads us to think about how we might measure them [8].

In software measurement, three kinds of entities are measurable - processes, products, and resources [10]. A product can be defined as any artifact developed as a result of process activity. These entities may have attributes which are of two kinds - internal and external. An external attribute is one that cannot be measured directly. In contrast, internal attributes can be measured directly. If we can measure something directly then this means that we can measure it independently. Relevant metrics are termed 'direct metrics'[11]. For example, the size of a program can be measured directly in several ways: by counting the number of lines of code; by counting the number of 'methods'; etc. In software engineering measurement terminology, a metric is a quantitative indicator of a software attribute; a metrics model specifies relationships between metrics and the attributes being measured by these metrics. The topic of measurement, with respect to reuse, covers six areas: modeling cost/benefits; assessing maturity; assessing the amount of reuse; identifying the failure modes; identifying the reusability metrics; identifying a library of reusability metrics [12].

A reusability assessment approaches review reveals that none of the approaches considers variability to

assess reusability [13]. In the context of SPLs it is not viable to assess the potential for reuse without considering the capacity of the component to provide variability. The importance of variability is reflected in the literature. Variability and commonality are the central concepts of SPLs. Systematic reuse is made possible by introducing variability into the core assets. The fundamental concept of variability is presented in [14], whilst types of variability are discussed in [14] and [15]. Variability implementation mechanisms are discussed in [16] and [17]. A synthesis of the literature on variability types and implementation mechanisms is provided in [18] and [19]. It gives a description of variability implementation mechanisms and relates the mechanisms with the types of variability and their scope.

In object-oriented programming a 'class' is a basic unit of encapsulation that facilitates its reuse. In [20] Szyperski's notion of a component, given in [21], is mapped to a class, and it follows that in the context of object orientation a class can be said to be a component because it is the only unit of composition.

## 3 FACTORS AFFECTING REUSABILITY OF OSS

The nature of the study to explore the factors affecting the reusability of OSS in a product line environment demands the use of an exploratory research method, and to serve this purpose interviews are used as the tool to collect data. A literature review was conducted prior to this study that confirms that the factors are as yet unexplored [13].

The interview is a means of collecting primary data; it is a conversation between two persons, one of which is a researcher. Interviews can be used for data collection where the nature of the study is exploratory. Interviews are helpful when the data to be gathered is about a person's knowledge, preferences, attitude or values [22]. Interviews may help to gather impressions and opinions about something. Interviews enable one to get personalized data, provide an opportunity to probe, establish technical terms that can be understood by the interviewee, and facilitate mutual understanding. The interview provides an in-depth view. Interviews are best for exploring the perspective of informants [22]. In the context of this study the informants are those who have experience with open source and product lines and preferably have academic/research experience. The authors have contacted several people and managed to conduct interview sessions with five informants. A brief introduction of them is presented in table-1.

The results are obtained using the grounded theory approach [23]. Open, axial and selective coding is performed to get meaningful results. The results are divided into different categories. The details of the study cannot be presented here due to space limitations. However, the results relevant to this paper are presented here. The category that this paper is concerned with is factors affecting reusability of OSS in an SPL environment.

Table 1: Information about the respondents

| Respondent ID | Experience | Experience Type | Current Affiliation |
|---|---|---|---|
| *Rsp-A* | 05 years | Academic, Industrial | Academia |
| *Rsp-B* | 10 years | Academic, Industrial | Industry |
| *Rsp-C* | 22 years | Academic, Industrial | Industry |

| | | | |
|---|---|---|---|
| *Rsp-D* | 08 years | Academic, Industrial | Academia |
| *Rsp-E* | 10 years | Academic, Industrial | Academia |

Table 2: Means used to conduct interviews

| **Means used** | ***Number of Interviews*** |
|---|---|
| **Skype** | 01 |
| **Face to face** | 03 |
| **Telephone** | 01 |
| **Total** | 05 |

The results are obtained using the grounded theory approach [23]. Open, axial and selective coding is performed to get meaningful results. The results are divided into different categories. The details of the study cannot be presented here due to space limitations. However, the results relevant to this paper are presented here. The category that this paper is concerned with is factors affecting reusability of OSS in an SPL environment.

The following factors relating to documentation are identified: Flexibility; Maintainability; Portability; Scope Coverage; Stability; Understandability; Usage History; Variability. Documentation is one of the factors that affect the reusability of an OS component. Documentation has an influence on the understandability of a component. In the case of open source the importance of documentation increases because of the numerous contributions to the code by different developers. The analysis of code is difficult without having the documentation.

Flexibility is related to reusability in two capacities. First it is the ability of a component to be used in multiple configurations. Second, it is a necessary attribute concerning future requirements and enhancements.

Maintainability is related to reuse in terms of error tracking and debugging. If the component is maintainable it is more likely to be reused. In cases where OSS components are running on systems connected to others system then a bug is particularly problematic. Sometimes debugging a component on one configuration may not work on other configurations. On the other hand in black box reuse maintainability is not considered a factor of reusability.

Portability is considered a factor in the sense that a cohesive component is more potable. A component having all the necessary information within it or having less interaction with another module during its execution is more reusable. Again in the case of black box reuse it is not a factor.

Another characteristic of the open source components explored is that the developer looks for a component covering more of the scope of the application. In some situations even the size does not matter but size is a concern in large sized components as it relates to increased complexity and poor understandability.

Furthermore, scope coverage is important in situations where future enhancements are already envisioned or there are chances that more features would be added in future.

The interviewees consider stability as an important factor to be considered while making decisions. Here, the term ‘stability’ refers to security in numbers, that is, a reasonable number of developers have contributed in the development of the component and also it has been used by a reasonable number of developers. Stability is also related to

the usage history of the component. Usage history provides a hint about the usefulness of the component. Another side of usage history is the maturity of the component.

The subjects also have a consensus on the understandability attribute. It is also related to the maintainability of the component; a component that is easy to understand is easy to maintain. Understandability affects the reliability of a component.

Variability is one of the factors but on the other hand it decreases understandability. Variability is also seen as the configurability of a component, that it can be configured in multiple configurations.

The details of the exploratory study will be found in future publication [24] of the authors, it is work in progress.

## 4 PROPOSED REUSABILITY ATTRIBUTE MODEL

The proposed reusability assessment model (figure 2) contains six attributes related to the reusability of an SPL component: flexibility; maintainability; portability; scope coverage; understandability; variability. These emerged from the exploratory study. These attributes are selected due to their 'internal' nature as the following measurement is based on the code of the components.

In the IEEE standard for software quality methodology it is stated that software quality is measured by identifying a set of factors relevant to the software [11]. In our work the quality we concern ourselves with is 'reusability' and the factors affecting reusability are identified in the context of an SPL. Complex quality factors cannot be measured directly so factors may be split into sub factors. The factors/sub factors are measured by measures called 'metrics.'

The proposed model is derived using the GQM approach as shown below. The GQM model helps one to understand and define the factors to measure software quality. The 'object of study' defines the scope of measurement, which in this case is a 'class.' The 'purpose' of a measure is to predict the effort required to reuse the software. The 'viewpoint' considered is that of a software developer/user of the component. The 'environment' is one in which intense reuse is employed, as in the case of product line/family development.

Object of study: Class
Purpose: Prediction
Quality focus: Effort required to reuse
Viewpoint: Developer
Environment: Development of software in a reuse intensive environment (product line/ family)
**Goal:** Assessment of object oriented systems to predict reusability from the viewpoint of a developer.

1. How easy is it to reuse the component?
    - 1.1. How much variability is there in the component?
        - 1.1.1. What is the average number of methods per class?
            - 1.1.1.1. Number of methods ÷ Total number of classes
        - 1.1.2. What is the average number of children per class?
            - 1.1.2.1. Number of children ÷ Total number of classes
    - 1.2. How easy is it to understand the component?
        - 1.2.1. What is the size of the component?

1.2.1.1.Number of methods (NOM)
1.2.1.2.Lines of code (LOC)
1.2.2. How much coupling is there in the component?
1.2.2.1.Coupling between objects (CBO)
1.2.3. How much cohesion is there in the component?
1.2.3.1.Lack of cohesion in methods (LCOM)
1.2.4. How many comment lines are there in the component?
1.2.4.1.No. of comments
1.3. How easy is it to maintain the system?
1.3.1. Maintainability Index (MI)
1.3.2. McCabe's Cyclomatic Complexity (MCC)
1.4. How much flexibility is there in the component?
1.4.1. How much coupling is there in the component?
1.4.1.1.CBO
1.4.2. How much cohesion is there in the component?
1.4.2.1.LCOM
1.5. How portable is the component?
1.5.1. How independent is the component?
1.5.1.1.Depth of inheritance tree (DIT)
1.6. How much of the scope is covered by the component?
1.6.1. How many features are covered by the component?
1.6.1.1.NOM/Total number of methods in all classes

## 5 ATTRIBUTES AND METRICS

In this section a description of the attributes and metrics which are used to assess reusability is provided.

### 5.1 Maintainability

In [9] maintainability is defined as "the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment". Two metrics, MCC and MI, are used to measure maintainability.

### 5.2 Portability

It is defined as "the ease with which a system or component can be transferred from one hardware or software environment to another". The portability of a component depends on its independence, i.e. the ability of the component to perform its functionality without external support. In a scenario where an open source component is used in SPL development, the component should have the characteristic of portability. The component being a core asset may be used in the development of another product/family member within the product line/family.

### 5.3 Flexibility

It is defined as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed" [9]. In [25-27] flexibility is considered as a factor affecting the reusability of a component. In the context of an SPL, the flexibility characteristic is necessary for a core asset as it is intended to be reused in the development of other products.

### 5.4 Understandability

It is defined as "the ease with which a system can be comprehended at both the system-organizational and detailed statement levels"[9]. In [25, 28] understandability is considered a factor of reusability.

### 5.5 Scope coverage

It is the attribute that measures the number of features provided by the component against the total number of features in the SPL scope.

### 5.6 Independence

The term 'independence' is introduced to reflect the property of the system concerning the ability of a class to perform its responsibilities on its own. Independence is measured by DIT. The classes lower in the hierarchy are inherited by other classes; these classes depend on their ancestors to perform their functionalities.

### 5.7 Size Metrics

In [10] the aspect of the software dealing with its physical size is named the 'length' of the software. The metric used for size is lines of code (LOC). It counts the lines of source code. The second metric used to measure size is number of method (NOM).

### 5.8 Coupling and Cohesion Metrics

Coupling and cohesion are two key concepts in object oriented software engineering. Both of these are related to interaction between the entities. The higher the level of interaction, the higher is the level of dependency. The lower the level of interaction, the higher is the level of cohesion. Cohesion refers to the extent to which an entity can perform its responsibilities on its own. The metric used for coupling is CBO and the one used for cohesion is LCOM.

### 5.9 Variability Metrics

In [14] types of variability are defined on the basis of component reference models, namely CORBA and EJB. The building blocks of a component are defined as classes, workflow among classes, and interfaces.

We can consider the entities involved in object oriented programming. In Java these comprise the classes, interfaces, packages and Java beans. From the viewpoint of reuse, using Java beans is considered to be a black box approach. However, our work is concerned with a white box approach to the reuse of components.

An object oriented class consists of attributes, which hold data, and methods that exhibit behavior. An abstract class is used as a super-class for a class hierarchy, it cannot be instantiated.

In [14]variability types that are described are 'attribute', 'logic' and 'workflow.' Another view of variability types is presented in [15] where variability is categorized as positive, negative, optional, function and platform/environment. All of the variability types given in [14] can be mapped to the variability types given in [15], for instance, the 'attribute' variability type is a 'positive' variability type when a new attribute is added.

Attribute variability can be implemented using any of the following techniques: inheritance; aggregation; parameterization /generics; overloading. The cases of attribute variability are defined in [14]. One of these is the variation in the number of attributes. This type of variability is supported by inherence and aggregation. Another type of attribute variability is variation in the data types of the attributes; this variability is supported by parameterization/generics.

As described earlier, inheritance is one of the mechanisms to handle attribute

variability. In our work we propose variability metrics on the basis of the theory and mechanism of inherence.

With inherence the subclass inherits all the methods and attributes of the super-class. The subclass can define its own attributes in addition to those it inherits from the super-class, which causes the attribute variability. The other mechanism associated with inheritance is overloading which causes logic and work flow variability. So, a class that is higher in the hierarchy, and therefore having more accessible attributes and methods, has more variability.

A systematic review presents the state of the art in the area of software measurement [29]. The results of the review show that there is no measure available for variation. This shortage of metrics to measure variability, specifically at the implementation level, is also recognized in another study [30]. In our work we acknowledge this gap and propose metrics to assess the variability of software components.

Followings are the definitions of the metrics used in the proposed model.

### 5.10 CBO

These metrics count the number of classes to which a class is coupled [31]. Coupling prevents a class from performing its responsibility on its own, i.e. the class having a higher CBO value is more dependent on other classes. This dependence of a class on other classes decreases its understandability and flexibility. It is measured on an absolute scale; its domain is the set of integers [0, ∞).

### 5.11 LCOM

Cohesiveness is the property that enhances encapsulation. LCOM metrics indicate the lack of cohesion; lack of cohesion decreases understandability and flexibility[31]. It is measured on an absolute scale; its domain is the set of integers [0, ∞).

### 5.13 DIT

This is a measure that indicates the depth of a class within a hierarchy[31]. The class lower in the hierarchy depends on all the ancestor classes; it hinders its ability to be independent. A higher value of DIT reduces the independence which results in decreased portability. It is measured on an absolute scale; its domain is the set of integers [0, ∞).

### 5.14 LOC

This is a measure of the lines of source code. It is a size indicator of the entity. The size of the software affects its understandability. It is measured on an absolute scale; its domain is the set of integers [0, ∞).

### 5.15 NOM

This is used in [32]. It measures the number of methods declared within the class. It is an indicator of the size of a class. It is measured on an absolute scale; its domain is the set of integers [0, ∞).

### 5.16 NOC

NOC is the measure that counts the children of a class [31]. NOC itself shows the reuse of a class. A large number of children mean that the functionality of the class is reused through inheritance. It is measured on an absolute scale; its domain is the set of integers [0, ∞).

The equations used to calculate the attributes value are as following:

Flexibility =1 - [(0.5 X Coupling) + (0.5 X Cohesion)]

Coupling = adjusted CBO, Cohesion = adjusted LCOM

Understandability = 1 - [(0.25 X Coupling) + (0.25 X Cohesion) + (0.25 X Comments) + (0.25 X Size)]

Size = (0.5 X adjusted LOC) + (0.5 X adjusted NOM)

Portability = Independence = 1 - adjusted DIT

Scope coverage = NOM ÷ Total number of methods in all classes

Maintainability = (0.5 X adjusted MCC) + (0.5 X adjusted MI)

Variability = 0.5 X (NOC ÷ Total number of classes) + 0.5 X (NOM ÷ Total number of methods in all classes)

Reusability of Class = 0.16 X Flexibility + 0.16 X Understandability + 0.16 X Portability + 0.16 X Scope coverage + 0.16 X Maintainability + 0.16 X Variability

## 6 Validations

As with other engineering disciplines, software engineering is intended to help humans in solving their problems [33]. Software engineering, being a multidisciplinary field of research, involves issues raised by technology and society (humans). Software engineering activities depend on tools and processes. However, due to the involvement of humans, social and cognitive processes should also be considered [34]. Validation of new tools and processes is a necessary part of the advancement of software engineering [35]. The involvement of humans in software engineering demands the usage of research methodologies from the social sciences. Therefore, to validate the set of metrics selected to measure variability, a survey was used. A survey can be defined as a comprehensive system for collecting data using a standardized questionnaire [36-37]. The information collected from a survey is used to "describe, compare or explain knowledge, attitudes and behavior" [36]. This type of validation is used in [38], where the term 'experiment' is used for the process of assessment of software (classes) by experienced developers and students. In [28] a 'rating committee' is used. A questionnaire is used in [39] and [40] for the purpose of validation of results.

Survey research is common in the software engineering discipline. Due to the effectiveness of surveys in software engineering, researchers have laid down a process to conduct surveys. In [37] a comprehensive seven step process for conducting a survey is explained. We have used this approach presented and customized two steps. The details and the rationale for our decision are stated later in this section. The specific steps taken to conduct this variability assessment survey were:

- Identification of aim
- Identification of target audience
- Design of sampling plan
- Questionnaire formulation
- Pilot test of questionnaire
- Questionnaire distribution
- Analysis of the results

Let us clarify the purpose of this exercise. Our notion of a survey resembles the process used in [38] where, as we stated above, the term 'experiment' is used to conduct the assessment of software code by humans.

In this paper we have used the term 'survey' because we are using the questionnaire as a tool to assess the code.

The aim of this survey is to get an objective assessment, from humans, of selected software code. Turning to the second step, 54 students of a software engineering class were asked to assess the variability of classes in the class hierarchies. Out of 54, five samples were discarded due to lack of information. The selected students had knowledge and experience in Java programming, software engineering, and the concept of object-orientation. They were studying these subjects as part of a computer and information sciences degree program. A total of 15 classes were selected in three hierarchies related to three different components. Three components were selected, namely Component A from a rental domain, Component B from a computer user account domain and Component C from a bank account domain. More details of the components are provided in table-3. The components selected for this purpose were from Merobase (http://www.merobase.com). Merobase is database of source code files. The collection has more than 10 million indexed files, out of which eight million are Java files. A search and tagging engine is included.

Table 3. Component specifications

| Component | No. of classes | No. of methods | LOC |
|---|---|---|---|
| A | 03 | 31 | 204 |
| B | 03 | 11 | 78 |
| C | 09 | 34 | 281 |

A sampling plan was designed to decide the kind of statistical test used to interpret the results. The questionnaire was formulated and reviewed by the authors. The questionnaire was pilot tested and revised. The survey was conducted in two sessions, 18 respondents completed the questionnaire in the first session and 36 in the second session. Both sessions were conducted in the presence of the authors. The results of the survey were analysed using statistical software.

The response of the users was collected using a Likert scale from 1 to 5 - strongly disagree (1); disagree (2), neither agree nor disagree (3); agree (4); strongly agree (5). Next we made use of Cronbach's alpha (α). (This is commonly used in software engineering measurements to assess internal consistency.) The internal consistency and reliability of the data, measured in terms of α, is presented in table-4. The α coefficient of the responses for each component is computed. All the values of α are greater than .7, which is considered sufficiently good.

Table 4: Cronbach's α of component values

| Component | No. of evaluators | Cronbach's $\alpha$ |
|---|---|---|
| A | 49 | .771 |
| B | 49 | .848 |
| C | 49 | .832 |

The evaluators were asked eighteen questions to assess the variability of components. The arithmetic means of the responses is presented in table-5. The value of reusability is the mean value of the individual responses to the questions. The graph of these values is plotted in figures 3, 4, 5.

Table 5: Results of reusability assessment by human evaluators and using proposed mode

| Component | Evaluators Assessment (Mean) | Assessment using proposed metrics | |
|---|---|---|---|
| A | 3.31 | Class 1 | 2.89 |
| | | Class 2 | 2.88 |
| | | Class 3 | 2.68 |
| B | 3.15 | Class 1 | 3.03 |
| | | Class 2 | 3.22 |
| | | Class 3 | 3.31 |
| C | 3.22 | Class 1 | 3.09 |
| | | Class 2 | 3.24 |
| | | Class 3 | 2.98 |
| | | Class 4 | 3.07 |
| | | Class 5 | 3.08 |
| | | Class 6 | 3.1 |
| | | Class 7 | 2.93 |
| | | Class 8 | 3.03 |
| | | Class 9 | 3.26 |

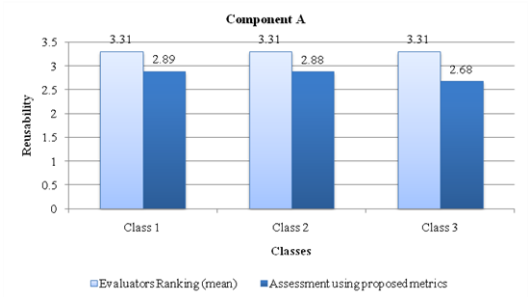


Figure 3: Classwise reusability values of component A

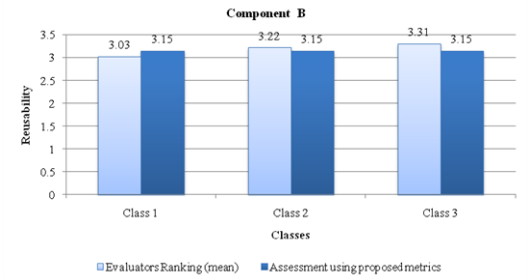


Figure 4: Class wise reusability values of component B

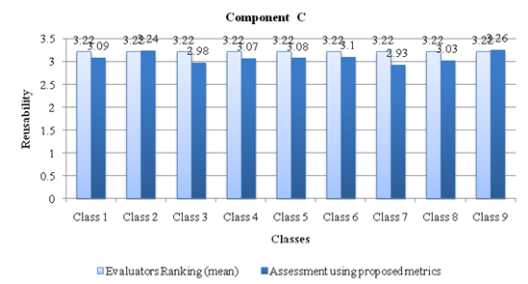


Figure 5: Class wise reusability values of component C

## 7 DISCUSSIONS

Our work involves identifying reusability assessment metrics. Some of these are known, whereas others have been introduced by us. In some other research metrics are presented but not validated e.g. [41]. In our work however, we both present the metrics and validate them empirically.

[42] and [43] assess reusability based on the degrees of coupling and cohesion. In comparison, our work considers these as well as other factors. Our work focuses on components written in java. The metrics that we have selected are to a certain extent dependent on java.

The list of factors affecting reusability was arrived at following interviews with experts. Next the metrics applicable to these was decided upon. Most of these came from literature review, however, a small number were devised by ourselves, details of which are the subject of a forthcoming paper. Finally, we took a number of classes and assessed their reusability by two means. One assessment was carried out using the metrics; the other assessment was done manually by final year computing student. The results were compared.

## 8 CONCLUSIONS

A reusability attribute model and metrics are presented in this paper as the result of an exploratory study. We have

highlighted gaps in the current literature: the variability and scope coverage attributes of a software component are not being catered for. In our opinion the current approaches are not suited to the context of OSS and SPLs. Therefore, the proposed addition of attributes will provide more reliable assessment of reusability. The results of the assessment survey are presented. The values of reusability and its attributes are compared with the ratings obtained by the survey. The other facet of this research work is the bridging of the gap between OSS and SPLs. The research is aligned with the work that is progressing in software engineering. Our work contributes to the knowledge base. Our future work is planned to include the automation and further validation of our approach.

## 9 REFERENCES

1. Krueger, C. W.: Software reuse, *ACM Comput. Surv., 24*, 131-83 (1992).
2. Mohagheghi, P., and Conradi, R.: Quality, productivity and economic benefits of software reuse: a review of industrial studies, *Empirical Softw. Engg., 12*, 471-516 (2007).
3. Clements, P., and Northrop, L.: *Software product lines: practices and patterns*, Addison-Wesley Longman Publishing Co., Inc. (2001).
4. Hummel, O., Janjic, W., and Atkinson, C.: Code Conjurer: Pulling Reusable Software out of Thin Air, *Software, IEEE, 25*, 45-52 (2008).
5. Ahmed, F., Capretz, L. F., and Babar, M. A.: A Model of Open Source Software-Based Product Line Development, In: *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pp. 1215--20 (2008).
6. Ahmed, F., Capretz, L. F., and Capretz, M. M. A.: Setting Up COTS-Based Software Product Lines, In: *Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS '07*, pp. 249 - (2007).
7. Capretz, L. F., Ahmed, F., Al-Maati, S., and Aghbari, Z. A.: COTS-based software product line development, *International Journal of Web Information Systems, 4*, 165 - 80 (2008).
8. Frakes, W. B., and Kyo, K.: Software reuse research: status and future, *IEEE Transactions on Software Engineering, 31*, 529-36 (2005).
9. IEEE: Systems and software engineering -- Vocabulary, In: *ISO/IEC/IEEE 24765:2010(E)*, pp. 1-418 (2010).
10. Fenton, N., and Pfleeger, S.: *Software Metrics: A Rigorous and Practical Approach*, PWS Publishing Co. (1997).
11. IEEE: IEEE Standard for a Software Quality Metrics Methodology (1998).
12. Frakes, W., and Terry, C.: Software reuse: metrics and models, *ACM Comput. Surv., 28*, 415-35 (1996).
13. Fazal-e-Amin, Mahmood, A. K., and Oxley, A.: A Review of

Software Component Reusability Assessment Approaches, *Research Journal of Information Technology, 3*, 1-10 (2011).

14. Kim, S. D., Her, J. S., and Chang, S. H.: A theoretical foundation of variability in component-based development, *Information and Software Technology, 47*, 663-73 (2005).
15. Sharp, D. C.: Containing and facilitating change via object oriented tailoring techniques, In: *First Software Product Line Conference*, Denver, Colorado (2000).
16. Gacek, C., and Anastasopoules, M.: Implementing product line variabilities, *SIGSOFT Softw. Eng. Notes, 26*, 109-17 (2001).
17. Pohl, C., Rummler, A., Gasiunas, V., Loughran, N., Arboleda, H., Fernandes, F. d. A., Noyé, J., Núñez, A., Passama, R., Royer, J.-C., and Südholt, M.: Survey of existing implementation techniques with respect to their support for the practices currently in use at industrial partners, In: *AMPLE Project deliverableD3.1* (2007).
18. Fazal-e-Amin, Mahmood, A. K., and Oxley, A.: An analysis of object oriented variability implementation mechanisms, *SIGSOFT Softw. Eng. Notes, 36*, 1-4 (2011).
19. Fazal-e-Amin, Mahmood, A. K., and Oxley, A.: Mechanisms for managing variability when implementing object oriented components, In: *National Information Technology Symposium (NITS)*, King Saud University, KSA (2011).
20. Jilles, v. G.: Variability in Software Systems, the key to Software Reuse, Licentiate Thesis, University of Groningen, Sweden, (2000)
21. Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley (1998).
22. Gray, D. E.: *Doing Research in the Real World*, SAGE Publication Ltd. (2009).
23. Strauss, A., and Corbin, J.: *Basics of Qualitative Research Techniques and Procedures for Developing Grounded Theory*, Sage Publications (1998).
24. Fazal-e-Amin, Mahmood, A. K., and Oxley, A.: Using Open Source Components in Software Product Lines – an exploratory study, . In: *IEEE Conference on Open Systems 2011*, Langkawi, Malaysia (2011).
25. C. Sant'anna, A. G., C. Chavez, C. Lucena, and A. v. von Staa: On the reuse and maintenance of aspect-oriented software: An assessment framework, In: *Proceedings XVII Brazilian Symposium on Software Engineering* (2003).
26. Pohl, K., Böckle, G., and Linden, F. v. d.: *Software Product Line Engineering Foundations, Principles, and Techniques*, Springer-Verlag Berlin Heidelberg (2005).
27. Sharma, A., Grover, P. S., and Kumar, R.: Reusability assessment for software components, *SIGSOFT Softw. Eng. Notes, 34*, 1-6 (2009).
28. Washizaki, H., Yamamoto, H., and Fukazawa, Y.: A Metrics

Suite for Measuring Reusability of Software Components, In: *Proceedings of the 9th International Symposium on Software Metrics*, IEEE Computer Society, pp. 221-5 (2003).

29. Gómez, O., Filipe, J., Shishkov, B., Helfert, M., Oktaba, H., Piattini, M., and García, F.: A Systematic Review Measurement in Software Engineering: State-of-the-Art in Measures, in *Software and Data Technologies*, Vol. 10, Springer Berlin Heidelberg, pp. 165-76 (2008).
30. Mujtaba, S., Petersen, K., Feldt, R., and Mattsson, M.: Software Product Line Variability: A Systematic Mapping Study, In: *15th Asia-Pacific Software Engineering Conference APSEC 08* (2008).
31. Chidamber, S. R., and Kemerer, C. F.: A metrics suite for object oriented design, *Software Engineering, IEEE Transactions on, 20*, 476-93 (1994).
32. Li, W., and Henry, S.: Maintenance metrics for the object oriented paradigm, In: *Software Metrics Symposium, 1993. Proceedings., First International*, pp. 52-60 (1993).
33. Jackson, M.: The Name and Nature of Software Engineering, in *Advances in Software Engineering: Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*, Springer-Verlag, pp. 1-38 (2008).
34. Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D.: Selecting Empirical Methods for Software Engineering Research, in *Guide to Advanced Empirical Software Engineering*, pp. 285-311 (2008).
35. Deelstra, S., Sinnema, M., and Bosch, J.: Variability assessment in software product families, *Information and Software Technology, 51*, 195-218 (2009).
36. Pfleeger, S. L., and Kitchenham, B. A.: Principles of survey research: part 1: turning lemons into lemonade, *SIGSOFT Softw. Eng. Notes, 26*, 16-8 (2001).
37. Kasunic, M.: *Designing an Effective Survey*, Vol. CMU/SEI-2005-HB-004 SEI, CMU (2005).
38. Etzkorn, L. H., Hughes, W. E., and Davis, C. G.: Automated reusability quality analysis of OO legacy software, *Information and Software Technology, 43*, 295-308 (2001).
39. Dandashi, F.: A method for assessing the reusability of object-oriented code using a validated set of automated measurements, In: *Proceedings of the 2002 ACM symposium on Applied computing*, ACM, Madrid, Spain, pp. 997-1003 (2002).
40. Münch, J., Abrahamsson, P., Washizaki, H., Namiki, R., Fukuoka, T., Harada, Y., and Watanabe, H.: A Framework for Measuring and Evaluating Program Source Code Quality, in *Product-Focused Software Process Improvement*, Vol. 4589, Springer Berlin / Heidelberg, pp. 284-99 (2007).
41. Cho, E. S., Kim, M. S., and Kim, S. D.: Component Metrics to Measure Component Quality, In:

*Proceedings of the Eighth Asia-Pacific on Software Engineering Conference*, IEEE Computer Society, pp. 419-26 (2001).

42. Gui, G., and Scott, P. D.: Ranking reusability of software components using coupling metrics, *J. Syst. Softw.*, *80*, 1450-9 (2007).
43. Gui, G., and Scott, P. D.: Measuring Software Component Reusability by Coupling and Cohesion Metrics, *Journal of Computers*, *4*, 797-805 (2009).