Journal of Engineering Technology
Volume 6, Issue 1, Jan. 2018, PP. 365-377

RESEARCH ARTICLE -ENGINEERNG TECHNOLOGY

Reusability Strategy Based on Dynamic Reusability Object Oriented Metrics

Eun Young Byun, Hyuen Seung Son, Byungkook Jeon* and R. Young Chul Kim

SE Lab., Dept. of CIC(Computer and Information Communication),

Hongik University, Sejong Campus, 339-701, Korea

*Dept. of Software, Gangneung-Wonju National University, Wonju City, Gangwon Prov. 26403 Korea

Abstract: As incredibly expanding our software market scale in diverse areas, reusability is indispensable to rapidly develop good quality of software. Several researches were focusing on model/object/service/product among reusable maturity levels. Most of these researches are based on static analysis. Our approach considers how to enhance reusability on both static & dynamic analysis. To do this, we propose dynamic reusability metrics (DRM), and apply code visualization to identify reusability maturity level of method/class/class-object with inheritance/associations on a software system. On developing a new project with any reusable codes within the legacy software, we expect to reduce time and cost of development, and to enhance productivity of software.

Keywords— Reusable unit level; Reusability metrics; Dynamic analysis; Reusability strategy; Code visualization

1. Introduction

Recently, software is used in social and cultural fields as well as various IT fields. As the software market scale expands, it is likely to increase the need of large software that performs various functions. Even through frequent changes of requirements, it keeps on the productivity and good quality of software [1, 2]. For achieving these, software engineers are considering of using the methodology, software architecture, process improvement, and reuse [3]. Among these, we are focusing on the reusability as the unit expanded from the low level of module to the high level of product [4].

In this paper, we propose the reusability strategy for identifying a reusable module or multiple modules as a chunk in the legacy system. We also define reusability metric based on static/dynamic analysis to measure reusability at method and object level among various reusability maturity levels. We use reusability factors such as cohesion, coupling, memory allocation size and frequency of object. We propose the reusability strategy to automatically identify reusable modules by applying the metrics. Reusable approach can reduce time and cost of development, and enhance productivity and quality of software.

The chapter 2 describes related work, and chapter 3 defines reusability metric at method and object levels. The chapter 4 explains the reusability strategy for the metrics and code visualization mechanism to automatically identify reusable modules. The chapter 5 shows how to identify reusable modules through our code visualization on the reuse strategy with one example of the target program. Finally, we mention conclusions and future work.

2. Related work

So far, many papers and researches have been mentioned several studies to measure reusability[5-12]. Fig. 1 shows the number of some factors to measure reusability[5]. However, the most previous studies have been focused on static analysis rather than dynamic analysis. Therefore we perform both of them. In static analysis, modularity is checked by typically using the cohesion and coupling. Most of the previous studies used CK metrics to measure these factors. Cohesion is measured by Lack of Cohesion in Methods (LCOM), which is an object oriented metric used to measure the cohesiveness of a class. Coupling is measured by Coupling Between Objects (CBO), which is a count of the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the variables of the other[13]. Dynamic analysis measures the object allocation size and frequency.

3. Reusability Metrics

In static analysis, modularity is measured based on cohesion and coupling. Dynamic analysis measures the size and frequency of object allocation. The reusability metrics is defined with both results of static analysis and dynamic analysis. The reusability with some relationships among the factors is shown that the higher cohesion and object allocation frequency and the smaller coupling and object allocation size, the more suitable for reuse [14-17]. The factors and the metrics are as follows.

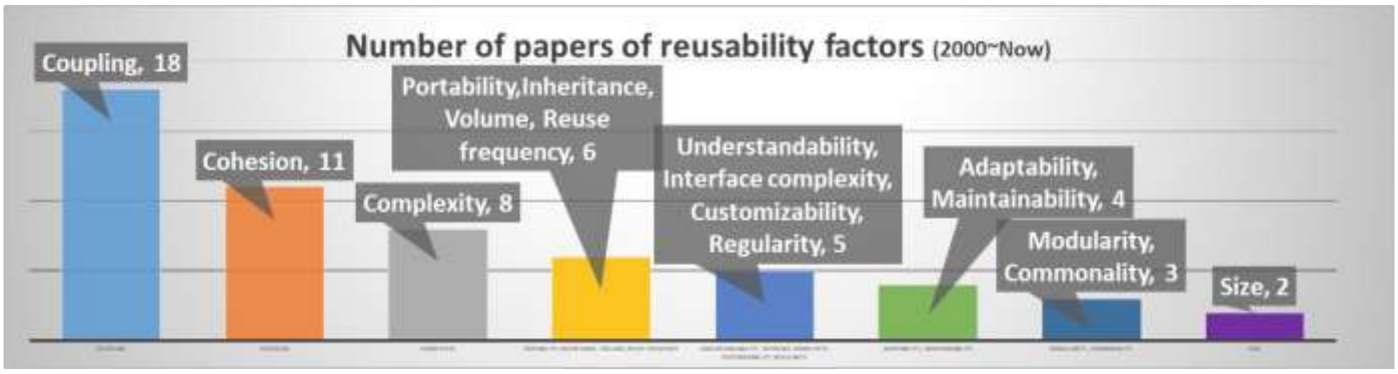


Fig. 1. Number of papers of reusability factors [5]

3.1 Cohesion

Cohesion is a measure of the affinity of components within a module. The higher cohesion of a module, the higher the probability that the components necessary for the module exist in the module. The lower cohesion of a module, the higher the probability that unrelated components exist in the module. The highly cohesive modules can perform a single task within a module with little or no interaction with other modules. The cohesion is classified into seven categories as functional, sequential, communicational, procedural, temporal, logical, and coincidental cohesion. Table 1 is the definition of types of cohesion [18].

Table 1 Definition of Cohesion [18]

Cohesion	Definition
Functional Cohesion	All components in the module are associated with one function.
Sequential Cohesion	The output of one component in the module becomes the input of another component.
Communicational	The module performs diversity functions, and the components in the module use the

Cohesion	**Definition**
Cohesion	same input data or output the same output data.
Procedural Cohesion	The components in the module are associated and performed in a specific order.
Temporal Cohesion	The components in the module perform different functions at the same time.
Logical Cohesion	The components in the module perform a logically related task or similar function.
Coincidental Cohesion	The components in the module are not related to each other.

According to the Table 1, the higher cohesion is, the more suitable for modularization (Functional Cohesion) is. Conversely, the lower cohesion is, the less suitable for modularization (Coincidental Cohesion) is.

3.2 Coupling

Coupling is a measure of how much a module depends on another module. The lower coupling is, the less relevance to the external module is suitable for modularization. The higher coupling, the higher the relevance to the external module, which is not suitable for modularization of the corresponding modules. The types of coupling are classified into six categories as content, common, external, control, stamp, and data coupling. Table 2 is the definition of types of coupling [18].

Table 2 Definition of Coupling [18]

Coupling	**Definition**
Content Coupling	The module modifies the internal operation of the other module or depends on its internal operation.
Common Coupling	Two modules share the same global data.
External Coupling	Two modules share the same external data.
Control Coupling	The module controls the flow of another module by passing information about what to do to other modules.
Stamp Coupling	The modules share a data through the parameter that is not the primitive data type.
Data Coupling	The modules share a data through the parameter that is the primitive data type.

According to the Table 2, the higher coupling is, the less suitable for modularization (Content Coupling) is. Conversely, the lower coupling is, the more suitable for modularization (Data Coupling) is.

3.3 Modularity

Modularity is a typically factor used to identify reusable modules, and is based on cohesion and coupling. The module with high cohesion and low coupling is suitable for modularization and reuse.

We use modularity to measure reusability. Fig. 2 is the cohesion and coupling at the method level. There are four methods(method1/2/3/4) and four statements(A,B,C,D) in them. We can measure modularity at the method level through the cohesion of internal statements and the coupling with external statements.

Table 3 shows the modularity measurement of single module and multiple modules in Fig. 1. It is difficult to say that method 1 is a reusable module because the number of coupling in the modularity is greater than the number of cohesion in the modularity. Assuming this module and method4 connected through the coupling

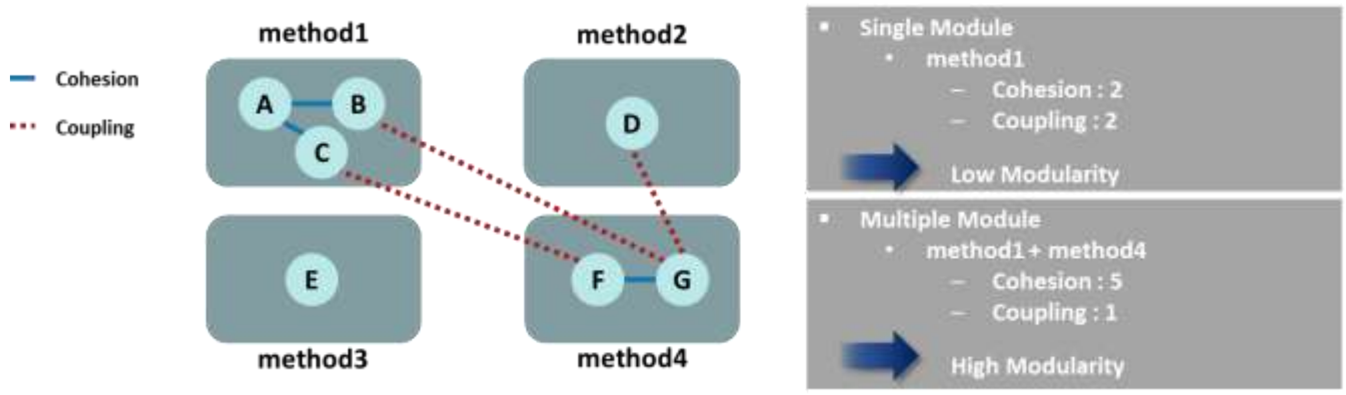


Fig. 2. Cohesion and Coupling of Method Level

as one module, it is a good case because the number of cohesion (5) in the modularity is greater than the number of coupling (2) in the modularity. Until now, we have seen approximate modularization by the number of cohesion and coupling. But in reality, there are many types of cohesion and coupling. Metrics is needed for measurement by quantification because of types of intensity.

Table 3 Modularity Measurement

Module	**Modularity**			
	Type	***Cohesion***	***Coupling***	***Reusable Module***
method1	Single	2	3	False
method2	Single	0	1	False
method3	Single	0	1	False
method4	Single	1	3	False
method1 + method3	Multiple	3	2	True
method1 + method4	Multiple	5	2	True
method2 + method4	Multiple	2	2	False

3.4 Modularity Metrics

To quantify the modularity, we define modularity grade for each type of cohesion and coupling. Table 4 and Table 5 show the modularity grade and weight of the cohesion and coupling, respectively. The grade increases sequentially from 1. Because the strong cohesion is good for modularization, the strongest functional cohesion has the highest grade. By contrast, since weakest coupling is good for modularization, the weakest data coupling has the highest grade. To help distinguish each type, we define weight. The weight of the highest grade is set to 1, and the other weights are set the respective weights divided by the number of types. The modularization metrics is defined at the method level based on these grades and weights. Since one

Table 4 Modularity Grade & Weight of Cohesion [18]

Cohesion	**ce**	**Grade**	**Weight**
Functional Cohesion	ce_f	7	1
Sequential Cohesion	ce_s	6	0.86
Communicational Cohesion	ce_m	5	0.71

Cohesion	**ce**	**Grade**	**Weight**
Procedural Cohesion	ce_p	4	0.57
Temporal Cohesion	ce_t	3	0.43
Logical Cohesion	ce_l	2	0.29
Coincidetal Cohesion	ce_c	1	0.14

Table 5 Modularity Grade & Weight of Coupling [18]

Coupling	**co**	**Grade**	**Weight**
Content Coupling	co_n	6	1
Common Coupling	co_m	5	0.87
External Coupling	co_e	4	0.67
Control Coupling	co_c	3	0.5
Stamp Coupling	co_s	2	0.33
Data Coupling	co_d	1	0.17

module can have various types of cohesion and coupling, we measure the cohesion and coupling at the method level by measuring the average of all cohesion and coupling. (1) is average of all cohesions grade, and (2) is average of all coupling grade.

$$\mathrm{ce} = \frac{1}{n}\sum_{i=0}^{n} ce_i = \frac{ce_f+ce_s+ce_m+ce_p+ce_t+ce_l+ce_c}{n} \qquad (1)$$

$$\mathrm{co} = \frac{1}{m}\sum_{i=0}^{m} co_i = \frac{co_d+co_s+co_c+co_e+co_m+co_n}{n} \qquad (2)$$

In addition, since the maximum grade of cohesion and coupling are not equal to 7 and 6, we normalize between 0 and 1 to match range of the cohesion and coupling grade. (3) and (4) are normalized expressions, and the value of denominator is not 0, assuming that ‘The maximum and minimum values of ce are not equal and The maximum and minimum values of co are not equal’. This allows to measure the cohesion and coupling of modularity at the method level.

$$ce_{nor} = \frac{ce-ce_{min}}{ce_{max}-ce_{min}}\ (ce_{max}-ce_{min} \neq 0) \qquad (□□$$

$$co_{nor} = \frac{co-co_{min}}{co_{max}-co_{min}}\ (co_{max}-co_{min} \neq 0) \qquad (□□$$

3.5 Reusability Metrics

The reusability metrics measures the average of the cohesion of all the internal methods of the object and the average of the coupling of all calls of the object in order to obtain the cohesion and coupling of the object. (5) is cohesion of object, and (6) is coupling of object. The object’s memory allocation size and frequency are also normalized between 0 and 1. (7) and (8) is the normalization of object allocation size and frequency respectively.

$$ce_{avg} = \frac{\sum_{i=0}^{n} ce_i}{n}\ (n: Method\ Count) \qquad (5)$$

$$size_{nor} = \frac{size-size_{min}}{size_{max}-size_{min}}\ (size_{max}-size_{min} \neq 0) \qquad (7)$$

$$co_{avg} = \frac{\sum_{i=0}^{m} co_i}{m}\ (m: Call\ Count) \qquad (6)$$

$$cnt_{nor} = \frac{cnt-cnt_{min}}{cnt_{max}-cnt_{min}}\ (cnt_{max}-cnt_{min} \neq 0) \qquad (8)$$

Reusable value R is measured on these values. (9) is the expression of Reusable value R. The larger the object allocation size has, the greater the probability of having unnecessary code and complexity has. Thus, since the smaller object allocation size increases the more reusability, it reduces the R value by multiplying the number by this value minus 1. The object allocation frequency shows the versatility as the object access frequency and the reusability in similar programs. Thus, since the highly object allocation frequency increases more reusability, it increases the R value by multiplying the number by this value plus 1. Since all values used to measure R are normalized between 0 and 1, it is assumed that they are suitable for reuse when the median of this number is greater than 0.5.

$$R = (\frac{1}{2}(ce_{avg} + co_{avg})) \times (1 - size_{nor}) \times (1 + cnt_{nor}) \quad (9)$$

$$(R > 0.5)$$

4. Reusability Strategy

In this session, we construct a reuse strategy to automatically identify and visualize reusable modules. This strategy performs static/dynamic analysis with open sources, and applies the reusability metrics to identify a reusable module and multiple modules. Fig. 3 shows the overall structure of the reusability strategy.

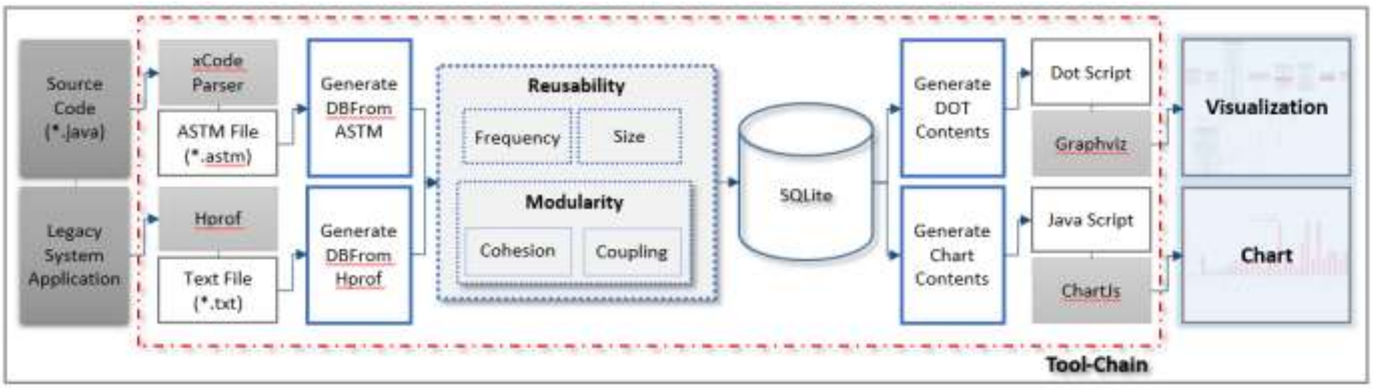


Fig. 3. Reusability Strategy

The overall process consists of five steps. We explain each step as follows.

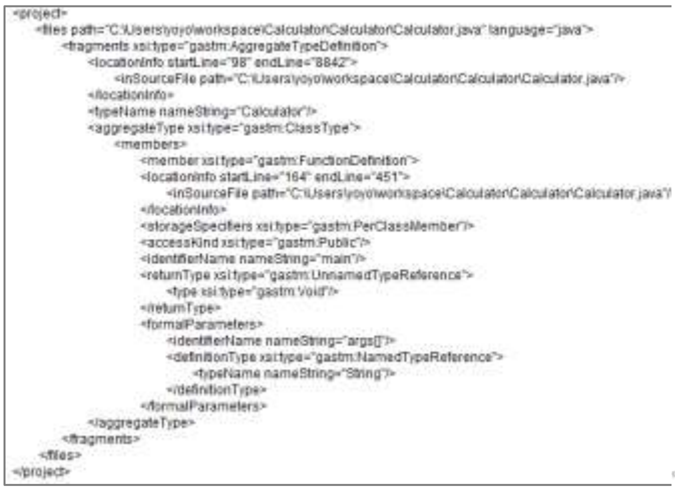
4.1 Input: Enter the target program

Because the static analysis and the dynamic analysis are all performed, we use source code and application. The source code is analyzed via xCodeParser, and the application is analyzed via HPROF[19].

4.2 Analysis: Extract the results through analysis

In static analysis, the analysis is performed with xCodeParser which are developed by our SE lab. of Hongik University. It extracts Abstract Syntax Tree Metamodel (ASTM). It is an integrated model of Abstract Syntax Tree (AST) so has the advantage of being able to convert to heterogeneous codes. Fig. 4 shows the ASTM as the result of the analysis, which represents XML-like text. Fig. 5 shows a tree of this structure [20,21].

In dynamic analysis, the analysis is performed with HPROF. It is actually a JVM native agent library which is dynamically loaded through a command line option at JVM startup, and becomes part of the JVM process. By supplying HPROF options at startup, users can request various types of heap and/or CPU profiling features from HPROF. The generated data can be in textual or binary format, and be used to track down, which isolate performance problems involving memory usage and inefficient code. Fig. 6 shows a screen that actually executes the analysis in the command line. Fig. 7 is the resulting file as textual format [19].



```
<project>
    <files path="C:\Users\yoyo\workspace\Calculator\Calculator\Calculator.java" language="java">
        <fragments xsi:type="gastm:AggregateTypeDefinition">
            <locationInfo startLine="98" endLine="8842">
                <inSourceFile path="C:\Users\yoyo\workspace\Calculator\Calculator\Calculator.java"/>
            </locationInfo>
            <typeName nameString="Calculator"/>
            <aggregateType xsi:type="gastm:ClassType">
                <members>
                    <member xsi:type="gastm:FunctionDefinition">
                    <locationInfo startLine="164" endLine="451">
                        <inSourceFile path="C:\Users\yoyo\workspace\Calculator\Calculator\Calculator.java"/
                    </locationInfo>
                    <storageSpecifiers xsi:type="gastm:PerClassMember"/>
                    <accessKind xsi:type="gastm:Public"/>
                    <identifierName nameString="main"/>
                    <returnType xsi:type="gastm:UnnamedTypeReference">
                        <type xsi:type="gastm:Void"/>
                    </returnType>
                    <formalParameters>
                        <identifierName nameString="args[]"/>
                        <definitionType xsi:type="gastm:NamedTypeReference">
                            <typeName nameString="String"/>
                        </definitionType>
                    </formalParameters>
            </aggregateType>
        </fragments>
    </files>
</project>
```

Fig. 4. ASTM

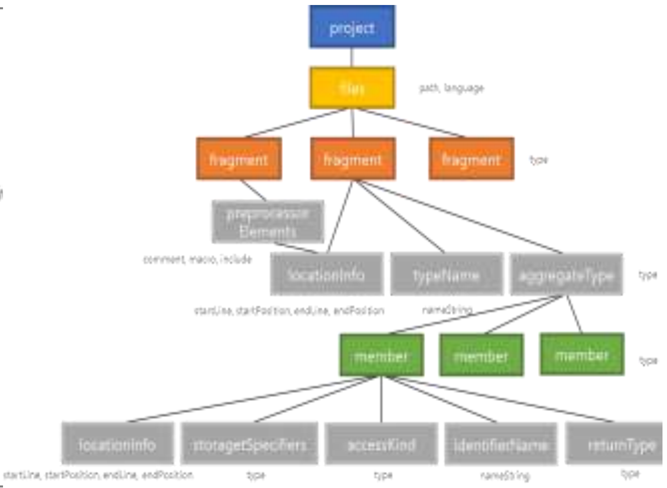
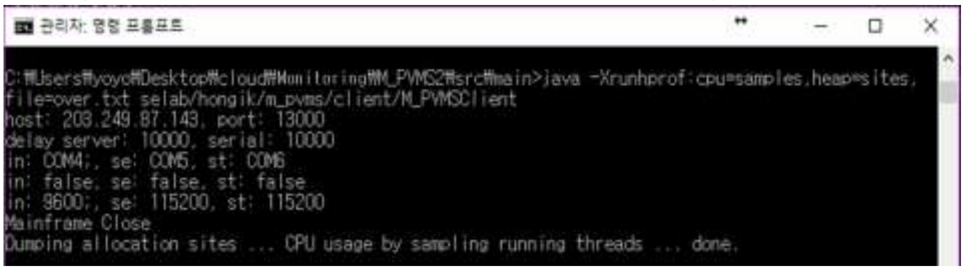
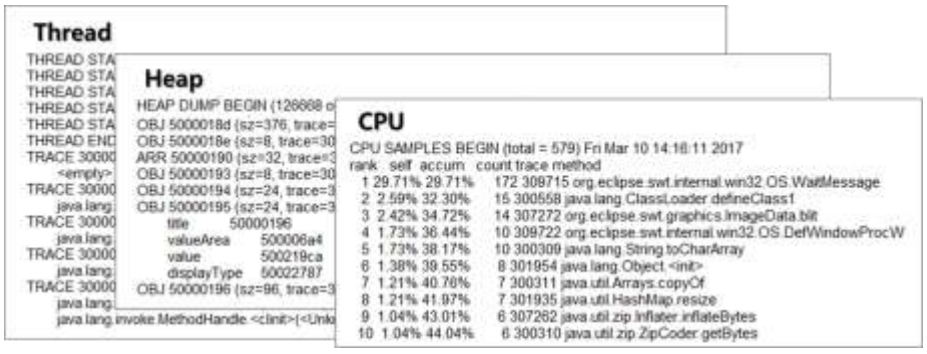


Fig. 5. ASTM Tree



```
관리자: 명령 프롬프트
C:₩Users₩yoyo₩Desktop₩cloud₩Monitoring₩M_PVMS2₩src₩main>java -Xrunhprof:cpu=samples,heap=sites,
file=over.txt selab/hongik/m_pvms/client/M_PVMSClient
host: 203.249.87.143, port: 13000
delay server: 10000, serial: 10000
in: COM4:, se: COM5, st: COM6
in: false, se: false, st: false
in: 9600:, se: 115200, st: 115200
Mainframe Close
Dumping allocation sites ... CPU usage by sampling running threads ... done.
```

Fig. 6. Command line executing HPROF



```
Thread
THREAD STA
THREAD STA
THREAD STA
THREAD STA
THREAD STA
THREAD END
TRACE 30000
    <empty>
TRACE 30000
    java.lang.
TRACE 30000
    java.lang.
TRACE 30000
    java.lang.
TRACE 30000
    java.lang.
    java.lang.invoke.MethodHandle.<clinit>(<Unk

Heap
HEAP DUMP BEGIN (126868 o
OBJ 5000018d (sz=376, trace=
OBJ 5000018e (sz=8, trace=30
ARR 50000190 (sz=32, trace=3
OBJ 50000193 (sz=8, trace=30
OBJ 50000194 (sz=24, trace=3
OBJ 50000195 (sz=24, trace=3
    title          50000196
    valueArea      500006a4
    value          500219ca
    displayType    50022787
OBJ 50000196 (sz=96, trace=3

CPU
CPU SAMPLES BEGIN (total = 579) Fri Mar 10 14:16:11 2017
rank   self  accum   count trace method
   1 29.71% 29.71%     172 309715 org.eclipse.swt.internal.win32.OS.WaitMessage
   2  2.59% 32.30%      15 300558 java.lang.ClassLoader.defineClass1
   3  2.42% 34.72%      14 307272 org.eclipse.swt.graphics.ImageData.blit
   4  1.73% 36.44%      10 309722 org.eclipse.swt.internal.win32.OS.DefWindowProcW
   5  1.73% 38.17%      10 300309 java.lang.String.toCharArray
   6  1.38% 39.55%       8 301954 java.lang.Object.<init>
   7  1.21% 40.76%       7 300311 java.util.Arrays.copyOf
   8  1.21% 41.97%       7 301935 java.util.HashMap.resize
   9  1.04% 43.01%       6 307262 java.util.zip.Inflater.inflateBytes
  10  1.04% 44.04%       6 300310 java.util.zip.ZipCoder.getBytes
```

Fig. 7. Resulting files of HPROF analysis

4.3 Storing

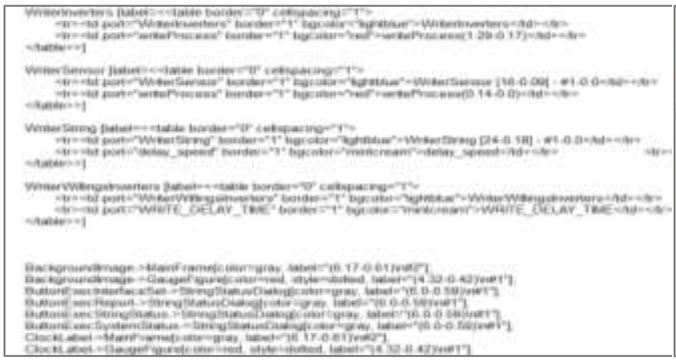
In this session, the reusability metrics defined above is applied on the ASTM as a result of static analysis and the text file as a result of dynamic analysis. Then, this result is stored in a database (SQLite). The tables in the database are 'ASTM_contents', 'ASTM_link', 'ASTM_API', 'Cohesion', 'Coupling', 'Profiling'. 'ASTM_contents' stores a property of variable and method. 'ASTM_link' stores a property of call and object creation. 'ASTM_API' stores a property of API used in the program. All three tables store the overall structure information of the system. Cohesion and Coupling store the grade of cohesion and coupling which is applied the modularization metrics to a program, respectively. Profiling stores the grade of the object allocation size and frequency through dynamic analysis.

4.4 Reconstructing

The visualization tools used in this paper Graphviz which is visualized the result of static analysis, and Chart.js which is visualized the result of dynamic analysis. Both tools have corresponding grammars, so we need to reconstruct the data in the database according to this grammar.

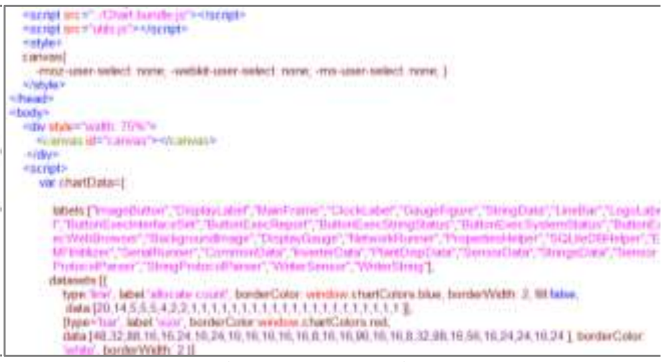
Graphviz is open source based graph software initiated by AT&T Labs Research for drawing graphs specified in DOT language scripts. Graph software is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, data base and web design, machine learning, and in visual interfaces for other technical domains. It can visualize nodes in various styles from normal nodes to complex nodes such as table-shaped nodes. It is also easy to set options like line styles, hyperlinks [23,24]. It visualizes graphs in image of Scalable Vector Graphics (SVG) format through DOT Script. We have to reconstruct the DOT Script according to this definition. Fig. 8 is the actual DOT Script structure.

Chart.js is open source with charting of numerical data using Java Script which provides eight types of charts. It has advantage of being able to run in any browser and allowing the user to define and use the chart type directly. Likewise, we have to reconstruct the Java Script [25]. Fig. 9 is the actual Java Script structure.



```
WriterInverters [label=<<table border="0" cellspacing="1">
        <tr><td port="WriterInverters" border="1" bgcolor="lightblue">WriterInverters</td></tr>
        <tr><td port="writeProcess" border="1" bgcolor="red">writeProcess(1.29-0.17)</td></tr>
</table>>]

WriterSensor [label=<<table border="0" cellspacing="1">
        <tr><td port="WriterSensor" border="1" bgcolor="lightblue">WriterSensor [16-0.09] - #1-0.0</td></tr>
        <tr><td port="writeProcess" border="1" bgcolor="red">writeProcess(0.14-0.0)</td></tr>
</table>>]

WriterString [label=<<table border="0" cellspacing="1">
        <tr><td port="WriterString" border="1" bgcolor="lightblue">WriterString [24-0.18] - #1-0.0</td></tr>
        <tr><td port="delay_speed" border="1" bgcolor="mintcream">delay_speed</td></tr>                <tr><
</table>>]

WriterWillingsInverters [label=<<table border="0" cellspacing="1">
        <tr><td port="WriterWillingsInverters" border="1" bgcolor="lightblue">WriterWillingsInverters</td></tr>
        <tr><td port="WRITE_DELAY_TIME" border="1" bgcolor="mintcream">WRITE_DELAY_TIME</td></tr>
</table>>]


BackgroundImage->MainFrame[color=gray, label="(6.17-0.61)\n#2"]
BackgroundImage->GaugeFigure[color=red, style=dotted, label="(4.32-0.42)\n#1"]
ButtonExecInterfaceSet->StringStatusDialog[color=gray, label="(6.0-0.59)\n#1"]
ButtonExecReport->StringStatusDialog[color=gray, label="(6.0-0.58)\n#1"]
ButtonExecStringStatus->StringStatusDialog[color=gray, label="(6.0-0.58)\n#1"]
ButtonExecSystemStatus->StringStatusDialog[color=gray, label="(6.0-0.59)\n#1"]
ClockLabel->MainFrame[color=gray, label="(6.17-0.61)\n#2"]
ClockLabel->GaugeFigure[color=red, style=dotted, label="(4.32-0.42)\n#1"]
```

Fig. 8. DOT Script



```
  <script src="./Chart.bundle.js"></script>
  <script src="utils.js"></script>
  <style>
  canvas{
    -moz-user-select: none; -webkit-user-select: none; -ms-user-select: none; }
  </style>
</head>
<body>
  <div style="width: 75%">
    <canvas id="canvas"></canvas>
  </div>
  <script>
    var chartData={

      labels:["ImageButton","DisplayLabel","MainFrame","ClockLabel","GaugeFigure","StringData","LineBar","LogoLab
      l","ButtonExecInterfaceSet","ButtonExecReport","ButtonExecStringStatus","ButtonExecSystemStatus","Button[illegible]
      e:WebBrowser","BackgroundImage","DisplayGauge","NetworkRunner","PropertiesHelper","SQLiteDBHelper","[illegible]
      MFInitilizer","SerialRunner","CommonData","InverterData","PlantDispData","SensorData","StringData","Sensor
      ProtocolParser","StringProtocolParser","WriterSensor","WriterString"],
     datasets:[{
       type:'line', label:'allocate count', borderColor: window.chartColors.blue, borderWidth: 2, fill:false,
        data:[20,14,5,5,5,4,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]},
       {type:'bar', label:'size', borderColor: window.chartColors.red,
       data:[48,32,88,16,16,24,16,24,16,16,16,16,16,8,16,16,96,16,16,8,32,88,16,56,16,24,24,16,24 ], borderColor:
       'white', borderWidth: 2 }]
```

Fig. 9. Java Script

4.5 Visualization : Finally, in the visualization step, DOT script and Java Script are visualized using the Graphviz and Chart.js.

5. Visualization of our Target Program

We try to apply the reuse strategy to the actual program which is a solar and geothermal power generation monitoring system for checking current generation in real time. Fig. 10 show the system structure and the actual execution screen of the monitoring program. This data communication is performed by integrating heterogeneous data protocol from local to server based on metamodeling [26].

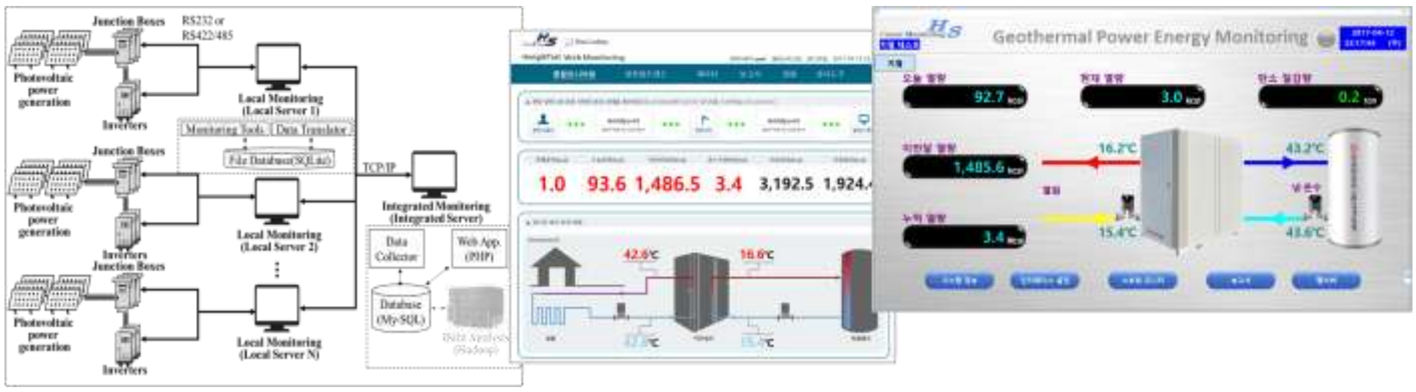

Junction Boxes
RS232 or RS422/485
Photovoltaic power generation
Inverters
Local Monitoring (Local Server 1)
Monitoring Tools
Data Translator
File Database(SQLite)
TCP/IP
Integrated Monitoring (Integrated Server)
Local Monitoring (Local Server 2)
Data Collector
Web App (PHP)
Database (My-SQL)
Local Monitoring (Local Server N)
Geothermal Power Energy Monitoring

Fig. 10. The target program of an integrated Solar & Geothermal power energy monitoring system

We define the types of nodes and edges before visualization. Fig. 11 shows types of nodes and edges. Packages (Green) and Classes (Blue) inside the package are represented by subgraphs, and distinguished by the color. When classes are represented as a nodes, it is displayed in the form of a class diagram using a table type to represent an internal structure. This node has information on variables and methods declared in the class. The method is represented by an oval node. It is shown in sky blue with a cohesion grade of 0.5 or higher. With a cohesion grade of less than 0.5 it is likely to be unsuitable for reuse, and display this as a red triple oval to increase visibility. Within each method, the cohesion grade is displayed. The used API is represented by subgraph. Relationships between components such as classes, methods, and APIs are inheritance, each call, and object creation which can be distinguished by the color of the edges. When a component's coupling grade is 0.5 or more, it is indicated by a solid blue line. While it is less than 0.5, it is represented by a red dotted line.

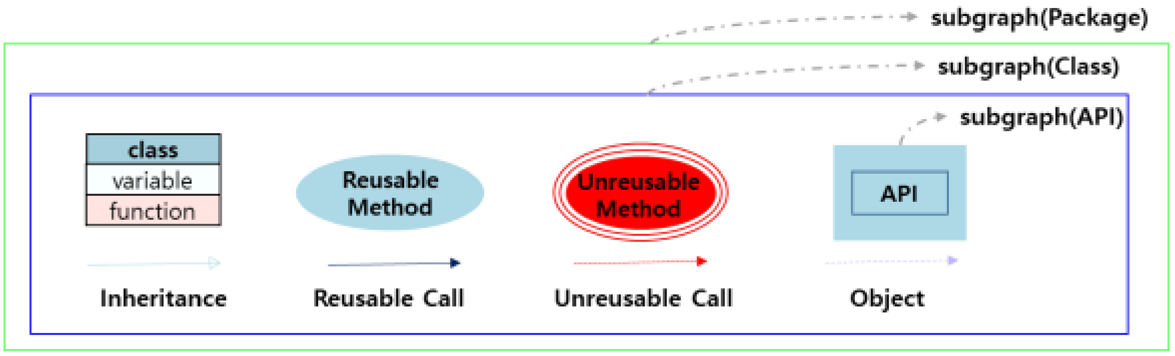


Fig. 11. Types of Visualization of Node & Edge

Fig. 12 is a visualization chart using Chart.js. It visualizes the object allocation size and frequency in memory as a line graph and a bar graph, respectively, in a mixed graph. The 'ImageButton' object should be considered for reusability because the object allocation size is the average number, and the object allocation frequency is the highest number. Since the 'GaugeFigure' object and the 'StringData' object have the small object allocation size and a large object allocation frequency, then we should be considered for reusability. Thus, we can roughly identify objects that be considered of reusability through chart visualization.

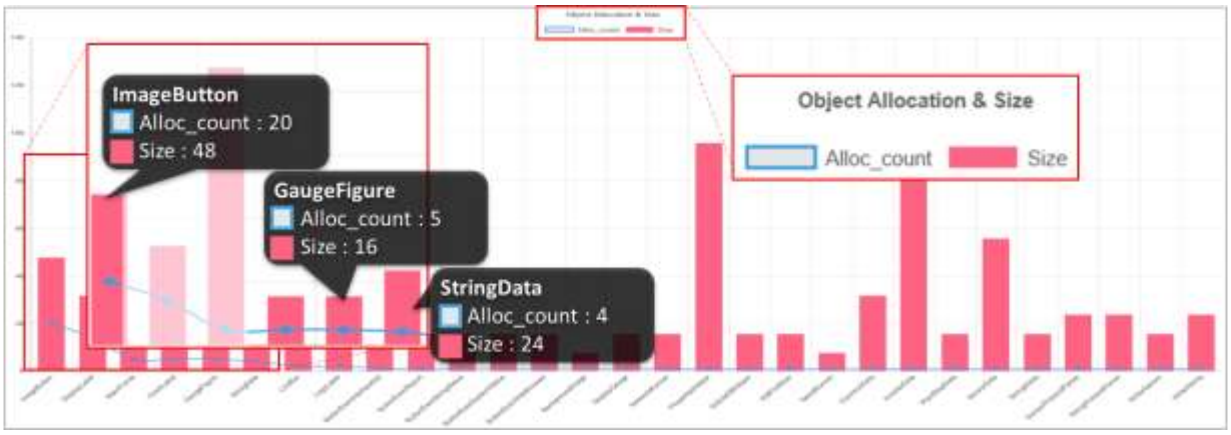


Fig. 12. Object Allocation Size & Frequency Chart

Fig. 13 shows method level visualization using Graphviz. Typically, the 'update' and the 'getValue' method are identified as a good case by modularization metrics. Method 'update' is called four times. Method 'getValue' is called once. Therefore, the method 'getValue' may be more reusable than the method 'update'

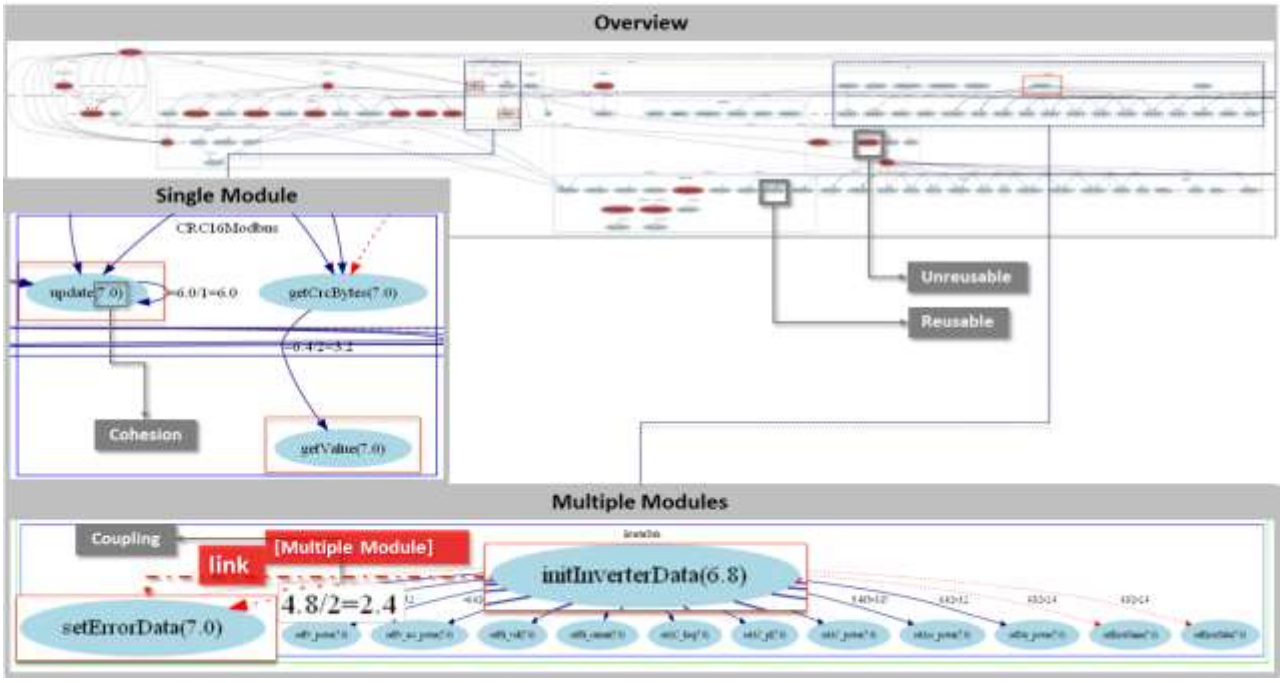


Fig. 13. Method Level Visualization

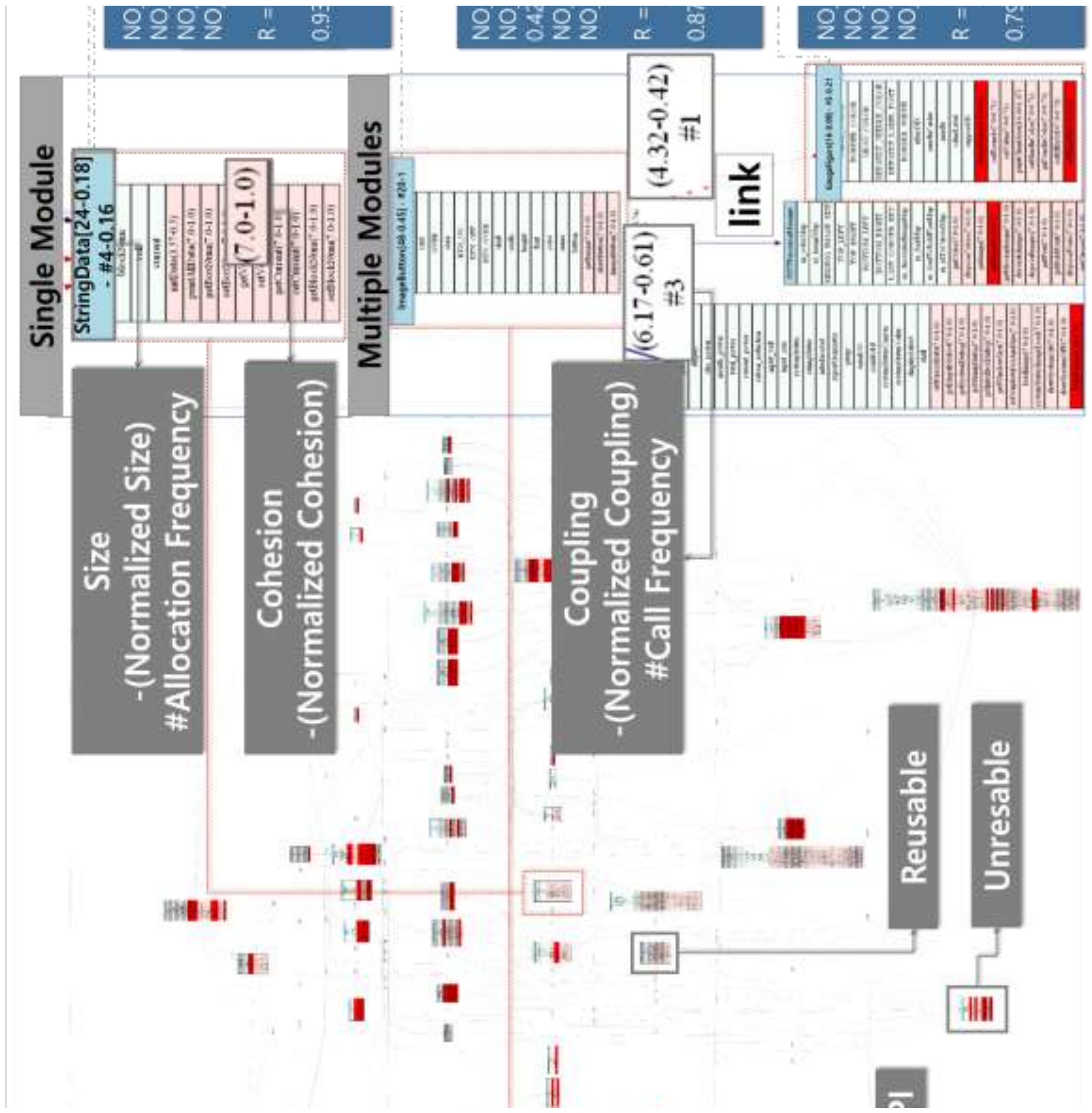

Single Module
StringData[24-0.18] - #4-0.16
(7.0-1.0)
Multiple Modules
(4.32-0.42) #1
link
(6.17-0.61) #3
R = 0.93
R = 0.87
R = 0.79
Size -(Normalized Size) #Allocation Frequency
Cohesion -(Normalized Cohesion)
Coupling -(Normalized Coupling) #Call Frequency
Reusable
Unresable

Fig. 14. Object Level Visualization

with more call frequency. The multiple modules are a chunk that is connected with a good single module and the other modules which have high coupled with its modules. The method 'initInverterData' is a good single module, but has high coupling with the method 'setErrorData'. Therefore, it is effective to use the method 'initInverterData' as multiple modules by connecting with the method 'setErrorData'.

Fig. 14 shows object level visualization using Graphviz. A calling relationship from the methods in the class (with lower than o.5 of the cohesion grade) to classes (with lower than 0.5 of the coupling grade) is displayed with a red dotted arrow to increase visibility. A node is to be displayed as a class with the object name, allocation size, and frequency at the top line. The typical single module 'StringData' class is identified as a reusable module. The reusable value R using the reusability metrics is 0.93, which is higher than 0.5 that is set before. In the multiple modules, the 'ImmageButton' object is a good case for reuse by the metrics. But it has a high coupling with the 'GaugeFigure' object. At this time, if the 'GaugeFigure' object is also a good case for reuse, it is effective to reuse the chunk of two modules as multiple modules.

6. Conclusion & Future work

In this paper, we perform the static & dynamic analysis to measure the reusability. And we develop the xCodeParser for static analysis, and use the HPROF for dynamic analysis. We use cohesion, coupling, object allocation size, and frequency to define reusability metric on the static & dynamic analysis. In the previous works, CK metrics was used to measure the cohesion and coupling, but in practice, not possible to measure the value of cohesion and coupling with different types. In order to solve this problem, we define the modularity metric based on each definition of cohesion and coupling. We then apply the object allocation size and frequency, which is the result of the dynamic analysis, on this metric. And we construct the reusability strategy through this metric. We construct the visualization tools with this strategy to identify the objects that need to consider reusability. Thus, this strategy allows automatic identification of module and chunk of modules which is suitable for reuse at method and class level. This makes it possible to improve reusability. On developing a new project with any reusable codes within the legacy software, we expect to reduce time and cost of development, and to enhance productivity of software.

In the future work, we will add new quality factors based on static & dynamic analysis on object oriented approach. And we will conduct in-depth researches on dynamic analysis in particular. These will improve the reusability metrics.

Acknowledgments

This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea (NRF-2015H1C1A1035548), and also by a grant (17CTAP-C133299-01) from Technology Advancement Research Program funded by Ministry of Land, Infrastructure and Transport of Korean government.

References

[1] Science and Technology Policy Institue, “Characteristic analysis on innovation by software application sector,” 2015.
[2] Richard N. Taylor “Software Development Using Domain-Specific Software Architectures’ ACM” Vol. 20, No. 5, pp. 27-38, December, 1995.
[3] Stephen H. Kan, “Metrics and models in software Quality Engineering,“ In Addison-Wesley Professional, 2002.
[4] W Frakes, C. Terry, “Software reuse: metrics and models,“ In ACM Computing Surveys, Vol. 28, No. 2, pp. 415-435, June, 1996.
[5] Marko Mijac and Zlatko Stapic, “Reusability Metrics of Software Components: Survey” in Central European Conference on Information and Intelligent, Varazdin, Croatia, pp. 221-230, September, 2015.
[6] Bojana Koteska and Goran Velinov, “Component-Based Development: A Unified model of Reusability Metrics” in ICT Innovations 2012, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 335-344.
[7] Vijai Kumar, Arun Sharma, Rajesh Kumar, and P.S. Grover, “Quality aspects for component based systems: A metrics based approach” Sofw.Pract.Exp.,42(12), pp. 1531-1548, Dec. 2012.
[8] V. Lee, “Automated source code measurement environment for software quality” Doctoral, Auburn University, Auburn, USA, 2007.
[9] A. Sharma, R. Kumar, and P. S. Grover, “Managing component based systems with reusable components”

Int. J. Comput. Sci. Secure:, pp. 52-57, 2007.

[10] P.K. Suri and Neeraj Garg, "Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse" Int. J. Comput.Sci.Netw.Secur., vol. 9, no. 5, pp. 237-248, 2009.

[11] A Sharam, Rajesh Kumar, and p. s. Grover, "Critical Survey of Reusability Aspects for Component Based Systems" presented at the Enformatika, Vol. 1, no. 9, pp. 420-424, 2007.

[12] G. Shanmugasundaram, V. Prasanna Venkatesan, and C. Punitha Devi, "Research opportunities in service reusability of service oriented architecture" in 2012 International Conference on Emerging Trends in Science, Engineering and Technology, Tamilnadu, India, 2012, pp. 396-403.

[13] Ramanath Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", in IEEE Transactions on Software Engineering, Vol. 29, No.4, April, 2003.

[14] P. Sandhu and H. Singh, "Automatic Reusability Appraisal of Software Components using Neurofuzzy Approach," Int. J. Inf. Technol., vol. 1, no. 8, pp. 2407–2413, 2007.

[15] Gui gui, Paul D. Scott, "Measuring Software Component Reusability by Coupling and Cohesion Metrics" in Journla of Computers, vol. 4, No. 9, pp. 576–578, September 2009.

[16] J. Al Dallal, "Software similarity-based functional cohesion metric" in Institutioin of Engineering and Technololgy, vol. 3, pp. , 46-57, Feburary, 2008.

[17] L. H. Etzkorn, W. E. Hughes, and C. G. Davis, "Automated reusability quality analysis of OO legacy software" Inf. Softw. Technol., vol. 43, no. 5, pp. 295–308, 2001.

[18] Eun Young Byun, Bo Kyung Park, Woo Sung Jang, R. Young Chul Kim, Hyun Seung Son, "Constructing an Open Source Based Software System for Reusable Module Extraction," Korea Instutitue of Information Scientists and Engineers, vol. 23, no. 9, pp. 535-541, Septempber, 2017.

[19] HPROF: A Heap/CPU Profiling Tool, http://docs.oracle.com/javase/7/docs/technotes/samples/HPROF.html.

[20] Eun-Young Byun, Hyun-Seoung Son, So-Young Moon, Woo-Sung Jang, Bo-Kyung Park, R. Youngchul Kim, "Constructing A Visualization & Reusable Metrics based on Static/Dynamic Analysis," Korea Information Processing Society, vol. 24, pp. 621-624, April 2017.

[21] Hyun Seung Son, So young Moon, R. Young Chul Kim, "Replacing Source Navigator with Abstract Syntax Tree Metamodel(ASTM) on the open source oriented tool chains SW Visualization," In International Conference on Convergence Technology, Vol. 5, No. 1, pp. 366-367, 2015.

[22] Hyun Seung Son, R. Young Chul Kim, "xCodeParser based on Abstract Syntax Tree Metamodel(ASTM) for SW Visulization," In INFORMATION : an international interdisciplinary journal, Vol. 20, NO. 2(A), pp. 963-968, Feburary 2017.

[23] J Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, Gordon Woodhull, "Graphviz and Dynagraph-Static and Dynamic Graph Drawing Tools," 2004.

[24] J Ellson, "Graphviz-Open Source Graph Drawing Tools," In Lecture Notes in Computer Science, Vol. 2265, pp. 483-484, February, 2001.

[25] Chart.js: API Documentation, http://www.chartjs.org/docs.

[26] Hyun Seung Son, R. Young Chul Kim "Modeling a Photovoltaic Monitoring System based on Maintenance Perspective for New & Renewable Energy' International Joint Conference on Convergence"

[27] pp. 144-147, January.