Component-Based Explicit Software Reuse

O. K. Harsh+ and A. S. M. Sajeev*

Abstract— **We suggest an analytical method to calculate explicit component based software reuse. Our method is based on three dimensional Vector Measure of software size which we have derived in terms of the functionality and problem complexity by extending the work of Hastings and Sajeev. Our calculated analytical results are in close agreement with expected results.**

We try here to confirm reuse by predicting reuse in terms of reuse benefit by satisfying all the properties defined by Devanbu et al. Present method is available early in the life cycle of a project.

Present technique is useful in understanding components composition environment including the software development life cycle and the deployability. Present reuse calculation also overcomes many difficulties faced by software developers.

Index Terms— **Reuse, Vector Reuse Measure, Reuse Benefit, Three Dimensional Model**

1. Introduction

Problems of reuse in large systems (James (1996), Objected Oriented Systems (Hitz (1995)), complex systems etc. has been studied by many workers. However, none of these workers comprehensively outlined theoretical or experimental models to calculate, observe or validate the reuse measure across the entire life cycle or during the early phase of the life cycle.

Measurement becomes difficult at the higher reuse rate especially in the large and the Object Oriented Systems. So far reusability has not been treated as an independent measurable quantity which can resolve the problem of component relationship in a given composition environment. We know that most of the existing reuse metrics can measure the code. However measuring code cannot predict the amount of reuse at the early stage of system life cycle. Measuring components interaction is a great problem at system, project or domain level and it becomes quite tedious task to measure the system's performance. Question here is that, can we propose a model which could account all possible coupling of components at all levels of system developments where the reusability will have a direct bearing on the systems performance? Yes, we are trying to dig the answer in the present work which is based on the extended work of Hastings and Sajeev (2001) and which has already successfully provided calculation of efforts in several projects. We will also involve the extended reuse benefit properties of Devanbu et al. (1996) (from two to three dimensional) to further support our argument.

2. Related Research & Problems

Devanbu et al. (1996) evaluated analytically and empirically "how well several published software reuse metrics measure the "time, money and quality", benefits of software reuse." Frakes and Terry (1996) surveyed the metrics and models of software reuse and reusability and reviewed the six types of metrics and models namely cost benefit models, maturity assessment models, amount of reuse metrics, failure modes models, reusability assessment models and reuse library metrics. However, most of these models fail to address the issues of explicit relationships of reusability with the domain properties, functions and the integration relation problems. These models could not differentiate between the domain reuse and project reuse also.

Above discussions suggest that we do not have a mathematical formulation or logic which helps us in uniformly selecting, deploying and integrating the components for the reuse. We also do not have a technique to classify and manage the components in the different type of systems.

Review of past theories and discussions suggest that reusability should be treated explicitly which should have a direct relation with the entire efforts at all stages in building a software so that we can deal problem of reusability at all stages. Our knowledge about components and its availability in day to day tasks has become so wide that every day we find ourselves in a new environment. Practically speaking reuse is now not domain specific or project specific rather it has become time or temporal specific. Reuse has now become a relative term as compared to what we have and what we can do? Day to day enhancement of knowledge and applications has certainly created a situation where we have to say that "reuse is a time oriented concept".

In this work we suggest a technique for the measurement of reuse in components based systems. Present measure is based on three dimensional Vector Measure of software size which

we have derived in terms of the reusability, functionality and problem complexity by extending the work of Hastings and Sajeev (2001) which is based on Operators and Operands. Present method which is based on the Vector Reuse Measure (VRM) specifications is not only available early in the life cycle of a project to overcome difficulties met by inexperienced software developers, but also helps us in selecting components for the reuse by Customers based-on the properties mentioned herewith.

3. Vector Size Measure and Reusability

The Vector Size Measure (VSM) (Hastings and Sajeev 2001) can measure software size from requirements specifications. To estimate early in the lifecycle, we need the measurement of size based on the specification. This helps us to estimate effort early during the cycle. VSM is a software size vector measured in terms of functionality and problem complexity. VSM is based on specifications which are formally expressed as Abstract Data Types (ADT).

On the basis of two fundamental software size attributes, i.e. functionality and problem complexity, Hastings and Sajeev (2001) represented software size as the two-dimensional vector which has both magnitude and direction. Given that we have three fundamental software size attributes in the new model, i.e., functionality, problem complexity and reusability, we can represent software size as a three dimensional vector which has both magnitude and direction (Figure 1). This representation allows us to understand and transform software size measurements using well-defined mathematical functions.

Consider a three dimensional picture (Figure 1) where the ADT's are defined along the axis of X, Y and Z as functionality, problem complexity and reusability respectively. Three dimensional model is the extension to the Fenton's (1991) suggestion that software size, S, is a function, f_S, of length, functionality, and problem complexity, such that: S :: = f_S (l, f, c_o) where l represents the total number of entities in a system, f represents the number of functions a system provides, and c_o represents the underlying problem complexity.

According to the work of Hastings and Sajeev (2001), length is a derived attribute, thus they considered S :: = f_S (f, c_o). We, in the present work, extend this as follows:

S :: = f_S (f, c_o , r_{eS}) where r_{eS} is the reusability.

As mentioned in the Figure 1, (By using the plane vector algebra (Ayres 1972)) magnitude, m, may be defined in terms of the reusability as: m = Real Part +Imaginary Part = **i**f + **j**c - ¡**k** r_e, where **i, j** and **k** are the unit vectors along the x, y and z axis respectively and ¡ = √- 1 and

m = √ ($f^2 + c^2 - r_{eS}^2$) [OP] (1)

Direction is defined as theta = tan-1 (c / √(($f^2 - r_{eS}^2$)). However, we are more interested in the ratio between problem complexity and so called effective functionality, i.e., the gradient, g, to indicate the relative dimensions of a software system, where:

g = sqrt (c^2 – r_{eS}^2)/f (2)

Gradient is a ratio of problem complexity and effective functionality (in terms of the reusability) which tells us about the relative dimensions of systems, i.e., the characteristics of software systems.

Vector Representation

As discussed above the size has been extended in the present work to include the percentage value of reusability (r_{eS}) as: m = $\sqrt{(f^2 + c_0^2 - r_{eS}^2)}$ (3) where f, c_o and r_{eS} are the vector quantities, and the direction as [tangent of] theta = g = sqrt (c^2 - r_{eS}^2)/f . This represents the relative dimensions of a software system (note that f > 0) which suggest that presence of reusability reduces the effective value of the gradient. If the reusability is zero then equation (3) reduces into the Hastings and Sajeev (2001) formula. The square of the reusability in this formula will not be affected by positive or negative sign of the reusability.

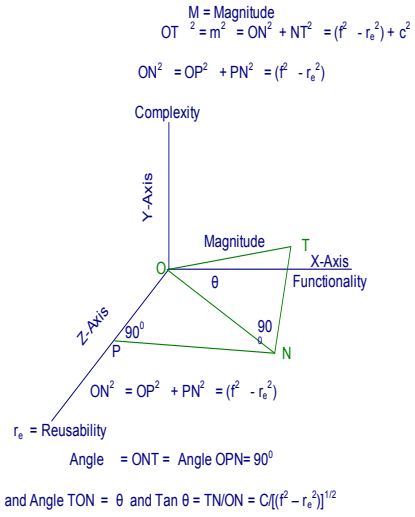

Figure 1: Software Size as a vector in terms of three dimensional Representation .

Let Z_k be the size of Component k where Z_k is a vector quantity. Note that Z_k is greater than 0 except for a null specification. Given a system, S consisting of q components we can measure its size as:

$$Z_S = \sum_{k=1}^{q} Z_k \qquad (4)$$

We now define Expanded Size of a system S as:

$$X_S = \sum_{k=1}^{q} n_k Z_k \qquad (5)$$

Where n_k is the number of times Component k's public functions (and attributes, if any) are used in other components. This definition of expanded size is similar to that of Devanbu and Karstu (1994). The only difference is that present method measure the expanded size from specifications, where as Devanbu and Karstu's measured it from code. We can also define that Cost is proportional to Size, that is:

$$C \propto Z_S \qquad (6)$$

Using equation (6), one can predict the variation in cost with the size. Present reuse measure is based on Vector Reuse Measure (VRM) specifications, which is available early in the life cycle of a Project.

4. A Measure for Reuse

Here we define a reusability measure called Vector Reuse Measure (VRM). Reuse is possible in two ways: One is the number of times a component is used by other components, and the other is the amount of change that is made to a component before it is reused.

It should be noted that a component which is set to be developed as part of a system 'S' is considered internal to 'S' while a component that is already available, but is set to be used in System S is considered an external component and its reusability will be different than the reusability of an internal component. Since in the present work, size is directly related with the functionality, complexity and reusability on equal footing, therefore, it is easy to realize reuse equally in terms of functionality and complexity.

5. Reusability and Adaptability of components

Two components k and k' are close provided their specifications are modified in such a way that:

$$\wedge = \frac{Z_k}{Z_k + \delta Z_k} = \wedge_k + \delta Z_k \qquad (7)$$

Where δZ_k is a very small quantity which is a difference of the specifications of Components k and k'. Similarly we can also define the reusabilities of two components k and k' as:

$$r_{eS1} = r_{eS2} + \delta r_k \qquad (8)$$

Where δr_k is the difference of reusabilitites of Components k and k' and it may be expressed as $\delta r_k = k - k'$ (9)
Having defined reuse of a component, we can measure reusability of a system S as:

$$r_{eS} = \sum_{k=1}^{p} \wedge_k n_k Z_k \qquad (10)$$

Where Z_k is the size of the components and n_k is the number of times component k's public functions (and attributes if any) are used in other components.

6. Vector Size Measure in Three Dimensions and Reuse Benefit Estimation

Magnitude m may be expressed in three dimensions as

$$m = \sqrt{(f^2 + c_0^2 - r_{eS}2)} \qquad (11) \quad \text{and}$$

$$g = \text{sqrt}\,(c\text{^}2 - r_{eS}\text{^}2)/f \qquad (12a)$$

Where r_{eS} is the reusability, g is the gradient, f is the functionality and c_o is the problem complexity of the system.

If (r_{eS}/f) is small then $g = (c_0/f)$ (12b)
which correspond to the two dimensional result. Therefore, using equations (3) and (10), we get

$$m = \sqrt{(f^2 + c_o^2 - r_{eS}^2)} =$$

$$\sqrt{(f^2 + c_o^2 - (\sum_{k=1}^{p} \wedge_k n_k Z_k)^2} \qquad (13)$$

If we know the f, c_o and m then we can calculate the reusability (r_{eS}) from the equation (13) or if we know the f, c_o then we can calculate the different values of m as the reusability (r_{eS}) varies. In the present work we take the Reuse Benefit (Devanbu et al. (1996)) as:

Reusable Benefit =

$$R_b(S) = \frac{C(S\ without\ reuse) - C(S\ with\ reuse)}{C(S\ without\ reuse)} \qquad (14)$$

Where as mentioned in the equation (6), C is proportional to size.

Frakes and Terry (1996) proposed cost and productivity model for software reuse. According to them the relative cost of software development (C) may be expressed as:

$$C = [\,(b'-1)\,R\,] + 1 \qquad (15) \quad \text{or}$$

(C-1) = [(b'-1) R] (16)

Where R is the type of reuse level (company's overall reuse rate) and b' is the cost relative to that for all new code, of incorporating the reuse code into the new product. (b' =1 for all new code). Using equation (15), we can also calculate the reuse benefit by using the equation (6). Thus equation (15) and equation (14) may be expressed as:

$$C' = \frac{C(S\ without\ reuse) - C(S\ with\ reuse)}{C(S\ without\ reuse)} = R_b(S)$$

$$= [(b'-1)R] \quad (17)$$

Hastings and Sajeev (2001) have derived an expression for the estimated effort by considering a two dimensional model of problem complexity and functionality as:

$$E = a\, m^b\, g^z \quad (18)$$

Where m is the measured magnitude of a software specification and g is the measured gradient; a, b and z are coefficients. In the present work we have derived a three dimensional formula for the magnitude m (equation 11) and gradient (equation 12b). Equation (18) can furnish the estimated effort for software using the three dimensional model involving reusability, problem complexity and functionality if we replace m and g in equation (18) by equations (11) and (12b) as:

$$E_n = a\, (\sqrt{(f^2 + c_o^2 - r_{eS}^2)})^b\, g_1^z = a\, m_1\, g_1^z \quad (19)$$

Clearly the difference of equations (18) and (19), gives us the reusable size as:

$$E_{Reus} = C(S \text{ with reuse}) = [a\, m^b\, g^z - a\, m_1\, g_1^z] \quad (20)$$

The ratio of equations (20) and (18) will give us the reuse benefit as:

$$C' = E_{Reus} / E \quad (21)$$

Where a = 0.7113 , z = 1.1585 and b = 1.0244 (Hastings & Sajeev 2001) are constants, $m_1 = (\sqrt{(f^2 + c_o^2 - r_{eS}^2)})$ is the reusable magnitude and g = sqrt (c^2 - r_{eS}^2)/f while g and m are respectively the original gradient and the original magnitude of the system respectively. Equation (21) can calculate the reuse benefit provided we know f and c_o for a given $r_{es.}$ This can be compared with equation (17) since we define that Cost as proportional to Size (equation 6), that is:

$C \propto Z_S$.

We have predicted theoretically reuse benefit values in two projects discussed by Hastings and Sajeev (2001) by using equation (21) (by substituting f, c and assumed reusabilities); this is then compared to that of Frakes and Terry (1996) formula for same reusability values (using equation 17) (see Tables I and II). A close agreement between them has been obtained. In our present formulation we can prove all the eight properties of reuse benefit as defined by Devanbu et al. (1996).

7. Reuse Benefit Properties for three dimensional model by extending the work of Devanbu et al. (1996):

Property 1: $\forall S,\ 0 \le R_b(S) < 1$, that is reuse benefit always lies between zero and one.

Using equations (13) and (14), we can write

$$R_b(S) = (\sqrt{(f^2 + c_o^2)} - \sqrt{(f^2 + c_o^2 - r_{eS}^2)})/\sqrt{(f^2 + c_o^2)}$$

or $R_b(S) = [(1 - \{\sqrt{(f^2 + c_o^2 - r_{eS}^2)})/\sqrt{(f^2 + c_o^2)}\}]$ the second term of this equation is always less than 1 since ∀S, $0 \le r_{eS} < 1$ where r_{eS} is given by equation (10)

Therefore, $0 \le R_b(S) < 1$ (22)

Therefore, this proves property 1. From equations (5) and (13) we can find out the reuse benefit for the expanded size. Clearly the numerator of equation (13) is always less than one or greater (or equal) to zero depending upon the type of components being reused and therefore reuse benefits satisfies the property 1.

Property 2:

$$\exists S_1, S_2 \mid Function(S_1) = Function(S_2) \ \ but\ \ R_b(S_1) \neq R_b(S_2)$$

That is, it is possible to have two systems with the same functionality but different reuse benefits.

Let $R_b(S_1)$ be the Reuse Benefit of a System S_1 consisting of two components of which the second component is reused three times while $R_b(S_2)$ is the Reuse Benefit consisting of same two components but replacing one use of the second component by a component which has the same functional and behaviour specification. In this case clearly reusabilites (r_{eS1} and r_{eS2}) are different, and, therefore, $r_{eS1} \neq r_{eS2}$, while complexities and functionality in both the systems are same.

Using equations (13) and (14), we can express for $R_b(S_1)$

$R_b(S_1) = [(\sqrt{(f^2 + c_o^2)} - \sqrt{(f^2 + c_o^2 - r_{eS1}^2)})/ \sqrt{(f^2 + c_o^2)}]$

or $R_b(S_1) = [(1 - \{\sqrt{(f^2 + c_o^2 - r_{eS1}^2)})/ \sqrt{(f^2 + c_o^2)}\}]$ (23)

Similarly for $R_b(S_2)$ (where the second component is reused three times) we have

$R_b(S_2) = [(1 - \{\sqrt{(f^2 + c_o^2 - r_{eS2}^2)})/ \sqrt{(f^2 + c_o^2)}\}]$ (24)

Since $r_{eS1}^2 \# r_{eS2}^2$ and therefore equation (23) is not equal to equation (24) which proves property 2.

Property 3: $\forall S \mid R_b(S_1) > 0, \exists S_1$ Function(S_1) = Function(S_2) and $R_b(S_1) > R_b(S_2)$

That is, for any system S, it is possible to develop another system, S_1 with the same functionality such that the reuse benefit of S_1 is less than that of S.

Let $R_b(S_1)$ be the reuse benefit of System *S*. $R_b(S_1) > 0$ means there is at least one component, *k* which is reused. Replace one use of a particular component with a new component with the same functional behaviour to create S_2 (It is an external component used once. similar to property (2)), otherwise, we can also replace one use of k with a new component with the same functionality and behaviour specifications (like property 2) to create S_2, If we look at the numerators of $R_b(S_1)$ (Equation 23) and $R_b(S_2)$ (Equation 24), we can see that the numerator of $R_b(S_2)$ is less as compared to the numerator of $R_b(S_1)$ because r_{eS2} is less than r_{eS1}, which makes the numerator of second term of equation (24) greater and hence the total value of $R_b(S_2)$ becomes less (since as discussed above we have replaced one used component with a new component to create S_2, therefore, r_{eS1} is greater than r_{eS2} and therefore equation (23) is greater than equation (24) or $R_b(S_1) > R_b(S_2)$ which proves property 3.

This property helps us to maximize reusability without changing the functionality.

Property 4: $\forall S, c \mid R_b(S_c^n) > R_b(S_c^{n-1})$

That is, for any system S that reuses a component c, the reuse benefit of using the component n times is more than that of reusing it n-1 times. Using equation (14) and (13) (since we define that Cost is proportional to Size (equation 6)), we can express for reuse benefit for using n times component is

$R_b(S_c^n) = [\sqrt{(f^2 + c_o^2)} - \sqrt{(f^2 + c_o^2 - (r_{e(S-c)} + n_c \wedge_c Z_c)^2}] / [\sqrt{(f^2 + c_o^2)}]$

or $R_b(S_c^n) = [1 - \{\sqrt{(f^2 + c_o^2 - (r_{e(S-c)} + n_c \wedge_c Z_c)^2} / (\sqrt{(f^2 + c_o^2)}\}]$ (25)

Similarly for $R_b(S_c^{n-1})$

$R_b(S_c^{n-1}) = [1 - \{\sqrt{(f^2 + c_o^2 - (r_{e(S-c)} + (n_c - 1) \wedge_c Z_c)^2} / (\sqrt{(f^2 + c_o^2)}\}]$ (26)

Where in equation (26) the number of times Component c has been reduced by 1. We can note from equations (25) and (26) that:

$(r_{S-c} + n_c \wedge_c Z_c) > (r_{S-c} + (n_c - 1) \wedge_c Z_c)$ which makes the numerator of equation (25) greater that the numerator of equation (26), and therefore,

$R_b(S_c^n) > R_b(S_c^{n-1})$ Which proves the property 4.

Using equations (5), (7) and (11) it also suggests us that how many components are there in the expended size?

Property 5:

if Cost(C) > Cost(c) then $R_b(S_c^-) > R_b(S_C^-)$

Where $R_b(S_c^-)$ is the reuse benefit of a system which is a perturbation of S by removing one use of component c. Similarly, $R_b(S_C^-)$ for Component C. According to this property if c is a less expensive component compared to C then the reuse benefit after removing one use of c will be greater than the reuse benefit after removing one use of C or we can say that it is beneficial to reuse higher cost components than lower cost components.

we can demonstrate property 5 mathematically by using equations (13) and (14) (since we define that Cost is proportional to Size (equation 6)). As we can note that property 5 is about reusing the two components, c and C, of different costs, therefore, it is suitable to assume that they are used verbatim, which means that $\wedge_c = \wedge_C = 1$. Thus we can write

(using equations (13) and (14) as:

$R_b(S_c) = [\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2 - (r_{e(S-c)} + n_c Z_c)^2}\}] / [\sqrt{(f^2 + c_o^2)}]$

$R_b(S_c^-) = [\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-C)} + (n_c - 1) Z_c)^2)\}] / [\sqrt{(f^2 + c_o^2)}]$ (27)

Similarly for $R_b(S_C)$

$R_b(S_C) = [\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-C)} + n_C Z_C)^2\}] / [\sqrt{(f^2 + c_o^2)}]$

$R_b(S_C^-) = [\sqrt{(f^2 + c_o^2)} - \sqrt{(f^2 + c_o^2} - ((r_{e(S-C)} + (n_C - 1) Z_C)^2] / [\sqrt{(f^2 + c_o^2)}]$ (28)

Since we know that Cost is the function of size, so Cost(C) is proportional to Z_C, and similarly Cost(c) is proportional to Z_c. Since C is a high cost component compared to c therefore it is straight forward to say that $Z_C > Z_c$.

Let us assume that $R_{S\text{-}c\text{-}C}$ is the reusability of System S excluding the reusability of Component c and C. Thus we can express the numerator of Equation (27) as:

$[\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-C)} + (n_c - 1)Z_c)^2)\}]$

$= [\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c-C)} + (n_c - 1)Z_c + n_C Z_C)^2)\}]$

$= [\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c-C)} + (n_c - 1)Z_c + (n_C - 1) Z_C + Z_C)^2)\}]$ (29)

Similarly we can express the numerator of Equation (28) as:

$[\sqrt{(f^2 + c_o^2)} - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c-C)} + (n_C - 1)Z_C + (n_c - 1) Z_c + Z_c)^2)\}]$ (30)

Comparing Equation (27) and (28),

$R_b(S_c^-) > R_b(S_C^-)$ if $Z_C > Z_c$ which demonstrated the property 5.

Property 6: $R_b(S_{c^e}) > R_b(S_c)$

where S_{c^e} is a system with an external component c^e and S_c is a system with c^e replaced by an internal component c of equivalent functionality. The property states that the reuse benefit out of using an external component is higher than that of using an internal component of same functionality. Internal component is said to be reused after using it once. It means that first time use of internal component is not counted towards reuse.

Let us assume that S_{c^e} be the System S which reuses an external (pre-existing) component c^e , therefore, its reusability is $r_{e(S-c}e_{)}$. Using equation (13) and (14), we can express its reuse benefit (external component c^e) as :

$R_b(S_{c^e}) = [1 - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c}e_{)} + n_c^e \wedge_{c^e} Z_c e)^2 / (\sqrt{(f^2 + c_o^2)}\}]$ (31)

Where $(r_{e(S-c}e_{)}$ is the reusability of the system with the external component and f is the functionality.

If we assume that S_c is the system with the same functionality as S in which uses of all external component c^e is replaced by an internal component c. Using Definition 1, we can say that number of reuses of internal component c is one less as compared to the number of reuses of the external component c^e . Therefore, we can express the reuse benefit of S_c as:

$R_b(S_c) = [1 - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c)} + (n_c^e - 1) \wedge_c Z_c)^2 / (\sqrt{(f^2 + c_o^2)}\}]$ (32)

Note that rest of the reuse is not affected by this perturbation, $(r_{e(S-c}e_{)}$ and $r_{e(S-c)}$ should be the same and as we know that c^e and c has the same specification therefore, $\wedge_{c^e} Z_c e$ is the same as $\wedge_c Z_c$. So we can express Equation (32) as:

$R_b(S_c) = [1 - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c}e_{)} + (n_c^e - 1) \wedge_c e Z_c e)^2 / (\sqrt{(f^2 + c_o^2)}\}]$ (33)

Thus comparing equations (31) and (32), we can express as:

$R_b(S_{c^e}) > R_b(S_c)$, which proves the property 6.

Property 7: $R_b(S_{c^{e,n}}) \geq R_b(S_{c^{e,n-1}}, c'^e)$

where $S_{c^{e,n}}$ is a system *S* where an external component c^e is used *n* times. c'^e is an external component identical in functionality to c^e. According to this property it is not more beneficial to use an external component c^e (n-1) times plus another equivalent external component c'^e once as compared to use it c^e n-times. Using equations (13) and (14) (since we define that Cost is proportional to Size (equation 6)) we can $S_{c^{e,n}}$ as

$R_b(S_c^n) = [1 - \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c} e) + n \wedge_{c^e} Z_c e)^2 / (\sqrt{(f^2 + c_o^2})\}]$ (34)

Similarly for $R_b(S_{c^{e,n-1}}, c'^e)$ we have

$R_b(S_{c^{e,n-1}}, c'^e) = [1- \{\sqrt{(f^2 + c_o^2} - (r_{e(S-c} e) + (n-1) \wedge_{c^e} Z_c e + \wedge_c{}'^e Z_c{}'^e)^2 / (\sqrt{(f^2 + c_o^2})\}]$ (35)

We can assume here that the two components are used verbatim, and thus their Λ's should be 1. Since c^e and c'^e are equivalent, their specifications should be the same. Therefore, $Z_c e = Z_{c'}{}^e$. Substituting these in Equation (34) and Equation (35), we get:

$$R_b(S_{c^{en,n}}) \geq R_b(S_{c^{e,n-1}, c}ie)$$

Hence it proves the property 7.

We can also see from this fact that according to Devanbu et al. (1996) in the second case there is "incorporation of new code which involves the needles additional work to identify, procure and validate the component; therefore, the added extra component should not increase the benefit from reuse."

Property 8: $R_b(S_{c^v}) > R_b(S_{c^m}) > R_b(S_\phi)$

where c^v, c^m and ϕ are components of equivalent functionality. c^v is reused verbatim, c^m is a component which is a modification of another component and ϕ is custom developed. According to this property the reuse benefit is highest for verbatim reuse, followed by modified reuse followed by custom development.

Here we have a component c which is being used three different times with three different ways by implementing of a systems S differently. In first case a verbatim reuse of a component which is represented by c^v, secondly use the modified reuse of an another component which is represented by c^m and finally use the custom developed internal component and is represented by ϕ. This property is not concerned to the number of reuses. Using equations (13) and (14), we can write for verbatim reuse as:

$R_b(S_{c^v}) = [\sqrt{(f^2 + c_o^2}) - \sqrt{(f^2 + c_o^2} - (r_{e(S-c)} + n \wedge_c v \ Z_c v)^2] / [\sqrt{(f^2 + c_o^2})]$ (36)

similarly for modified reuse

$R_b(S_{c^m}) = [\sqrt{(f^2 + c_o^2}) - \sqrt{(f^2 + c_o^2} - (r_{e(S-c)} + n \wedge_c{}^q Z_c{}^m)^2] / [\sqrt{(f^2 + c_o^2})]$ (37)

and for custom reuse

$R_b(S_\phi) = [\sqrt{(f^2 + c_o^2}) - \sqrt{(f^2 + c_o^2} - (r_{e(S-c)})^2] / [\sqrt{(f^2 + c_o^2})]$ (38)

Note that functionality of three components are the same, so $f_1 = f_2 = f_3$, $Z_c{}^m = Z_c{}^v$ and $\Lambda_c{}^v = 1$. Since c^v is used verbatim and $\Lambda_c{}^q$ is less than 1 since c^q is a modified component. Therefore, we can say that

$$R_b(S_{c^v}) > R_b(S_{c^m}) > R_b(S_\phi)$$

Our successful validation of Devanbu (1996) properties under the three dimensional reuse techniques further supports our model. In addition, they are also helpful in many ways such as:

(a) These properties help us in selecting components for the reuse by customers. Customer may be given choices to opt the given components from the available systems. (b) A system developer or customer can identify the properties which has been successful in past in his or her environments. Based on these properties he or she may select classes and objects for his or her recent project. (c) Based on the properties, customers can check and verify at each stage of software development about the behavior of reuse. (d) Externally available reusable components or sub-systems may be selected once the trends in the applications have been identified. (e) Types of the components through the reuse properties may help us in identifying the components or classes which are useful or creating hindrance in the reuse. Because it can be judged whether coupling between components (due to an external or internal components) are increasing or decreasing the systems size? (f) Present model confines the searching and retrieval of components within the space of 'Functionality, Complexity and Reusability'. Use of properties further classifies the components according to our requirements. Therefore, search space is not only reduced, but it becomes easy to customize our applications. (g) As mentioned by Price et al. (1997), in design perspective in a health care system, that "related class hierarchies encourage the designers to group their components into reusable portions at the earliest stages in the design process", suggests that it is possible to find such hierarchies where we have the classes like Person, Patient, Physician, Business, Record etc. as a general classes and which can be further reused in the various small systems.

It is possible to classify these general classes on the basis of the properties mentioned in the three dimensional space which can be grouped into our desired reusable classes and can be further used in our future systems. Classes based on these properties can be suitably grouped for the future reuse in the other

systems. If we know about the characterization of classes based on these properties early in the initial design phase, then we can prevent the undesired coupling between these classes. From our point of view an application is a specific combination of complexity, functionality, and reusability. We can achieve a balanced solution based on our requirements and needs.

8. Component Reuse Benefit and Composition Environment

Component based development (CBD) has been growing considerably among developers, vendors and Information Technology organizations. Reuse of software may be considered as the most effective means for improvement of productivity in software development projects.

Reuse of software generally thought as to increase productivity, improve product reliability and lower overall costs. Four process steps may be used in reusing an artifact (Dusink et al. (1995)) they are (i) find (ii) select (iii) understand; (iv) adopt.

Devanbu et al. (1996) evaluated analytically and empirically "how well several published software reuse metrics measure the "time, money and quality", benefits of software reuse." They assessed several existing software reuse metrics using these properties. Devanbu's work not only reflects the applications of their different analytic properties while it also raised some practical issues.

Reuse Benefit may be optimized during the composition and deployability of software (components) if we use the above mentioned properties. It is also possible to obtain desired deployable software if we follow the one or more properties according to our requirements.

Composing applications out of reusable and pre-existing software components is an important question in creating applications. Through software components we can make a clear distinction between available components and the applications from these components. We need the particular type of component to compose an application.

If we have the multiple applications which share the particular type of components or if we upgrade an application then it becomes quite obvious to use the above properties to optimize and integrate the required application. If we know the exact functionality and complexity during each phase of the SDLC (by using these properties) then we can create a strategy to deploy an application not only safe and successful while we can achieve maximum benefit (in a friendly environment) out of reusable components.

From the model of a component we can determine kinds of components that can be used and composed, and thus influences the properties of the components. Component properties determine how the user interacts with components and composed applications, and thus influence the usability of the environment.

9. Reusable Software Components and Frameworks:

According to DENG-JYI CHEN et al. (2000), we can ideally regard a Reusable Software Component or Framework to be designed for use in constructing many different applications to maximize its applicability and for easy reuse or adaptation by software designers and programmers so that it can ease of tailoring for specific applications. Fig. 2 depicts an ideal RSC or framework in which a 2-dimensional graphic may be used.

Generalization is used to create generalized components which are general to the many applications while Specialization is to create specialized components which can not be used in general. Thus according to authors (DENG-JYI CHEN et al. (2000)) "an RSC or framework must be designed

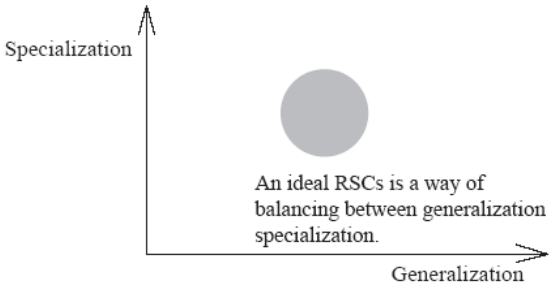

Figure 2
An ideal Reusable Software Component
or Framework (DENG-JYI CHEN et al. (2000)

and implemented so that there is a balance between generalization and specialization". According to the authors (DENG-JYI et al. (2000)) a reusable component behaves like a server while a client (application program) only requires the specification of a server. It does not need to know the details of the services provided by server. Therefore, we can use the reusable components based on our properties for a given system. Once we identify our needs then it is possible to select the components based on the properties. We can optimize the particular type of the properties according to the requirements.

10. Conclusion

Present three dimensional model is more capable and can deal simultaneously several component problems. This model indicates a new way of dealing existing problems and limits the variables required for the component based development. However, a rigorous experimental procedure is desired to verify the present outcome in greater details. We hope to present experimental findings using a visual basic (Microsoft) environment in the future. As a result of present work we propose that: 1. It will be easy to understand the overall reusability in terms of functionality and complexity rather than an elaboration of large number of factors. 2. It removes the ambiguity in selecting parameter "b' " in the Frakes and Terry (1996) formula. Value of "b' " is different for different types of projects. 3. Systems will behave as modular in terms of the proposed model and hence helps us in the calculation of reusability. 4. Standardization and collection of similar components will be an easy task due to atomic character of

OP's. 5. It will provide the description of a complete system where the reusability will be fully explored within the theoretical available limit. 6. It also increases the portability of the system because of limited search.

ACKNOWLEDGEMENTS

The second author's contribution to this paper was in part supported by an ARC Discovery Grant DP0209483.

The first author is grateful to Professor A .S. M. Sajeev, Chair of Computer Science Department and the Head of the School of Computer Science, Mathematics and Statistics, University of New England, Armidale, Australia for his constant guidance and valuable supervision without which this work would have not been possible.

REFERENCES

F. Ayres Jr., Theory and Problems of Differential and Integral Calculus, second ed. New York, N.Y.: McGraw-Hill1, 972.

DENG-JYI CHEN, CHORNG-SHIUH KOONG, WU-CHI CHEN, SHIH-KUN HUANG+ AND N. W. P. VAN DIEPEN (2000), "Integration of Reusable Software Components and Frameworks Into a Visual Software Construction Approach:, JOURNAL OF INFORMATION SCIENCE AND ENGINEERING., 16, 863-884.

Devanbu, P. and Karstu, S. (1994): Measuring the Benefits of Software Reuse, Technical Report, AT&T Bell Labs.

Devanbu, P., Karstu, S., Melo, W. and Thomas, W. (1996): Analytical and Empirical Evaluation of Software Reuse Metrics, Proceeding of Intl. Conf. on Software Engineering (ICSE-18), IEEE, pp. 189-198.

Dusink, L. and Katwijk, Jan van (1995): Reuse Dimensions, Proceedings of the ACM SIGSOFT Symposium on Software reusability (SSR'05)**,** April, pp.137-149.

Fenton, N. E. (1991): Software Metrics: A Rigorous Approach. London: Chapman and Hall.

Frakes, W. and Terry, C. (1996): Software Reuse: Metrics and Models, ACM Computing Surveys, June, pp. 416-435

Hastings, T. E. and Sajeev, A. S. M. (2001): A Vector-Based Approach to Software Size Measurement and Effort Estimation, IEEE Transactions on Software Engineering, April, pp. 337-350.

Hitz, Martin (1995): Measuring reuse attributes in object-oriented systems. In Proceedings of the Int'l Conference on Object-Oriented Information Systems, Dublin, Ireland.

James, M. Neighbors (1996): Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96), p 2. Publisher: IEEE Computer Society Washington, DC, USA.

O'Dell, J. (2004): Indexing software components: A proposal for Enabling Reusing 'Novel'Techniques, Apr-Jun, Vol. 12 Issue 2, p59, 2p; (AN 13049200) April-Jun, pp. 59-60.

Price, Margaretha, W and Demurjuan, Sr., Steven, A (1997): Analyzing and Measuring Reusability in Object-Oriented Designs, Conference on Object Oriented Programming Languages and Applications, Proceedings of the 12th ACM SIGPLAN, Atlanta, Georgia, pp 22-23. Publisher ACM Press New York, NY, USA

Rine, C. David (1997): Success Factors for Software Reuse that are Applicable across Domains and Businesses, Symposium on Applied Computing Proceedings of the 1997 ACM symposium on Applied computing, San Jose, California, United States, ACM Press, pp 182 – 186.

Tracz, W. (1994): Software Reuse Myths Revisited, Software Engineering. Proceedings. ICSE-16., 16th International Conference, Sorrento, Italy, 16-21 May 1994 pp 271-272.

*Professor A.S.M. Sajeev is the Chair of Computer Science and Head of the School of Computer Science, Mathematics and Statistics at the University of New England, Armidale, Australia.

+ O. K. Harsh is associated with University of New England, Armidale, Australia and he has been working under the supervision of Professor A. S. M. Sajeev, Chair of Computer Science Department and the Head of the School of Computer Science, Mathematics and Statistics, University of New England, Armidale, Dr Harsh has been having long experience of working in varieties of fields

Table I

Comparison of Estimated Reuse Benefit to that of Theoretical Calculated Values of Frakes and Terry (1996) formula.

Complexity = 1083 and Functionality = 381. Implementation Language: C++ RDBMS (b' = 0.15)

Project Description (Hastings and Sajeev (2001)	Reuse Rate	Size After Reuse $am_1^b g_1^z$	Original Size $am^b g^z$	Reuse Benefit using Present Formula (Equation	Reuse Benefit using equation (17) Frakes and Terry(1996) formula
Application Type: Management Information Delivery Platform: MSWindows	10 %	3111.9005	3252	0.0431	0.0581
	20%	2776.5356	3252	0.1462	0.1163
	30%	2407.3984	3252	0.2597	0.1744
	40%	2088.4415	3252	0.3578	0.2325
	50%	1836.5335	3252	0.4353	0.2906
	60%	1644.4302	3252	0.4943	0.3488
	70%	1495.5141	3252	0.5401	0.4069
	80%	1380.1121	3252	0.5756	0.4650
	90%	1289.4386	3252	0.6035	0.5231

Table II

Comparison of Estimated Reuse Benefit to that of Theoretical Calculated Values of Frakes and Terry (1996) formula.

Complexity = 348 and Functionality = 65. Implementation Language: Assembler (b' = 0.15)

Project Description (Hastings and Sajeev (2001)	Reuse Rate	Size After Reuse $am_1^b g_1^z$	Original Size $am^b g^z$	Reuse Benefit using equation (21) Present Formula	Reuse Benefit using equation (17) Frakes and Terry(1996) formula
Application Type: Control Systems Delivery Platform: Embedded System	10 %	1743.13438	2034	0.1430	0.1090
	20%	1273.43275	2034	0.3739	0.2181
	30%	912.621427	2034	0.5513	0.3271
	40%	677.675113	2034	0.6668	0.4361
	50%	510.140423	2034	0.7492	0.5451
	60%	390.222177	2034	0.8082	0.6542
	70%	295.51451	2034	0.8547	0.7632
	80%	212.683514	2034	0.8954	0.8722
	90%	135.680524	2034	0.9333	0.9812