The Experience of Software Reengineering by Using Haptics through Tangible User Interface

Makoto Yoshida, Tomoya Okazaki
Department of Information & Computer Engineering
Okayama University of Science
Okayama, Japan
Email: yoshida@ice.ous.ac.jp

Noriyuki Iwane
Department of Information & Computer Science
Hiroshima City University
Hiroshima, Japan
Email: iwane@hiroshima-cu.ac.jp

***Abstract*—The need for software reengineering is ever increasing. To satisfy the need, several metrics and tools are developed. We developed the toolkit which reorganizes the software programs using the haptics through tangible interface. The toolkit decomposes the Java source programs into small classes, and integrates them into the harmonized classes by using the haptic device. The metrics analyzed are mapped to the attributes of the virtual objects, and can be touched and perceived by the haptics through tangible interface and integrated into the harmonized object by coupling objects. This paper describes the way that software programs are reengineered by the toolkit. Software reengineering methodology using the toolkit is proposed. Sound coupling and cohesion coupling by using haptics through tangible interface are introduced, and some experiments performed are presented.**

Keywords-Software Reengineering; Toolkit; Metrics; Haptics; Tangible User Interface.

I. INTRODUCTION

Most of the programs are not newly written; they are reused, and the systems are maintained. The objective of reengineering is to produce a new maintainable system with least efforts [8]. Many tools are developed [5]. Metrics analysis and visualization help to reorganize programs [1][4]. The reorganized program must have adequate modularity; modules with high cohesion and low coupling must be maintained [2][3]. M. Lanza, etc. express the metrics of the program by 3D visualization and by the metaphor of cities [6][7].

We developed the software reengineering toolkit with tangible user interface by using the haptic device [9]. The program modules are visualized as 3D objects like spheres and cubes; each objects having its tangible attributes, mapped from the program metrics. The program module structure is reorganized by decoupling and coupling modules by using the tangible user interface. It supports two types of interfaces, the active interface and the passive interface. In this paper, we focus on the active interface. Especially, the sound coupling interface and cohesion coupling interface are presented. The toolkit was developed not only for software reengineering but also for learning programming.

In this paper we make the following contributions.

- We propose the metrics schema that integrates objects into the harmonized programs and observe its effectiveness.
- We present the tangible user interface that is easy to manipulate, easy to evaluate and easy to undo the coupling operations, adding the cohesion coupling to the toolkit [9].
- We present some experimental results and observe the effectiveness of the toolkit for software reengineering.

The rest of the paper is organized as follows. Section II presents the reengineering toolkit. Section III describes the experiments of the sound and cohesion couplings. Finally, Section IV concludes the paper with future works.

II. REENGINEERING SOFTWARE

Disharmonized programs, those are the programs that may have intensive coupling, shotgun surgery, dispersed coupling, god class, etc. [1], must be recognized and reorganized on the point of software maintenance.

A. Toolkit

This section summarizes the toolkit we developed. More detail of the toolkit is described in the reference [9].

The motivation of the toolkit developed can be summarized as follows.

- Flexible and simple tool for software reengineering is required, as the direct metrics manipulation is too complex
- The operation of several module couplings and decouplings must be performed easily , understood easily, and undone easily
- Everybody, including software non-professionals, can manipulate the module couplings and decouplings easily

Figure 1 shows the system structure of the toolkit. It consists of three parts: the program analysis part, the object perception part, and the code generation part. The program analysis part analyzes the Java source programs, and produces the metrics of the programs. Numbers of classes, lines, methods, fields, dependency of classes, etc. are analyzed. It can also decompose a class into smaller classes. A class can be decomposed into more small classes.

Decomposed classes are represented as the objects in the object perception part. The metrics of a class are mapped to the attributes of the corresponding object. The haptic device, shown in Figure 2, the physical device which provides the tangible user interface to the toolkit, is used to manipulate the object couplings. The toolkit provides several coupling methods [9]. The sound coupling, color coupling, undo coupling and cohesion coupling are provided. This paper introduces the cohesion coupling. The other coupling are described in the previous paper [9]. In the code generation part, the Java source code is automatically generated from the results of object integration manipulated either in the object perception part or in the analysis part.

Figure 1 also shows the way that the program is reengineered using the tangible user interface. There exists three cycles in the Figure 1. The first cycle is depicted by the arrows ① and ②. This cycle directly uses metrics to reorganize the modules. In this cycle, the metrics such as overviewPyramid, complexity, hotspots and blueprint [1] are visualized, and the program is reorganized. The second cycle is depicted by the arrows ③,⑤, ⑦ and ②. The third cycle is depicted by the arrows ④, ⑥ , ⑦ and ② .The analyzed metrics are passed to the object perception part, and those data are mapped to the attributes of the haptic objects.

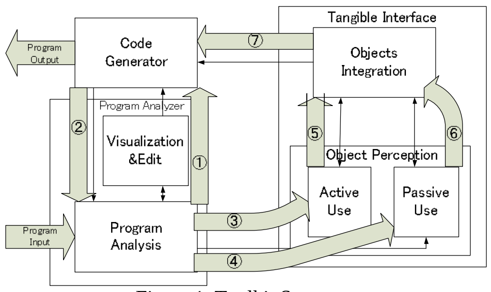


Figure 1. Toolkit Structure

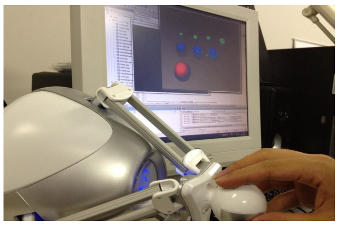


Figure 2. Haptic Device

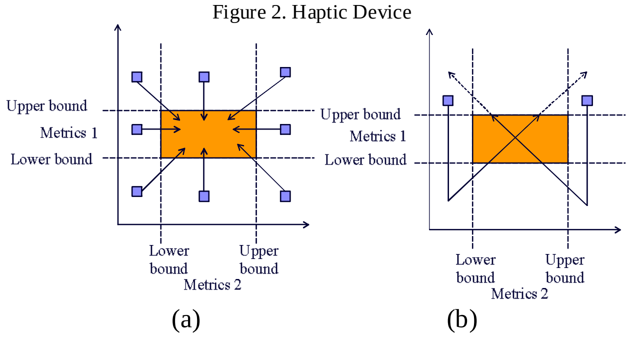


Figure 3. Object Integration Schema

B. Metrics Schema

The program must be reorganized to have the metrics to be the appropriate value. The value of the metrics can be normalized by decomposing and integrating the program modules step by step.

Figure 3 shows the metrics schema for software reengineering using the toolkit we developed. Two metrics, metrics 1 and metrics 2, and their appropriate domains, which is under the shadow area within the rectangular, are shown in Figure 3. Figure 3(a) shows the way that both metrics, metrics 1 and 2, are converging into the appropriate values. Figure 3(b) shows that the metrics 1, which is out of the appropriate domain and have higher value, is first decreased by decomposing a module. And, second, the decreased value is increasingly changed to the appropriate value by coupling modules. This is performed by adjusting the metrics 2 which must be also situated within the shadow area.

Assuming that, the number of methods in a class is assigned to the metrics 1, and the lines of code in a method is assigned to the metrics 2. In this case, the schema adopted for reorganizing the modules works as follows:

Operation1. Assign the metrics M1 to the modules of a program. Decompose the program. The toolkit we developed can analyze and decompose a class into smaller units of module, and create the corresponding objects. This operation always decreases the value of M1, and it is repeated until the M1ôs value becomes lower than the threshold of the lower bound predefined.

Operation 2. Integrate the objects. Use the metrics M2 to integrate the objects. The coupling operations provided are performed to lead the M1 and M2ôs value to converge into the shadowed rectangular area.

C. Coupling & Decoupling in Java

It is possible to decompose the program into smaller elements, to the subclasses or submodules in Java. One class can be decomposed into several subclasses, and integrated into one large class with the combination of different modules. Figure 4 shows the decomposition of the Java program. It shows that class 0 can be decomposed into two classes: class 1 and class 2, because of the reason that the method A and B only access the field a, and method c only accesses the field b and c. Figure 4 also shows the integration possibility of two classes, class 1 and class 2, into a class 0. The basic policy for program decomposition is the fact that the program can be decomposed into the smaller unit if there exists no dependency among units, namely if no units interaction occurs. Figure 5 shows an example of the program coupling and decoupling in Java. These are performed by using the toolkit we developed [9]. In Figure 5, Program AB is decomposed into program Aa and Bb, and Program CD is decomposed into program Cc and Dd. Then, class Aa and Cc are coupled into one program AC, and Bb and Dd are coupled into another program BD.

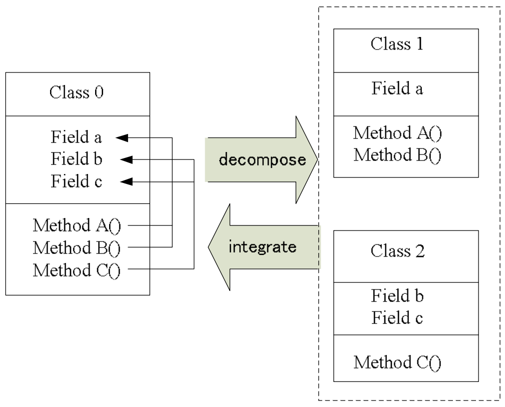
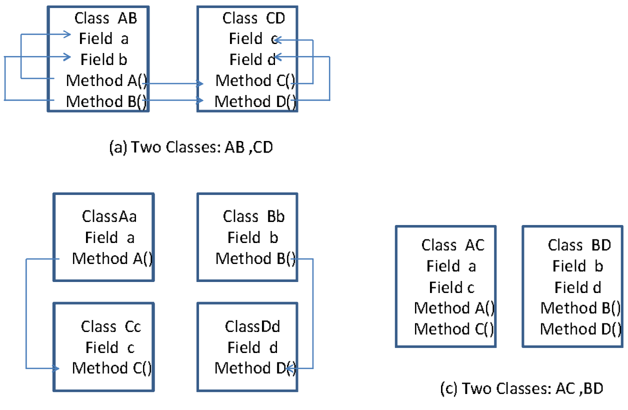


Figure 4. Class Decomposition & Integration



(a) Two Classes: AB ,CD

(b) Four Classes: Aa, Bb, Cc, Dd

(c) Two Classes: AC ,BD

Figure 5. Program Integration Example

III. EXPERIENCES

This section describes the coupling methods of the objects. The sound coupling and the cohesion coupling in the toolkit are introduced, and some experiments using these couplings are examined.

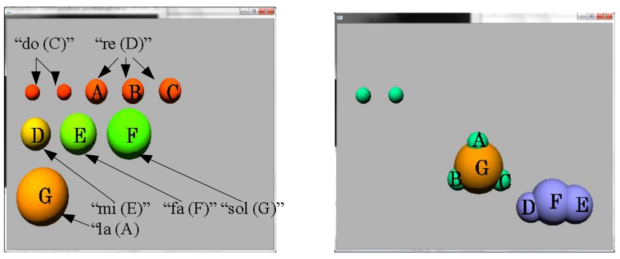
A. Metrics

The attributes of the object, visualized and touched by the haptic device, are gravity, magnetism, spring, color, transparency, etc. The metrics analyzed in program analysis part are mapped to those of the attributes of the object.

Table I shows the metrics mapped to the object in the experiment. Two metrics, metrics 1 and metrics 2, are used in this experiment. The metrics 1, M1, is the average number of lines per method in a class, and the metrics 2, M2, is the number of methods per class. The metrics M1 is mapped to the size of an object. In the sound coupling, M2 is mapped to the sound of the object. In the cohesion coupling, M2 is mapped to the distance among the objects. According to the metrics schema described, the metrics 2 is used to decompose the modules, and metrics 1 is used to integrate the modules.

TABLE I. METRICS MAPPING

Metrics	Sound Coupling	Cohesion Coupling
Number of classes	Number of objects	Number of objects
(M1) Average lines/method/class	Size of the object	Size of the object
(M2) Number of methods/class	Sound of the object	Distance among objects



(a) Sound Objects (b) Sound Coupling

Figure 6. Sound Coupling

B. Sound Coupling and Cohesion Coupling

In the sound coupling, the metrics 2 (M2) is mapped to the sound attribute of an object. Figure 6 shows an example of the sound objects, Objects A, B and C have the sound “re”, D has “mi”, E has “fa”, F has “so”, and G has sound “la”. Touching and moving the objects, object A, B, C and G are merged into one object, as is shown in Figure 6(b), and the sound of the object changes to “mi”, that is assigned as the average height of the objects merged.

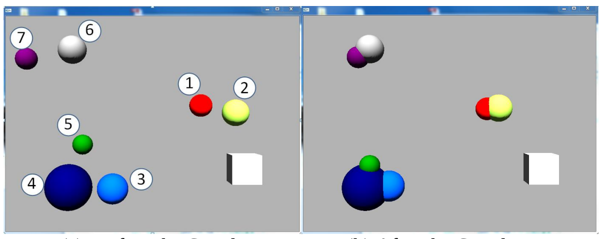
Cohesion refers to the degree of tightness to which module components belong together [10][11]. Cohesion coupling visualizes the class tightness by the distance of classes. The class tightness is the dependency of the objects. The number of run time accesses among objects, which represents the class tightness, is measured and logged by using ASPECT-J. And, the distance among the objects is calculated by using the following formula (1) [10]:

$$dis(x,y)= 1-|b(x)\cap b(y)|/|b(x)\cup b(y)| \qquad (1)$$

with $b(x):=\{Pi\in B|$ x possesses Pi , Pi is a method$\}$
and {b(x) is the set of methods accessed by an object x.}

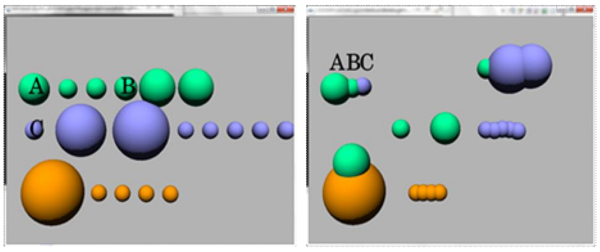
The similarity of the two objects x, y with respect to a property subset B is calculated from the similarity measure, that is represented by $|b(x)\cap b(y)|/|b(x)\cup b(y)|$. The distance calculated by formula (1) is visualized by the multidimensional scaling method (MDS), which uses the Young-Householder translation [12]. The detail of the MDS is shown in the Appendix. Using MDS, the object are situated into the similarity-based distance, where the tighter objects be located closer. Figure 7 shows an example. The Figure 7(a) shows that object 1 and 2 have the tighter relation, and so is the 3, 4 and 5, and 6 and 7. Figure 7(b) shows the cohesion coupling. Object 1 and 2 are coupled into one module, and the object 3, 4 and 5 are merged into one module, because they have the close distance. Object 6 and 7 are also merged into one module with the same reason.

When the control of the toolkit is passed to the code generation part, the code of the merged objects is automatically integrated into the one class, and it is reformed into the one object. Figure 8 shows the example of several couplings performed sequentially. Figure 8(a) is the original structure of the program. Figure 8(b) shows the sound coupling, where the object A, B, and C are merged into one module, and the other modules are also merged. And, the structure of the program changed to be shown in Figure 8(c) when the program is reorganized. Nineteen modules are reorganized into the seven modules, as shown in Figure 8(b), and the structure of the program is changed to be shown in Figure 8(c). Then again, another coupling, color coupling, is performed, as is shown in Figure 8(d). These coupling can be performed repeatedly any time with any operations including decoupling operation [9].

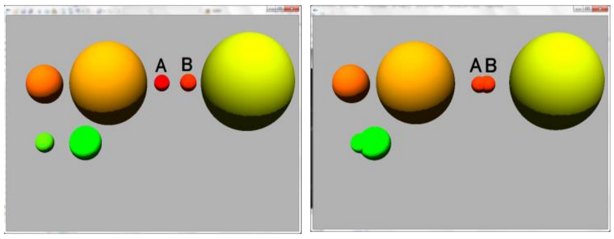


(a) Before the Coupling (b) After the Coupling

Figure 7. Cohesion Coupling

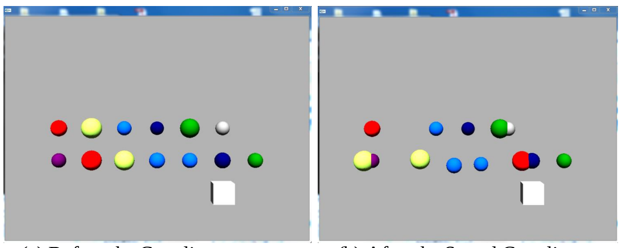


(a) Before the Coupling (b) After the Sound Coupling



(c) Reformed Objects (d) After the Color Coupling

Figure 8. Several Coupling Repeated



(a) Before the Coupling (b) After the Sound Coupling

Figure 9. Sound Coupling Visualized

TABLE II. RESULT OF THE SOUND COUPLINGS

(a) BEFORE THE COUPLING

Class name	Metrics M1	Lines	Metrics M2
Bullet	6	48	6
Enemy	7	109	13
Game	2	21	9
GameObject	2	9	1
KeyInput	6	91	13
Level	3	18	4
MyBullet	4	23	4
Mycanvas	7	101	11
Objectpool	12	149	10
Particle	4	41	7
Player	4	33	6
Score	8	43	3
Title	3	29	7
Average	5.2	55	7.2

(b) AFTER THE COUPLING

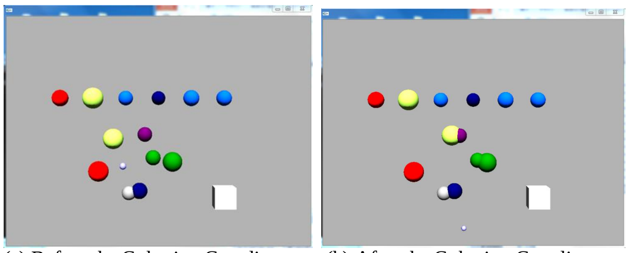
Class name	Metrics M1	Lines	Metrics M2
Bullet	8	48	6
EnemyMy	11	132	8.5
Game	2	21	9
GameObjKe	8	100	7
Level	3	18	4
MycanSco	15	144	7
Objectpool	12	149	10
Particle	4	41	7
Player	4	33	6
Title	3	29	7
Average	7	71.5	7.2

C. *Experimental Results*

The sound coupling and the cohesion coupling are examined, and the results are presented.

Figure 9 shows the before and the after of the structure of the modules of the reengineering program. Figure 9(a) shows the original program structure, and (b) shows the structure of after the sound couplings. The cube in Figure 9 shows the object that has the typical sound in which the coupled object to be met. In the experiment, we assumed the appropriate value for M1 is 10, and M2 is 7, taking the experimental values from the reference [1]. Table II shows the metrics of the before and the after of the sound coupling. The program that has 13 classes has changed to the 10 classes. The metrics M2, which is the number of methods per class, did not change. On the other hand, the metrics M1, which is the number of lines per method, changed from 5.2 to 7. This means the reengineering performed by sound coupling has been succeeded.

The second experiment, which is the cohesion coupling, was performed using the same program as the sound coupling. Figure 10 and Table III show the experimental results performed. Figure 10(a) shows the original program structure. The location of the objects is calculated using the cohesion distance formula (1) and the MDS. The detail of the MDS is shown in the Appendix. The metrics M1 has changed from 5.2 to 7.5, and metrics M2 has changed from 7.2 to 7.6. This means that the M1 is improved, and the structure is well organized.



(a) Before the Cohesion Coupling (b) After the Cohesion Coupling

Figure 10. Cohesion Coupling Visualized

TABLE III. RESULT OF THE COHESION COUPLINGS

Class name	Metrics M1	Lines	Metrics M2
Bullet	6	48	6
Enemy	7	109	13
Game	2	21	9
GameObject	2	9	1
KeyInputTi	9	120	10
LevelSco	11	61	3.5
MyBulletOb	18	172	7
Mycanvas	7	101	11
Objectpool	12	149	10
Particle	4	41	7
Player	4	33	6
Average	7.5	78.5	7.6

IV. CONCLUSION

This paper described the experiences of the software reengineering using the toolkit we developed. The sound coupling and the cohesion coupling for reorganizing the program were introduced. And, some experiments for software reengineering for using these coupling were presented. The results showed that the reengineering was performed well by the couplings using the haptics through tangible interface.

The toolkit was built on the concepts of easy to understand, easy to use, and being use not only for the professionals but also for every person who is unfamiliar with the software metrics. Therefore, the tool can be used not only for the software professionals but also for everybody, for the student who is learning the programming, and for the children who likes to operate the computer just for fun.

The toolkit we developed can be used by the following procedures.

- Metrics are selected according to the metric schema described in section II-B. At least, two metrics are selected.
- The typical values of the metrics for the application are settled.
- Program is analyzed by the toolkit, and the modules that have disharmonized and large metrics value are found. Then, these modules are decomposed into small modules.
- Looking at the two metrics, several modules are coupled together.

The above procedures are repeated until both of the metrics stabilize into the appropriate values.

We are testing more cases for software programs using the toolkit. Reengineering the software depends a lot on the program properties. We need more experiences for the validation of the toolkit to be useful. The other aspect of the toolkit, the programming toolkit for education, is our next concern.

ACKNOWLEDGMENT

This work is supported by JSPS KAKENHI Grant Number 15K12174.

REFERENCES

[1] M. Lanza and R. Marinescu, Object oriented metrics in practice, Springer-Verlag Berlin Heidelberg, 2006.

[2] L. C. Briand, J. W. Daly, and J. K. Wust, òA Unified Framework for Coupling Measurement in Object-Orientd Systems,ò IEEE Transactions on Software Engineering, Vol.25, No.1, January/February 1999, pp.91-121.

[3] F. B . e Abren, G. Pereira, and P. Sousa, ñA Coupling-Guided Cluster Analysis Approach to Reengineering the Modularity to Object-Oriented Systems,ò Proc. of the Fourth European Conference on Software Maintenance and Reengineering (CMMR), 2000, pp.13-22.

[4] S. Demeyer, S. Ducasse, and O. Nierstrasz, ñFinding refactorings via change metrics,ò OOPSLA ô00, 2000, pp.166-177.

[5] S. Demeyer, S. Ducasse, and O. Nierstrasz, Object-Oriented Reengineering Patterns, free download from http:/scg.unibe.ch/oorp/ , 2013, [retrieved: 01, 2016].

[6] R. Wettel, M. Lanza, and R. Robbes, 2011, ñSoftware System as Cities: A Controlled Experiment,ò Proc. of the 33rd International Conference on Software Engineering (ICSEô11), 2011, pp.551-560.

[7] O. Greevy, M. Lanza, and C. Wysseier, ñVisualizing Live Software Systems in 3D,ò Proc. of the 2006 ACM Symposium on Software Visualization(SoftVis06), 2006, pp.47-56.

[8] S. R. Schach, From Modules to Object-Oriented & Classical Software Engineering, McGRAW-Hill Onternational Edition, 2007.

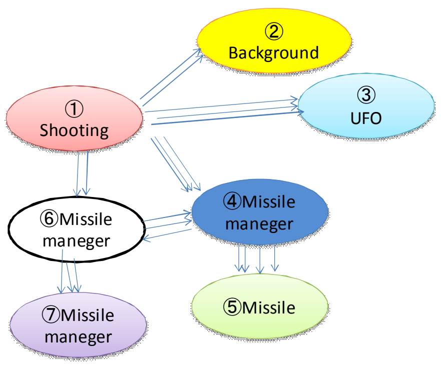
[9] M. Yoshida, S. Okumura, and N. Iwane, ñSoftware Reengineering Toolkit with Tangible Interface by Haptics,ò Proc. of the 9th International Conference on Software Engineering and Applications (ICSOFT-EA), Aug. 2014, pp.351-356.

[10] F. Simon, S. Loffer, and C. Lewerentz, ò Distance Based Cohesion Measuring,ò Proc. of the 2nd European Software Measurement Conference (FESMA), 1999, pp.69-83.

[11] J. M. Bieman and B. Kang, ò Meaduring Design-Level Cohesion, ñ IEEE Transactions on Software Engineering, Vol 24,No.2, February, 1998, pp.111-124.

[12] S. Nishisato, Y. Baba, H. Bozdogan, and K. Kanefuji, Measurement and Multivarite Analysis, Springer, pp.27, 2001.

APPENDIX



(a) Class Dependency

Similarity	1	2	3	4	5	6	7
1	0.00	0.90	0.80	0.79	1.00	0.91	1.00
2	0.90	0.00	1.00	1.00	1.00	1.00	1.00
3	0.80	1.00	0.00	1.00	1.00	1.00	1.00
4	0.79	1.00	1.00	0.00	0.63	0.79	1.00
5	1.00	1.00	1.00	0.63	0.00	1.00	1.00
6	0.91	1.00	1.00	0.79	1.00	0.00	0.57
7	1.00	1.00	1.00	1.00	1.00	0.57	0.00

(b) A Similarity Matrix among Classes

In.P.	1	2	3	4	5	6	7
1	0.329	-0.015	0.061	-0.003	-0.136	-0.094	-0.141
2	-0.015	0.459	-0.054	-0.125	-0.071	-0.116	-0.077
3	0.061	-0.054	0.433	-0.138	-0.084	-0.129	-0.089
4	-0.003	-0.125	-0.138	0.291	0.143	-0.009	-0.160
5	-0.136	-0.071	-0.084	0.143	0.399	-0.146	-0.106
6	-0.094	-0.116	-0.129	-0.009	-0.146	0.309	0.185
7	-0.141	-0.077	-0.089	-0.160	-0.106	0.185	0.389

(c) An Matrix of the Inner Product

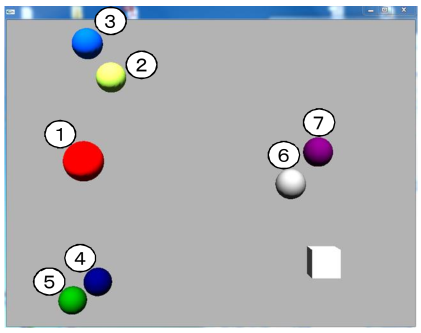
Eigenvector	x	y
1	-0.263	0.292
2	-0.157	0.34
3	-0.249	0.478
4	-0.208	-0.49
5	-0.306	-0.566
6	0.539	-0.092
7	0.645	0.037

(d) Eigenvector

The class dependency of the program, shown in Figure (a), is transformed to the similarity matrix, shown in Table (b).
The inner product, Table (c), is calculated from the similarity matrix by using the Young-Householder translation theorem [12].
The eigenvalue and eigenvector, Table (d), are calculated from the inner product. Two eigenvalues are selected. The eigenvalues selected are 0.756 and 0.690.
Table (e) shows the x and y coordinate of the objects calculated from the eigenvector. Finally, the distance of the similarity is visualized as the program structure, as is shown in Figure (f).

Coordinate	x	y
1	-0.23	0.243
2	-0.137	0.283
3	-0.217	0.397
4	-0.181	-0.407
5	-0.266	-0.47
6	0.47	-0.077
7	0.561	0.031

(e) Location of the objectS



(f) Program Structure Visualized by cohesion

Figure 11. Multidimensional Scaling Method (MDS) based on the Similarity Matrix