Design and Implementation of the Reuse-Easiness Measurement System Using Fuzzy Logic

Sung-joo Lee*, Wan-gyu choi**, Hwan-mook Chung***

※This study was supported by research funds from Chosun University, 1995.

ABSTRACT

Software reuse is a method which increases the productivity of software, nevertheless software reuse is not employed well in real world. One of The important factors occured this proplem is insufficient inforamtion in understanding and adapting the existing components. Understanding and adapting of components can be measured employing user's experience and the attributes which the existing programs provide. Especially user's experience is very important attribute in understanding and adapting components, and it can't be measured by simple metrics.

We propose in this paper, the reuse-easiness measurement system using fuzzy logic. This system can provide information regarding reusing components by reflecting user's experience from user's point of view, and can reduce the reuse effort significantly.

Ⅰ. Introduction

software engineering made slow progress, as compared with the rapid progress of hardware, and computer scientists have studied ways to cope with this situation.

Statics showed that 20-30% of the developed software modules is restricted to specific program and 70-80% is generally used by many programs[2]. The software development environment has drawn considerable attention in recent year and researchers concern the method to change user's requirement to system's requirement and to automatize the process of analysis and design. Owing to increase of cost and scale on development of software, personnel and cost which are requisite of maintaining and updating it are needed more. One way is to utilize reusable information obtained from existing software in current software development.

These reusable information are "codes(program modules)", "software architecture", "functional collections", "generic systems", "development knowledge", "application-area knowledge", "tech-transfer knowledge", and "utilization knowledge"[2]. Specially speaking, executable codes can be often viewed as the primary product of the programmer and their effective reuse be one of the oldest objectives of software reuse technology. After saving it in the component library, the reusable code units which are retrieved from the existing programs or are developed taking reuse into consideration on development of software, many

*Dept. of Computer Science, Chosun Univ., Professor
**Dept. of Computer Science, Chosun Univ., Ph. D student
***School of Electrnics & Information Eng. of College of Eng., Catolic Univ. of Taegu-Hyosung, Professor

researches which try to reuse them are in process.

Previous researches illustrated that component (below, this is "a reusable code") reuse increased the software productivity greatly[2]. However, component reuse is not employed well because of the following factors. First, most projects have the restricted budget and development schedule, programmers do not have enough time to search and create the reusable softwares. Second, programmers do not trust the codes which others generate. Third, programmers have difficulties of understanding and adapting the existing codes. Fourth, the standard system of the reuse management and support does not exist.

To solve problems described above, many researches are in process, which take reuse into consideration on developing the software, or reduce the user's search effort by constructing the effective classification and search system, or product the standard system of the reuse management and support[2][4].

Component reuse consists of three steps: accessing the existing codes, understanding them, and adapting them[2]. In accessing the existing codes, users find components that fit the purpose. If the appropriate components exist, users reuse them. If the searched components are very similar in function and differ only in minor implementation details, users modify and reuse them. If not, users create the new components. In understanding step, users find the most appropriate components by analyzing these components obtained at the previous step. In adapting step, users implement the new software after modifying the searched components and composing them.

In fact, components searched through accessing the existing codes may be very similar in function and differ only in minor implementation details. In this case, simple arrangement of components make it difficult to understand and adapt the existing components. Consequently, reusers would give up reusing the existing components, especially if lots of the similar components exist. A new system to evaluate the degree of effort required to apply components is needed to help users decide whether they reuse the existing components or create them.

But most of researches are concerned with quality measurement on obtaining and registrating the component[2][3]. The major purpose of these models is to find components that take good quality. Or as some component management systems arrange components in order corresponded to the range of metrics which users select, they help users to understand and adapt components[7]. But they had better take an interest in program's point of view than user's, and especially did not reflect user's experience.

On reusing components, understanding and adapting of components can be measured employing user's experience and the attributes which the existing programs provide; program size, logical structure, documentation, and etc. Especially user's experience is very important attribute in understanding and adapting components.

In this paper, we propose the reuse-easiness measurement system. This system measures effort of unde-
rstanding and adapting of components by reflecting the reuser's experience from reuser's point of view, and provides information, and help users to reuse the existing components in the least effort. In section 2, we describe and analyze the existing quality measurement models previously published and their problems. In section 3 is the central contribution of this work and this chapter proposes the reuse-easiness measurement model using fuzzy logic. In section 4, we explain a result of a experiment of this system. In secton 5, We present conclusion from an experimental result and future studies.

Ⅰ. The existing measurement models

Software may take three attributes; program size, program structure, and data structure. They are a special case of attributes that a module contains, excluding relevancy with other modules. The metrics

that measure these attributes can be classified into size metric which measures program size, logical structure metric which measures the logical structure of program, data structure metric which measures relation of data in program, and hybrid metric which measures complex attributes, not only one attribute. Table. 1 shows the classification of quality measurement metrics.

The reusability measurement model determines whether software components can be reused or not, based on several metrics. The representative models using these metrics are CARE system and McCall's Reuse Quality Factors.

Table 1. Classification of quality measurement metrics

Metrics	Methods and examples
Size	Lines of codes, Number of tokens(Halstead's & Jensen's metrics), Number of functions
Logical structure	Cyclomatic number. Prime metrics, Metirc of the nested structure, Nostructural metrics
Data Structure	Metrics of quantity of data Metrics of usable form of data Metrics of data relationship
Hybrid	Hansen's metrics, Oliedo's metrics, etc.

2.1 CARE System

As illustrated in Fig. 1, CARE system defines functional usefulness, reuse costs and quality as the quality factors, which affect the reusability. Functional usefulness includes the commonality and the variety of functions performed by components. Reuse costs are the required costs extracting the component from the old system, packing it into a reusable component, modifying it, and integrating it into the new system. The concept of quality includes correctness, readability, testability, ease of modification, and performance that are important for component reuse. These were measured by four metrics as following: Halstead's volume, complexity of module to be measured by McCabe's cyclomatic number, component regularity to be measured by the closeness of the estimate to the actual size, reuse frequency according to the number of calls addressed to a component.

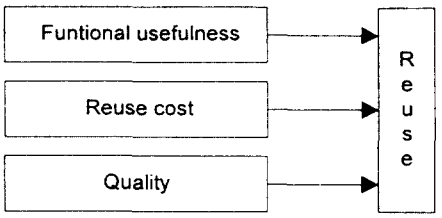

Fig. 1 The Reuse Quality Factors of CARE system

2.2 McCall's Reuse Quality Factors

As illustrated in Fig. 2, McCall defines 'generality', 'modularity', 'self documentation', 'hardware independency', and 'software system independency' as the quality factors which have effects on the software reuse[8]. Generality, which is the most fundamental property, is called the degree to be utilized not only for specific application areas and domains but also for general purposes.

Since the basic unit of the software reuse is module, each module should have such fundamental conditions as abstraction and information hiding and furthermore, the maximum cohesion and the minimum coupling.

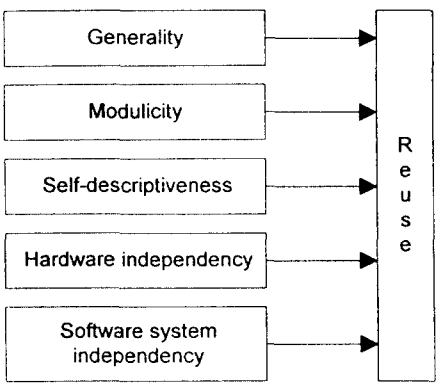

Fig. 2 McCall's Reuse Quality Factors

A modules to be reused have nothing to do with other softwares and hardwares to be executed. And for the reuse of software components, it is necessary to inform the correct function, usage, and interface of modules.

2.3 Problems of The Existing Measurement models

The existing complexity measurement metrics, or CARE System and McCall's quality factors to use them do not have an interest in user's point of view, but program's. As they do not provide the appropriate information required in order to understand and adapt the existing components, software is not reused very well. Especially, in case that many components are very similar in function and differ only in minor implementation details, a simple array of these metrics does not help users select the appropriate component.

CARE system tries to measure exactly functional usefulness, reuse cost, and quality of modules selected for reuse, employing four metrics. McCall's reuse quality factors try to measure reusability using the five metrics. Howerever, they do not reflect user's experience which is a very important element for understanding and adapting components, and do not measure how much each element affects user's understanding and adapting of program.

Therefore, based on theses problems, we propose the measurement model focused on user's experience.

Ⅲ. Reuse easiness measurement model

3.1 Definition of measurement factors

Component reuse consists of accessing the existing codes, understanding them, and adapting them. Components searched would rather be very similar in function and differ only in minor implementation details than exactly fit the development purpose.

In this case, as the model proposed in this paper measures easiness of understanding and adapting each component and then provides the measured value for users, it increases reusability and helps users reuse the existing components in the minimum effort. Or through increasing of the rate of reusability of the existing components, this model tries to reduce term and cost of software development, and to increase the software productivity.

This model defines targets and factors of measurement as like in Fig. 3, and obtains reuse-easiness through fuzzy inference of value measured by using factors directly or indirect metrics.

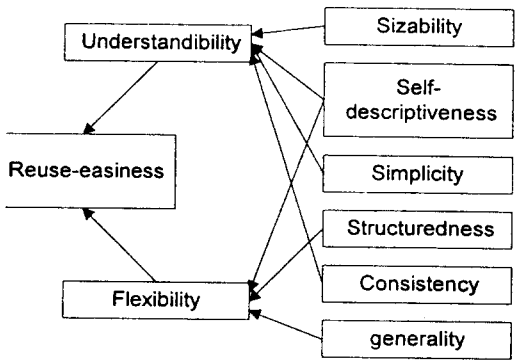

Fig. 3 Relating reuse quality characterishtics

Reuse-easiness is "how easy can users understand and adapt program", and is affected by the understandability and flexibility of components. Understandability is "how easy can user understand program", and is measured by sizability, self-descriptiveness, simplicity and consistency. Flexibility is "how easy can program be modified on appending the new function and on adapting it in new context", and is measured by self-descriptiveness, simplicity, structuredness, generality. A description of each factor is showed in Table 2.

3.2 Definition of Metrics

Metrics for measurement factors of reuse-easiness are showed in Table 3.

This paper assumes that the used languages are same. Because users have their own criteria and acceptable range, based mainly on experience, for

Table 2. Description of Measurement factors

Factors	Description
Sizability	How appropriate the program size is?
Self-descriptiveness	How obvious program's purpose and elements of input and output is? Or how readable program is?
Simplicity	Logical size which is intrinsic in program is simple enough to undestand without difficulty?
Structuredness	Program is structural enough and stable to be modified easily
Consistency	How consistent the used languages and symbols and expression rules are?
Generality	How often program is used to many developers?

Table 3. Metrics for the reuse easiness measurement factors

Facrots	Metrics	Criteria of reuse easiness
Sizability	LOC (Lines of code)	small
Simplicity	Halstead's Volume	simple
Self-descriptiveness	Subjective rating (1 to 100)	excellent
Structuredness	Cyclomatic number	structural
consistency	Used language	same
generality	Reuse Frequency	general

determining what "small", "simple", and "excellent" mean, and they do not have precise value, we evaluate them by using fuzzy set and control theory.

Fuzzy set theory, a method that represents the uncertain situations of the real world, was introduced by Lofti A. Zadeh in 1965. A fuzzy set is a class with fuzzy boundaries, which may be characterized by associating a membership grade in the class with every object that could be in the class. A fuzzy function maps a set of value in a fuzzy set to the interval [0, 1][2]. This characteristics of fuzzy set are enough to represent the criteria and the acceptable range not to be represented by quantitative value.

3.2.1 LOC

LOC, a metric which measures size, affects quality and understanding of software. LOC is a method which counts lines of code in order to measure the program size, and it contains all lines except blank lines and comment lines. Furthermore it includes program header, declaration, and nonexecutable statements. This method is not consistent, however it has a merit that user can know it intuitively. Even though there are many opinions for the appropriate lines of code, a module consisting of 50 to 80 lines is appropriate, not exceeding more than one page. LOC affects sizability.

3.2.2 Volume

Logical size which is intrinsic in program is measured by Halstead's software science. It defines program as a set of independent elements called "operator" and "operand" as following.

n_1 : the number of distinct operators that appear in a program

n_2 : the number of distinct operands that appear in a program

N_1 : the total number of operator occurrences

N_2 : the total number of operand occurrences

Formular defined using operators and operands are as following

$$N(\text{length}) = n_1 \log 2\ n_1 + n_2 \log 2\ n_2$$

$$V(\text{volume}) = N \log (n_1 + n_2)$$

The volume of component affects both simplicity and structuredness. If volume is too large, it is more difficult to understand and adapt components. Therefore the appropriate range of volume is about 2 to 10

[4].

3.2.3 Cyclomatic number

The cyclomatic number, one of the logical structure metrics, is the complexity measurement method proposed by McCabe. It measures the complexity of program as following, based on program's control flow graph.

$$V(G) = e - n + 2$$

e: the number of edges in the graph G,
n: the number of nodes in the graph G.

It is used as metric to measure the component's structuredness. The more structural program is, the easier the program's adaptation is. McCabe's classification is Shown in Table 4.

Table 4. Complexity Classification of modules

Cyclomatic number	Description
less than 5	very simple program
5 to 10	very structural and stable program
more than 20	Problem is very complex or structure is more complex than it needs
more than 50	very nonstructural and unstable program

3.2.4 Documentation

Documentation measures its overall relevance to program understanding. The more excellent it is, the easier to understand the program is. But the same documentation quality value may contribute more to novice than to user with experienced. Documentation is measured by lines of comment and is defined as following;

$$DOD = \frac{LOCO}{LOC}$$

DOD: Degree of documentation
LOCO: Lines Of Comment
LOC: Lines Of Code

3.2.5 Reuse Frequency

Reuse Frequency can be measured by comparing the number of calls addressed to the total components in component library and the number of calls addressed to the specific component. Because the measurement model presented in this paper does not find the reusable modules but searchs components previousely saved to component library and measures reuse-easiness of them, it assumes that if components are called, the number of calls is accumulated automatically. The use of component library is on the increase, the reuse frequency has the more precise value. Reuse frequency is defined as following.

$$RF = \frac{NCSC}{NCTC}$$

RF: Reuse Frequency
NCSC: Number of Calls to a Component
NCTC: Number of Calls to Total Component

3.2.6 Acceptable range of metrics

In order to help users select the appropriate criteria, this measure model provides criteria and the acceptable range for metrics defined above in Table 5. Users can select their own criteria and acceptable range by modifying them in system.

Table 5. Ranges of acceptable values

Metrics \ Acceptable Ranges	min	criteria	max
LOC (Lines of Code)	0.0	2.5	100
CN (Cyclomatic Number)	0.0	10.0	50.0
Vol (Volume)	0.0	2.0	10.0
DOD (Degree of Documentation)	0.0	0.5	1.0
RF (Reuse Frequency)	0.0	0.5	1.0

3.3 The Measurement Model using fuzzy logic

This measurement model fuzzifies the metrics's value using fuzzy set, and measures their relevance to reuse-easiness using fuzzy logic. Because fuzzy logic is more similar to our thought or natural language than the existing logic, it can be used effectively to describe uncertainty of real world. The central part of fuzzy logic is a sequence of the linguistic control rules, which involve the fuzzy relationship. Or it can change the uncertain phenomena into precise value, using fuzzy composition rules based on fuzzy relationship [6]. Fig. 4 illustrates the measurement model proposed on this paper.

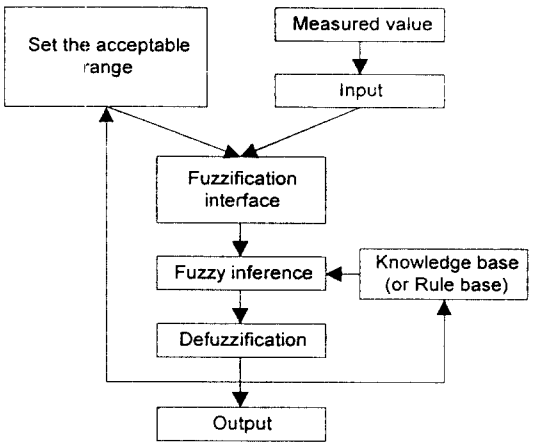

Fig. 4 Reuse-easiness measurement model

3.3.1 Setting of acceptable range and Input system

Users can either use criteria and acceptable range of the five metrics defined in Table. 6 or set their own criteria and acceptable range using feedback mechanism. The value set by users is employed as membership function on fuzzification interface. Input system provides measurement value for five metrics analyzed by this model for fuzzification interface.

3.3.2 Fuzzification interface

Fuzzification interface fuzzifies input value by changing input data into appropriate linguistic value. The linguistic value generally make use of name of sets defined within universal set. Universal set of input value is separated into the linguistic value, called "NB(Negative Big)", "PB(Positive Big)". Membership function of linguistic value is shown in equation. 1 and is illustrated in Fig. 5.

$$A(x) = (\min, mid, \max)$$

$$\mu(x) = 0, \qquad x < \min$$

$$= \frac{x - \min}{mid - \min}, \quad \min \le x \le mid \quad -(eq1)$$

$$= \frac{\max - x}{\max - mid}, \quad mid \le x \le \max$$

$$= 0, \qquad x > \max$$

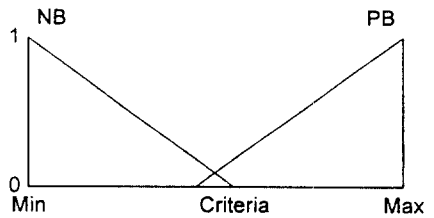

Fig. 5 The membership function for fuzzification

3.3.3 Knowledge Base (or Rule Base)

Knowledge base describes inference rules to use the fuzzified value. Table. 6 shows inference rules of this model.

Table 6. Inference Rules

Understandability / Flexibility	NB	PB
NB	NB	PS
PB	NS	PB

3.3.4. Fuzzy inference

The fuzzy inference presents a method to make human's decision making using fuzzy relation and fuzzy logic. We use the Max-Product method among many inference methods. It takes the maximum of products of membership value.

$U_0 = \max_{i,j} [U_{undestanding} \cdot U_{flexibility}]$

3.3.5 Defuzzification interface

Defuzzification changes into scalar value the result which is inferred using human's fuzzification rules and information. In this paper, we map scalar value to interval [0, 1], and based on these value, provide information on reuse-easiness of components. Defuzzification is as follows.

$U_0 = \text{defuzzifier}(U)$

U_0 : scalar value
U : value inferred

There are many defuzzification methods; Max Criterion method, Mean of Maximum Method, Center of Area Method, etc. Among them, we use the Center of Area Method, which takes long time due to many computations, but leads to precise result[6]. Equation is as follow and Fig. 6 illustrates the defuzzification function.

$$U_0 = \frac{\sum_{j=1}^{k} \mu(\mu_j) \cdot \mu_j}{\sum_{j=1}^{k} \mu_j} \qquad \text{(Eq.2)}$$

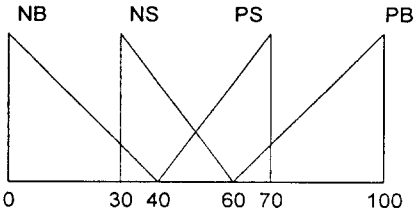

Fig. 6 The membership function for defuzzification

Ⅳ. Experiment and Result

4.1 Measurement of reuse easiness

Table. 7 shows the input value for four modules (internal sort).

Table 7. Input values for modules

Module name \ Input value	LOC	CN	DOD	Vol	RF
A	21	4.00	0	0.21	0
B	37	9.00	0	0.37	0
C	27	5.00	0	0.48	0
D	26	7.00	0	0.31	0

If Fuzzification function is table. 5, table. 8 shows the fuzzification result of four modules.

Table 8. Fuzzification of input value

Module \ Input		LOC	CN	DOD	Vol	RF
A	NB	0.222	0.636	1	0.892	1
	PB	0	0	0	0	0
B	NB	0	0.181	1	0.813	1
	PB	0.181	0	0	0	0
C	NB	0	0.545	1	0.758	1
	PB	0.051	0	0	0	0
D	NB	0.037	0.363	1	0.842	1
	PB	0.038	0	0	0	0

Table. 9 shows undestanding and flexibility of four modules measured using information Shown in Table. 8.

Table 9. The result of undertanding and adapting

Module \ Measured value	Understand-ability	Adapting
A	24.66564	11.18881
B	30.80028	25.17483
C	29.56159	13.98601
D	27.54806	19.58042

Table. 10 shows the result which is inferred using the Max-Product methods and is defuzzified using Eq. 2.

Table 10. The result of reuse-easiness

Module name	Reuse-easiness
A	23.49782
B	31.58748
C	27.38205
D	28.27335

4.2 Explanation of Result

We tested many modules using table. 5 as fuzzification function, and could classify modules as like table. 11. Note that if fuzzification function is changed, values of table. 11 also are changed.

Table 11. classification of modules

reuse easiness	description
less than 20	Because modules are very simple, they can be reused very easily
20 to 30	Modules is a little complex, but they can be reused without difficulty
more than 35	Because modules are more complex than it needs, users can give up reusing them

In the above experiment, if users have no information on the module reuse, they must inspect all modules. In this case, reuse-easiness of four modules is described in table. 12. If users have no information of table. 10, their reuse-easiness is 35 and they can give up reusing the existing modules. But if information of table. 10 is given them, users easily find the appropriate module "B", and can reuse it without difficulty.

Table. 12 also shows that the more the number of components is, the greater the degree of reuse-easiness is.

Table 12. The total reuse-easiness for modules

Input value		LOC	CN	DOD	Vol	RF
		111	25	0	1.385	0
Fuzzification	NB	0	0	1	0.307	1
	PB	1	0.390	0	0.042	0
Understandability		37.91237				
Flexibility		32.08255				
Reuse-easiness		35.84926				

V. Conclusion

Component reuse is a method which increases the productivity of software, nevertheless, component reuse is not employed well on real world. One of important factors occured this proplem is insufficient inforamtion about understanding and adapting components.

In this paper, we propose the reuse-easiness measurement system that measures effort of understanding and adapting components, and can provide information used on reusing components by reflecting user's experience from user's point of view. By measuring effort of understanding and adapting components and providing its information, this system has advantages as following.

First, if lots of the similar components exist, users know what components can be used by less effort.

Second, In case of finding the appropriate components, this system can decreases the reuse effort by making users inspect components reused easily first of all, or by preventing users from inspecting components which require the excessive reuse effort.

Reference

1. Ted J. Biggerstaff and Alan J. Perlis, "Software Reusablility] volume 1 Comcepts and Models", New York: ACM press, p. 99~123, 1989.
2. Freeman, "TUTORIAL: Software reusability", IEEE, pp. 2~8, pp. 10~23, pp. 50~56, pp. 106~116, pp. 151~154, pp. 155~178, 1987.
3. Rogers S. Pressman, "Software engineering: A Practitioner's approach", NewYork: McGraw-Hill

Book Company, pp. 452～458, 1987.
4. Caldiera, G. and V. R. Basili, "Identifying and Qualifying Reusable Software Components", IEEE Computer, pp. 61～70, 1991.
5. Tracz W., "Tutorial:Software Reuse:Emerging Technology", IEEE, pp. 18～22, 1988.
6. H. Y. Lee, "Fuzzy theory and application volume 2", Seoul:Hong-rong press, pp. 1-1～1-40, pp. 5. 1～5. 91, 1991.
7. K. H. Lee et., "A Study for software reuse", Dept. of Science, BSN20592, pp. 63, 1989.
8. Y. J. Song, "Software engineering", Seoul:Hong-rong press, 1992.
9. S. R. Ryoo, "Algorithm for C", DadaMedia, 1988.

이 성 주(Sung-joo Lee) 정회원
1970년 2월:한남대학교 물리학과 졸업(이하사)
1981년 2월:조선대학교 대학원 경제학과 졸업(경제학석사)
1992년 8월:광운대학교 대학원 전자계산학과 졸업(이학석사)
1988년~1990년:조선대학교 전자계산소 부소장, 전자계산학과 학과정
1990년~1994년:조선대학교 산업대학 교학과장
1984년~현재:조선대학교 전자계산학과 교수
1995년~현재:조선대학교 산업대학 학장
※관심분야:Rough Set & Fuzzy logic Software engineering, Object-oriented system

최 완 규(Wan-gyu choi) 정회원
1983년 2월:광주 숭일고등학교 졸업
1983년 3월:서울대학교 인문대학 철학계열
1988년 8월:서울대학교 인문대학 종교학과
1992년 5월:(주) 공성통신 전산실 입사
1993년 6월:(주) 공성통신 퇴사
1993년 6월:(주) 한양 시스템 전산실 입사
1994년 7월:(주) 한양 시스템 퇴사
1994년 9월:조선대학교 대학원 전자계산학과 석사과정 입학
1996년 현재:조선대학교 대학원 전자계산학과 재학중
※관심분야:소프트웨어 재사용, 객체지향 시스템