Intelligent Risk Analysis Model for Mining Adaptable Reusable Component

Iyapparaja Meenakshisundaram[1] and Sureshkumar Sreedharan[2]
[1]School of Information Technology and Engineering, VIT University, India
[2]Vivekanandha College of Engineering for Women, Anna University, India

Abstract: *Every elucidation for today's quandary has been achieved in an easier prospect, with due respect to the experience gained by a normal man. The engineers too look out for the better way in the development cycle of software apart from its traditional approach. Software being implemented in almost every machine, is in the urge of being developed with many improvisation techniques but obeying the time and cost constrains. Adding to the available simplifications methodologies in the development phases, the proposed Intelligent Risk Analysis Model (IRAM) would abridge the limitations of an Object Oriented Program (OOP) developed for a new software product showing betterments in time and budget needed. An OOP would comprise of individual and exclusive objects with indicated functionalities. Recognizing the usage of the objects in the existing programs would eliminate the necessity of a new coding, thus the component could be reused if it cannot be designated any better. This methodology does a primary verification whether there are any components which match with the stated requirements in the database of programs (e.g., C++, Java, Perl and Python). Based on the analysis of the matched component, it is categorized into Exact Match (EM), Partial Match (PM) or the Rejected Match (RM) which denotes its chances of applicability into the new product. This analysis of the correspondence in the reused object depends on the defined four parameters tuple namely Expected Language (EL), Module Description (MD), Argument Description (AD) and the Usage Threshold (UT). The component that matches exactly EM can be directly incorporated into the new software product whereas if the component falls into the other category PM then it is subjected to additional tests, Rank (R) is allotted, Intelligent Report (IR) is prepared and measures for its updating as an EM are taken. The RM component is eliminated from the list of possible outcomes at once.*

Keywords: *Software engineering, software reusability, OOP, IR, cohesion and coupling, regression test.*

Received February 3, 2013; accepted September 9, 2014; published online August 16, 2015

1. Introduction

The software has been devised with the intention of reducing the workload, time and cost metrics. But the manpower and the resources required for the software development itself had to involve the mightier and expertise, obey the strict principles and the top of all to satisfy the end user. Despite many simplifications, the development phases need proper follow-up and alternative plans for maintaining the product on the right track. Any minor change/mistake in the proposed plan would cost the developer his entire effort to a waste [14].

The software development phases (analysis, design, coding, testing and implementation) include dedicated functionalities of each phase, organized at the last would yield the desired software product. The Analysis phase observes the requirements of the user/customer and the design phase is for the developer's team to design the best plan to carry out. The coding phase is for the switching into machine level code [14]. The testing is to obtain the conditions in which the product works and fails (under predicted conditions) [12]. Testing is secondly to ensure the reliability of the software in feasible extremes. Implementation is to establish the developed product in the original environment it is supposed to be [1].

2. Testing Object Oriented Programs

The Object Oriented Programming (OOP) has introduced new innovative and much easier attitudes to design the software product, diverse from that of the traditional programming disciplines. Adding to the advantages, reduced time to be designed and ease of structure, promotes its practice among the recent programmers [6, 12]. The OOP introduces out of the ordinary concepts such as encapsulation, inheritance, polymorphism and data abstraction. Inheritance helps to promote the reusability factor, in turn helping for the development of the software more rapidly [4, 8].

Reusability factor includes along with its merits, the risk of unstable conditions in the new environment [17]. The existing environment may be the best platform and the new platform requires some reformation to the coding in order to make it adapt with the new environment [1, 8]. Hence, a risk analysis model is obliged to eradicate the limitations and promise the compatibility of the reusable component [3, 4].

3. Proposed: Intelligent Risk Analysis Model (IRAM)

The urge of a suitable Risk analysis model among the numerous models [2, 10, 11], motivates the design of

the proposed IRAM. Not all risk analysis models judges the risks of a component/module to be reused [13, 17]. This model IRAM ascertains the act of mitigating the risks associated with reusability.

The succeeding Figure 1. Illustrates the IRAM models which identify the reusable module from the warehouse, based upon the progression of phases are listed below.

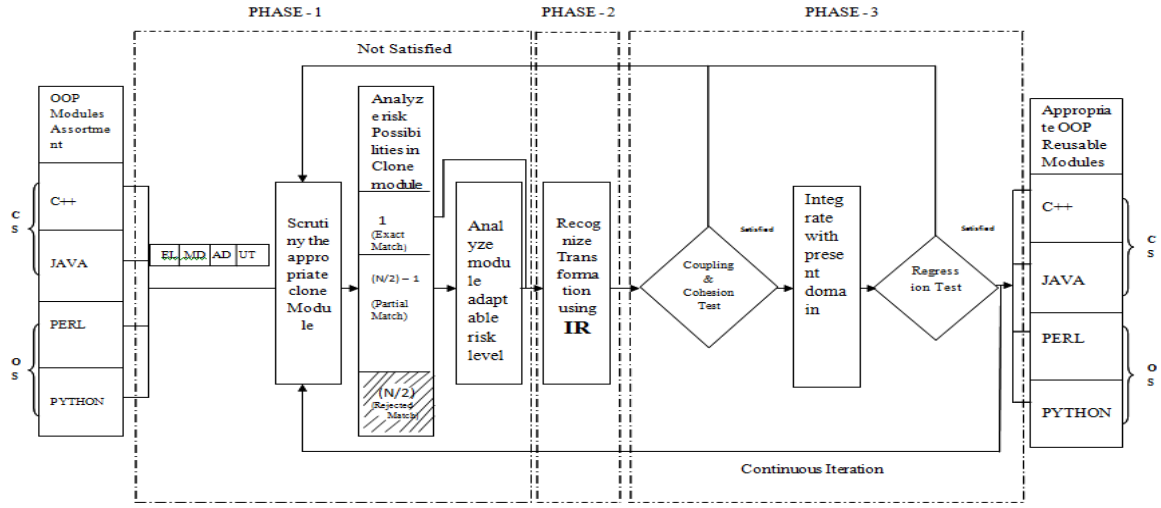

Figure 1. IRAM.

Where Closed Source (CS), Open Source (OS), OOP, Expecting Language (EL), Module Description (MD), Argument Description (AD), Usage Threshold (UT), Number of Possibilities (N).

- *Phase* 1:
  1. Search for the most similar module with affirmed parameters in the warehouse.
  2. The probability of the match categorizes the analyzed modules into defined groups (Exact Match (EM), Partial Match (PM), Rejected Match (RM))
- *Phase* 2:
  3. The Intelligent Report (IR) estimates the alterations required in a *PM* to convert into an *EM* module.
- *Phase* 3:
  4. The dependency metrics of each module with neighboring modules has to be determined by coupling and cohesion tests.
  5. The level of applicability in the new platform is evaluated by the regression test.
  6. After validation of a module in all these tests, the reusable component is proved to be risk free and can be implemented.

4. Module Description

4.1. Phase 1: OOP Reusable Modules Assortment

The reusable component requires some serious modifications in its coding for adaptability. Not all the components extracted from its original environment can be directly implemented in the new environment. Even the OOPs and their modules face difficulties in their applicability level.

To conquer these precincts, IRAM would extend its support beyond measuring the reusability related risks. IRAM would advise the revisions needed after a compilation of tests. For convenience, two CS OOP languages (C++ and Java) and two OS OOP languages (PERL and PYTHON) are used. The list of assessments and results are discussed as follows.

4.2. Phase 1: Scrutiny of Reusable Modules

There is an immense volume of modules to be compared with the listed parameters. This comparison otherwise, analysis is done by the four tuples as illustrated in Figure 2. This analysis checks the required parameters with the available parameters and forwards the result to the IRAM.

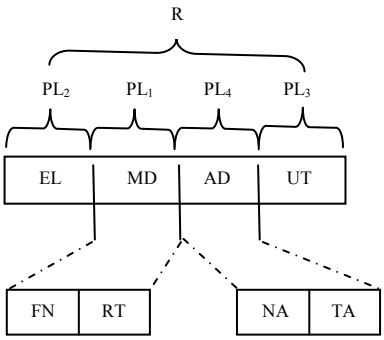

Figure 2. Four parameters tuple with Priority Level (PL).

The EL for the new software product is being developed (C++, Java, PERL, and Python); MD for further divided into Function Name (FN) and Return Type (RT) for the completion of a particular task and the variables used. The AD for which further divided into Number of Arguments (NA) used and TA for type of Arguments describing, the data type of the processed variables in the extracted module. UT for of

the function, denoting the importance of a reusable function based upon the frequent repetition of the same called by the module itself. In this Model, the threshold value is assigned by the developer who intends to reuse a matching component. In general, the UT value is assigned by the size/Line of Code (LOC) present in the reusable module, like:

1. LOC (Reusable module)≤ 200 then set UT value is UT=2.
2. LOC (Reusable module)> 200 then set UT value is 2≤ UT≤ 5.

With the requirements of a new software product, every module is analyzed for similarity constraints with the parameters tuple as configured with .xml coding as follows:

Source code of Parameter_config.xml:

```
“”
<?xml version=“1.0” encoding="utf-8" ?>
<configuration>
<configSections>
<sectionGroup name= “parameters”, type=“
System.Configuration.UserSettingsGroup”>
<sectionname=“EL&&MD&&AD&&UT”
type=“System.Configuration.ClientSettings Section/>
 </sectionGroup>
</configSections>
  <connectionStrings>
     <addname=“EL&&MD&&AD&&UT.Settings.DBConnection
String”, connection String=“DataSource=mix1;
InitialCatalog=DB;IntegratedSecurity=True”
 providerName=“System.Data.SqlClient”/>
 </connectionStrings>
 <userSettings>
     <EL&&MD&&AD&&UT/>
 </userSettings>
</configuration>
```

In this model, the fixation of PL to each parameter as MD-> PL_1, EL-> PL_2, UT-> PL_3 and AD-> PL_4 is needed, based upon its importance metric during the search of reusable module.

As per Set Theory, Property: If a set S has *n*-elements, then its power set has 2^n elements, viz., [16].

$$If\ |S| = n,\ then\ |P(S)|=2^n \quad (1)$$

According to this property, form the set which contains mixture of parameters with distinct PL as illustrates in following Table 1.

Table 1. Distinct set combinations.

Set No.	Distinct Set Combinations
S1	{}
S2	{ UT -> PL_3 }
S3	{ AD -> PL_4 }
S4	{ AD -> PL_4, UT -> PL_3 }
S5	{ MD -> PL_1 }
S6	{ MD -> PL_1, UT -> PL_3 }
S7	{ MD -> PL_1, AD -> PL_4 }
S8	{ MD -> PL_1, AD -> PL_4, UT -> PL_3 }
S9	{ EL -> PL_2 }
S10	{ EL -> PL_2,UT -> PL_3 }
S11	{ EL -> PL_2, AD -> PL_4 }
S12	{ EL -> PL_2, AD -> PL_4, UT -> PL_3 }
S13	{ EL -> PL_2, MD -> PL_1 }
S14	{ EL -> PL_2, MD -> PL_1, UT -> PL_3 }
S15	{ EL -> PL_2, MD -> PL_1, AD -> PL_4 }
S16	{ EL -> PL_2, MD -> PL_1, AD -> PL_4, UT -> PL_3 }

4.2.1. Analysis of Acceptable Risk levels of Reusable Modules

Searching the expected module from the reusable modules warehouse using Get() method with Parameter_config.xml file as follows,

```
// Get the module of the request parameters;
String name = “Parameter_config.xml”;
String value = req. getParameter(name);
 if (value == null)
    {
// The request parameter "EL && MD && AD && UT" was  not
present in the module set
   }
 else if (“ ”.equals(value))
        {
// The request parameter “EL && MD && AD && UT” was
present in the module set but has no value
        }
// Get the module of all request parameters match
Enumeration enum=req.getParameterNames
(Parameter_config.xml);
for (; enum.hasMoreModules();)
    {
 // Get the name of the request parameter
 name = (Object)enum.nextModule();
// Since object type casting supports all datatypes.
 out.println(name);
 // Get the module of the request parametermodule
=req.getParameter(name);
// If the request parameter can appear more than once in the
modules set, get all modules
Object [] modules = req.getParameterValues(name);
for (int i=0; i< modules. length; i++)
    {out.println("   "+module[i]);}}
```

The results of the evaluations of mentioned four parameters ($|S|=4$) would produce $|P(S)|=2^4=16$ possible outcomes as per Equation 1 and they are described as follows in Table 2.

Table 2. Acceptable risk possibilities of reusable module.

Set No.	EL	MD	AD	UT	Acceptable Possibilities
S1	0	0	0	0	RM
S2	0	0	0	1	RM
S3	0	0	1	0	RM
S4	0	0	1	1	RM
S5	0	1	0	0	PM
S6	0	1	0	1	PM
S7	0	1	1	0	PM
S8	0	1	1	1	PM
S9	1	0	0	0	RM
S10	1	0	0	1	RM
S11	1	0	1	0	RM
S12	1	0	1	1	RM
S13	1	1	0	0	PM
S14	1	1	0	1	PM
S15	1	1	1	0	PM
S16	1	1	1	1	EM

Here, 1 represents parameter match and 0 represents parameter not match.

These outcomes would help to order them into three groups based on their level of adaptability. There is always one distinct expected match module among the 16 possibilities. The *EM* is the best fitted reusable module satisfying all the needs of the new product. Yet it is subjected to some tests for evaluating the adaptability level. There are seven possibilities for a

PM and the remainder eight possibilities *RM* can be never taken into count that is they are simply discarded. Let consider:

- $n1$=Number of *EM* -reusable modules in a database.
- $n2$=Number of *PM* -reusable modules in a database.
- $n3$=Number of *RM*- reusable modules in a database.
- N=Total number of reusable modules in a database.

i.e., $N=n1+n2+n3$

According to Combinatorial Probability [7]:

$$P(A) = \frac{Number\ of\ favorable\ case}{Number\ of\ exhaustive\ cases} \quad (2)$$

Where A: An event occurs.

When applying an event "Suitable Module Drawn" in above Equation 2. We will get:

1. Probability of getting a *EM* module from N number of possible cases is:

$$P(EM) = \frac{n1C_1}{NC_1} = \frac{n1!/(n1-1)!}{N!/(N-1)!} \quad (3)$$

2. Probability of getting a *PM* module from N: number of possible cases is:

$$P(PM) = \frac{n2C_1}{NC_1} = \frac{n2!\ /\ (n2-1)!}{N!\ /\ (N-1)!} \quad (4)$$

3. Probability of getting a RM module from N: number of possible cases is:

$$P(RM) = \frac{n3C_1}{NC_1} = \frac{n3!\ /\ (n3-1)!}{N!\ /\ (N-1)!} \quad (5)$$

This test of the IRAM determines merely the possible outcomes and the categorizing of those outcomes into three groups (*EM*, *PM*, and *RM*). There are supplementary actions and tests to verify and produce the final reusable module.

According to the different (0/1) combinations and set combinations in the above Table 2 and Table1. Assign the Rank (R) to each combination set according to the highest PL-Parameters which contains. It describes in following Table 3.

Table 3. Assigning R to each set combination.

Set No.	PL- Parameters in each Set				R
S1	-	-	-	-	16
S2	-	-	-	PL_3	14
S3	-	-	PL_4	-	15
S4	-	-	PL_4	PL_3	13
S5	-	PL_1	-	-	8
S6	-	PL_1	-	PL_3	6
S7	-	PL_1	PL_4	-	7
S8	-	PL_1	PL_4	PL_3	5
S9	PL_2	-	-	-	12
S10	PL_2	-	-	PL_3	10
S11	PL_2	-	PL_4	-	11
S12	PL_2	-	PL_4	PL_3	9
S13	PL_2	PL_1	-	-	4
S14	PL_2	PL_1	-	PL_3	2
S15	PL_2	PL_1	PL_4	-	3
S16	PL_2	PL_1	PL_4	PL_3	1

4.3. Phase 2: IR

The availability of only one outcome cannot prove this Model to be a fruitful one. Hence, the possible outcomes should be reasonable to withstand its betterment. The IR endow with the increased number of the possible outcomes with further analysis and actions.

4.3.1. Analysis of Adaptable Risk Level Module

The IR testifies that the *PM* modules can be converted into *EM* modules if performed with specific transformations in the coding of the original component. The *PM* module is divided into individual literals and tokens and compared with the requisites of the expected OOP language. This comparison generates the IR, and the IR recommends the amendments in order to produce the *EM* module.

A same process would have a unique way of presentation in different OOP languages. Hence, the transformation of the reusable component in one language to the resultant OOP language would require the changes in the Syntax and Semantics.

- *Example* 1:

Let consider, a bubble sort code in C++ Language. The same code will look with different syntax and semantic format in different OOP- Languages (Here, Java, Perl and Python).

The following Table 4. Would make clear this condition. Table 4 shows that, the supreme need of the IR to mark the syntax which needs attention for adaptability.

Table 4. Sorting module program in different OOP language.

C++	Java	PERL	Python

Void sort (int a[], int len)
{
int temp;
for(int i=0;i< len;i++)
{
for(int j=0;j<len – i;j++)
{
if(a[i]<a[i+1])
{
temp = a[i];
a[i]=a[i+1];
a[i+1]=temp;
} } } }
	Public static void sort (int a[], int len)
	{
	int temp;
	for(int i=0;i< len;i++)
	{
	for(int j=0;j<len – i;j++)
	{
	if(a[i]<a[i+1])
	{
	temp = a[i];
	a[i]=a[i+1];
	a[i+1]=temp;
	} } } }
		sub Sort (my @a = @_ ,
		my $len)
		{
		my $temp;
		for my $i(0 .. $len)
		{
		for my $j(0 .. $len - $i)
		{
		if ($a[$i] < $a[$i+1])
		{
		$temp = $a[$i];
		$a[$i] = $a[$i+1];
		$a[$i+1] = $temp;
		} } } }
			def Sort (a, len):
			
			a = list(a)
			for j in range(len(a)-1, 0, -1):
			
			for i in range(j):
			
			if a[i] < a[i + 1]:
			
			a[i] = a[i + 1]
			a[i + 1] = a[i]
			return a

4.4. Phase 2: Recognize Transformation with Mitigating Risk

Since, then the analysis to determine level of applicability into the new product is shown. The successive process is to determine the risk level and conclude about its implementation stage.

The implementation of the reusable component is considered to be a successful level or the breakdown level. Initially the numbers of errors are checked and it should possess a value equal to zero. A value of the existing OOP module should be equal to the Expected OOP module in order to be a successful level. Otherwise, the implementation is at the breakdown level.

4.5. Phase3: Dependency of the Module

Every module which is tested for reusability is computed for its dependency with the other components. An OOP may be designed with numerous objects related to each other and a level of dependency on the other objects for their function. An object may or may not work independently. This factor also needs considerations as incomplete programs (dependent on other sub programs) or unwanted coding (independent but provides results to other sub-programs) may affect the performance of the system.

4.5.1. Cohesion and Coupling Tests

The successful identification of a reusable module is nevertheless adequate for reusability. The coupling and cohesion metrics have to be evaluated to promise its intended function with or without the help of other components.

Coupling denotes the dependency of one module on any other module for its specified task. If the depended process is not complete, then the module cannot work.

Cohesion is the measure of individual strength and reliability of a particular module to perform its task independently [5].

In order to maintain the cohesion and coupling level of the module, we have been using different metrics, like, Lack of Cohesion of Methods (LCOM), Tight and Loose Class Cohesion (TCC and LCC) method for testing the cohesion level along with Afferent, Efferent coupling, loose and tight coupling metrics used for measuring the stability level of the module but all these metrics are only focusing on the following interdependency among the modules [5] as shown in Figure 3.

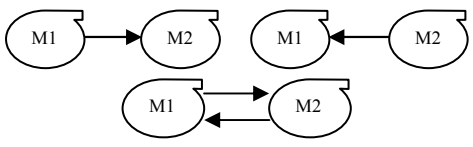

Figure 3. Interdependency among the two modules.

In addition to these, it is significant to test the following interdependency among the multiple numbers of modules as shown in Figure 4.

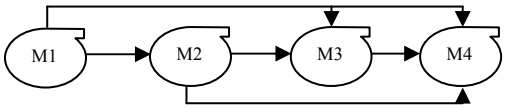

Figure 4. Interdependency among the multiple modules.

This relation shows that, when M1-> M2, M2-> M3 and M3->M4 then it is obvious to exist the dependency relation between M1-> M3, M4 and M2->M4.

Once the cohesion and coupling value of the module is calculated, if we get high cohesion and low coupling then it will be moved in to subsequent phase otherwise, reject the module and start the searching process a new [9].

4.6. Phase 3: Integration with Present Software Domain

These analysis and test results would finally yield the much awaited reusable module for integrating it into the new software product. Successful implementation would incorporate the module with the recent product and functions as a whole system.

4.7. Phase 3: Regression Testing

The need of Regression testing arises to confirm whether the new product is working normally as it should [15]. The new product may lead to certain errors which may not have been predicted by the developer. The regression test would eliminate the probability of errors before the product reaches the customer.

If there is an occurrence of an error, the remedy is taken at once or the whole process is restarted in case of system failures. This test is further used to determine the size of code, the performance time and the outcomes of the test cases.

4.8. Appropriate OOP Reusable Module

Finally the reusable component is revised, reviewed and thoroughly verified to perform its existing function in the new environment with the least or no possibility of errors, satisfying the customer and the end user. The point where this model gains advantage is in a complex program of thousands of lines of coding. Reusable components would perform the predefined function in respective platforms with fewer changes in their discipline and conserving the time and resources needed for creating anew.

5. Algorithm for IRAM

n:= Number of chances;
rcount:=reusable count;
ER:=Expecting reusable module;
ER:=Existing reusable module;
EOOP:=Existing Object oriented program;
POOP:= Present Object oriented program;
EL:=Expecting Language;
MD:=Module Description;

```
AD:=Argument Description;
UT:= Usage Threshold;
CCT:=Cohesion & Coupling Test;
RT:=Regression Test;
IR:=Intelligent Report;
Begin
rcount :=0;
Tot_space:=  allocated_memory(EOOP.ER);
for (i:=0; i<=n; i++)
New: Search(er.(EL,MD,AD,UT)==Tot_space;
    if(er.EL==1&&er.MD==1&&er.AD==1&&er.UT==1) then
          Set ER := EM;
          Test: Direct ER -> CCT; /* Perform Cohesion&
          Coupling test with ER*/
         if(er==1) then
              ER:=ER+POOP;  /*Integrateer with Present */
              Direct er+OOP -> RT; /*Perform Regression test
              with ER+POOP*/
                if((er+POOP)==1)then
                    rcount:= rcount+1; /*Fit into Present OOP &
                    increase reusable count value into to 1*/
                 else
                     goto New; /*Start new iteration*/
                     break;
          else
               goto New; /* Start new iteration */
               break;
    else if (er.MD==1&&(er.EL==1||er.AD==1|| er.UT==1) then
          Set ER := PM;
          Direct ER -> Analyze module risk level phase;
          Compare(P.EOOP(er)==P.POOP(er)); /*Compare the
          parameters of er in EOOP with er in POOP*/
          Generate IR; /* IR will be prepared based upon the syntax
          and semantic analysis*/
          Make ER.PM:= ER.EM; /* Successful Level */
          goto Test; /* Start testing iteration */
          break;
  else if (ER.MD==0&&(ER.EL==1||ER.AD==1||
  ER.UT ==1) then
          Set ER:= RM; /* Breakdown Level */
          goto New; /* Start new iteration */
         break;
else
    Goto New; /* Start new iteration */
    break;
end if;
end for;
end
```

6. Result and Discussion

- *Acknowledgements in the TROY Software* [*India*] *Pvt., Ltd.,*

The proposed IRAM model helps to identify the best reusable module in OOP environment with repeated tests and severe analyses.

The following tables reveal that the component "int swap (int, double). py" with minimum UT value = 2.

Here, *EL* = "*Python*"; *MD -> FN* = *"swap*()";
MD -> RT = int ; AD -> NA = 2;
AD -> TA = int, double;
UT = 2, *Since LOC* < =200.

As this requirement, the IRAM model displays all possible modules related to our search as follows in Table 6.

Table 6. Possible reusable modules set.

Component No.	Language	MD		AD		UT
		FN	RT	NA	TA	
C1	Python	swap()	int	1	float	0
C2	Java	swap()	int	2	int, double	0
C3	C++	swap()	int	3	float, int, double	1
C4	Perl	swap()	int	1	double	2
C5	Python	swap()	int	2	double, double	2
C6	Python	swap()	int	2	int, double	0
C7	Java	swap()	int	2	int, double	2
C8	C++	swap()	int	2	int, int	5
C9	Java	swap()	int	4	int, double, double, float	2

Depending on the results, available N from Table 6 is categorized into three groups (*EM*, *PM*, RM) as shown in Table 7.

Table 7. Components group formation.

EM	PM	RM
Nil	C1	Nil
	C2	
	C3	
	C4	
	C5	
	C6	
	C7	
	C8	
	C9	
	C10	

According to this categorization, identify the number of parameter matches that exist in each component as shown in Table 2, assign the corresponding PL level to each matched component. And R it based upon the Table 3. This process is clearly stated in following Table 8.

Table 8. Identifying suitable component among N-number of possible components.

Component	Combinations (0/1)				PL				R
	EL	MD	AD	UT					
C1	1	1	0	0	PL_2	PL_1	-	-	4
C2	0	1	1	0	-	PL_1	PL_4	-	7
C3	0	1	0	0	-	PL_1	-	-	8
C4	0	1	0	1	-	PL_1	-	PL_3	6
C5	1	1	0	1	PL_2	PL_1	-	PL_3	**2**
C6	1	1	1	0	PL_2	PL_1	PL_4	-	3
C7	0	1	1	1	-	PL_1	PL_4	PL_3	5
C8	0	1	0	0	-	PL_1	-	-	8
C9	0	1	0	1	-	PL_1	-	PL_3	6
C10	1	1	0	0	PL_2	PL_1	-	-	4

According to the outcome of the above Table 8, the components are sorted upon their Rs and the component possessing the highest R is selected for generating IR as shown in Table 9.

Table 9. Component R order.

Component	R	R Order	Component Order
C1	4	**2**	**C5**
C2	7	3	C6
C3	8	4	C1
C4	6	4	C10
C5	**2**	5	C7
C6	3	6	C4
C7	5	6	C9
C8	8	7	C2
C9	6	8	C3
C10	4	8	C8
Identifying Suitable Component			
Component	**Highest R**	**Status**	
C5	2	**Suitable component for generating IR**	

This can be represented in the following Figure 5.

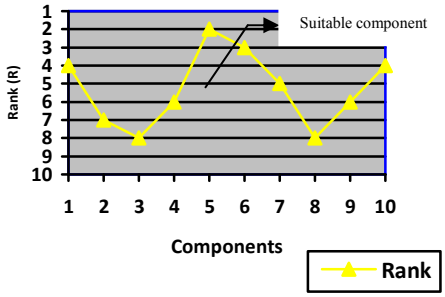

Figure 5. Identifying suitable component.

Taking the results of Tables 6, 7, 8, 9 and Figure 5. Into consideration, the numbers of possible modules are organized into three groups (*EM*, *PM*, RM). Ranking the component based upon its matched parameters PL. Suitable module which is rated to be the best among the N is selected. And an IR is generated, by subjecting the opted component into severe coupling, cohesion and regression tests to reform the selected into the best fitted adaptable reusable component.

7. Conclusions and Scope for Future Work

This paper presented an innovative model to handle the challenging methodology of reusability in OOP environment and proving that it would ensure the performance boost by inheriting suitable component from existing assortment. Testing technologies for the selected module would tend to eliminate the risk factors and errors leading to the support of the reusable module.

This concept has to extend to be functional with all OOP environments and this model would be capable of performing in cloud computing.

References

[1] Divya C. and Rajender Singh C., "Component Based Software Engineering Systems: Process and Metrics," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 7, pp. 91-95, 2013.

[2] Foo S. and Muruganantham A., "Software Risk Assessment Model," *in Proceeding of IEEE International Conference on Management of Innovation and Technology*, pp. 536-544, 2000.

[3] Hosseingholizadeh A., "A Source-based Risk Analysis Approach for Software Test Optimization," *in Proceeding of the 2nd International Conference on Computer Engineering and Technology (ICCET)*, Chengdu, China, pp. 601-604. 2010.

[4] Ivar J., *Object-Oriented software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1996.

[5] Iyapparaja M. and Sureshkumar S., "Coupling and Cohesion Metrics in Java for Adaptive Reusability Risk Reduction," *in Proceeding of IET Chennai 3rd International Conference on Sustainable Energy and Intelligent Systems*, Tiruchengode, India, pp.1-6, 2012.

[6] Jalender B., Govardhan A., and Premchand P., "Designing Code Level Reusable Software Components," *International Journal of Software Engineering & Applications (IJSEA),* vol. 3, no. 1, pp. 219- 229, 2012.

[7] Kandhasamy P., *Probability Statistics and Queueing Theory,* S.Chand publication, First Edition, 2004.

[8] Kirandeep K., Rekha R., and Jagdeep k., "Code Reuse and Reusability of the Software," *The International Journal of Engineering and Science (IJES)*, vol. 2, no. 4, pp. 28-30, 2013.

[9] Kuljit K. and Hardeep S., "An Investigation of Design Level Class Cohesion Metrics," *the International Arab Journal of Information Technology*, vol. 9, no.1, pp. 66-73, 2012.

[10] Muhammad O., Ahmed M., and Ahsan S., "Optimal Performance Model Investigation in Component-based Software Engineering (CBSE)," *American Journal of Software Engineering and Applications,* vol. 2, no. 6, pp. 141-149, 2013.

[11] Nida Y., Bushra J., and Javed F., "PDCML: A Model for Enhancing Software Reusability," *International Journal of Software Engineering and Its Applications,* vol. 7, no. 1, pp. 123-136, 2013.

[12] Paul C., *Software Testing, A Craftsman's Approach*, 3rd Edition, Auerbach Publications, 2011.

[13] Poonam k., "Software Effort Estimation and Risk Analysis-A Survey," *International Journal of Engineering and Innovative Technology*, vol.1, no. 1, pp. 18-22, 2012.

[14] Roger S., *Software Engineering: A Practitioner's Approach*, 7th Edition, McGraw-Hill International Edition, 2010.

[15] Suhaimi I., Norbik B.I., Malcolm M., and Aziz D., "Integrating Software Traceability for Change Impact Analysis," *the International Arab Journal of Information Technology*, vol. 2, no. 4, pp. 301-308, 2005.

[16] Tremblay J. and Manohar R., *Discrete Mathematical Structures with application to Computer Science*, McGraw Hill, Inc., 1989.

[17] Wang A., "Reuse Metrics and assessment in Component-based Development," *in Proceedings of Software Engineering and Application*, pp. 693-707, 2002.

Iyapparaja Meenakshisundaram He received the BE degree from Anna University, Chennai and ME degree from Anna University of Technology, Coimbatore in 2006 and 2010 respectively. Presently, he is a Senior Assistant Professor in School of Information Technology and Engineering. He has 7 years of experience in Teaching and software Engineering field. He received University Rank holder award for his ME degree. His research interests include Software Testing, Software Engineering and Agile Testing. He is life time member of ISTE.

Sureshkumar Sreedharan is presently the Principal, Vivekanandha College of Technology for Women, Tiruchengode. He received the BE degree from National Engineering College, MS, degree in Software system from Birla Institute of Technology and Science, MTech., degree from Indian Institute of Technology, Kharagpur and PhD degree from Anna University, Chennai in 1988, 1993, 2000 and 2009 respectively. He has published 30 numbers of papers in refereed international journals and conferences. His research interests include Image processing, parallel processing and Energy. He is a member of ISTE and IET.