Reusability Metrics for Program Source Code Written in C Language and Their Evaluation

Hironori Washizaki[1], Toshikazu Koike[2], Rieko Namiki[3], and Hiroyuki Tanabe[3]

[1] Waseda University, 3-4-1, Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
GRACE Center of National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 Japan
washizaki@waseda.jp
[2] Yamaha Corporation, 10-1 Nakazawacho, Naka-ku, Hamamatsu-shi, Shizuoka, 430-0904 Japan
toshikazu_koike@gmx.yamaha.com
[3] Ogis-RI Co., Ltd., MS-Shibaura Bldg., 13-23, Shibaura 4, Minato-ku, Tokyo, Japan
{Namiki_Rieko, Tanabe_Hiroyuki}@ogis-ri.co.jp

Abstract. There are various approaches to quantitatively and statically measuring the reusability of program source code; however, empirical demonstrations of the effectiveness of such approaches by considering actual reuse in actual development projects or of the magnitude of their effect on actual reusability have not been reported in depth. In this paper, we identified a set of metrics that are thought to be effective for measuring the reusability of C language program source code. Subsequently, for ten projects involved in development with existing software modification and adoption, during which conventional source code in an old project are extensively reused and adopted to a new project, we compared values of the static metrics identified and the reuse results before and after the development. Statistical analysis demonstrated that some of our metrics are effective for actual software development, and we accurately determined the magnitude of their effect on actual reusability. More concretely, it was found that when the percentage of files used outside the belonging directory is small and the number of function calls is small, the complexity of source code as the material of reuse and factors that are affected by the source code are limited, indicating high reusability.

1 Introduction

Reuse of common parts in some form (such as functions and directories) can reduce the development cost of new software[1]. Reusability is the extent to which a system or module-unit parts can be reused in a different environment. This quality characteristic is not regulated directly in the standard quality model specified in ISO9126-1[2]; however we have to consider its importance, particularly with respect to development efficiency within the same problem domain.

It has been said that you cannot control what you cannot measure[3]. A software metric is a measurement scale and the method used for measurement of

some property of software. Metrics for objectively evaluating the ease of reuse of target objects are essential for systematically controlling and conducting "development for reuse" and "development by reuse." Moreover, it is difficult to apply such reusability metrics in the actual development of software with satisfactorily high practicality, understandability, and persuasiveness unless the effectiveness of these metrics has been proven by accurate and empirical evaluation tests.

In this paper, we proposed a set of metrics for statically measuring (i.e., analyzing without executing programs) the reusability of C language program source code, and statistically evaluated the effectiveness of our metrics using a certain scale of actual metric values. The objective of our study is to provide a validated way for evaluating the ease of reuse of program source code accurately, without any other software materials such as documents and specifications.

The remainder of this paper is organized as follows. Next section introduces related works and problems in reusability measurement. Section 3 describes our reusability metrics and reuse rates for further evaluation. In section 4, we report on the results of empirical evaluations of our metrics by using ten projects data. In the last section, we draw a conclusion and state future works.

2 Problems of Reusability Measurement

Though many metrics have been proposed, it is generally difficult to select an appropriate one among them or to interpret the measurement results without appropriate models and goals[4, 5]. Various metrics for measuring the reusability of software have conventionally been proposed, mainly for program source code, including our previous researches[6–14].

In most cases, the effectiveness of metrics is evaluated by dividing a certain size of samples into a superior group and an inferior group from particular viewpoints (e.g., reuse results[8] and qualitative evaluation[14, 15]) and by comparing the distribution tendencies of the values of metrics between these groups.

However, data used in the evaluation by conventional metrics are based on the rough frequency of reuse of the partial or entire programs being examined and on qualitative evaluation, which inevitably depends on individuals. Therefore, whether the advantages of highly reusable program elements (or an entire program) are exploited in the programs being examined has not been accurately demonstrated through the consideration of concrete and detailed reuse results [Problem A].

In addition, for a similar reason, it has been difficult to analyze differences in the magnitude of the effect on reusability among individual metrics [Problem B]. Although there has been an approach to qualitative analysis and weighting, i.e., summarizing opinions from multiple specialists, when using multiple metrics[19], the effectiveness of this approach in actual software development has not yet been verified. Hence, the magnitude of the effects of individual metrics on reusability have not been quantitatively clarified.

3 Reusability Metrics and Reuse Results

We qualitatively identified multiple metrics considered to be effective for measuring reusability and compared their actual values with concrete and detailed reuse results. Thus, the effectiveness of individual metrics was accurately evaluated, resolving Problem A.

For the above comparison, regression analysis was carried out using reuse results as the objective variable and the actual values of individual metrics as explanatory variables, and the effect of individual metrics on actual reusability was accurately determined. As a result, Problem B was also resolved. In the following subsections, the proposed reusability metrics and reuse results are explained.

3.1 Definition of reusability metrics

Satisfactory understandability and persuasiveness cannot be obtained in the use of metrics unless the metrics themselves have been systematically derived following a particular policy and theory. In this paper, we adopted the Goal-Question-Metric (GQM) method[16]. The GQM method is a goal-oriented method for mapping a goal to a metric by using a question which must be evaluated in order to determine whether the goal has been achieved or not.

In working towards the goal of the accurate determination of reusability for supporting development for reuse and development by reuse, the questions to be examined were combined stepwise with the metrics used for projecting the properties of software to measures, thus obtaining multiple metrics. Although the process depended on human work, several specialists repeatedly reviewed and modified the hierarchy of the goal, questions, and metrics for as long as about one year, ensuring a reasonable certain level of plausibility and persuasiveness.

Table 1 summarizes the obtained metrics and the questions used to derive them. Seven metrics are derived from the four questions in the table. Each metric represents the complexity of the dependence or the complexity of the application programming interface (API) of the target itself.

For the six metrics other than MFl04, the smaller the value, the more positive the response to the corresponding question, which consequently indicates high reusability. For MFl04, it is considered that there is an appropriate range of values that give high reusability.

Because MFn05, MFn07, and MFn06 are metrics that evaluate functions, it is necessary to summarize measurement results by summation or other means in the case of evaluating upper unit levels, such as modules and the entire system. MFl and MMd are intended for source code files and modules (directories for C language), respectively, and similarly to MFn, these metrics require the summarization of results when upper unit levels are targeted.

MMd03 and MFn07 could be interpreted as variations of conventional information flow-based complexity metric called "fan-out"[17]; however MMd03 and MFn07 provide concrete ways to measure external dependencies at the module level and the function level, respectively. Similarly, MMd01, MFl02, MFn05 and

Table 1. List of reusability metrics obtained by qualitative identification

Question	Metric		
	ID	Name	Definition and interpretation
Is not API too complex?	MMd01	Percentage of externally used files	**Definition:** The percentage of files used outside the belonging directory among all files within the directory (module). $\frac{\text{Number of files used outside the belonging directory}}{\text{Number of all files in the directory}}$ **Interpretation:** The smaller the value, the more limited the use of API by external modules, indicating high reusability of the directory (module).
	MFl02	Number of externally used functions	**Definition:** The number of functions defined within the file and used outside the directory (module) to which the file belongs. Even when the same function is used multiple times, it is counted as 1. **Interpretation:** The smaller the value, the more limited the use of API by external modules, indicating high reusability of the file.
Is not the module dependent on too many other modules?	MMd03	Number of dependent modules	**Definition:** The number of modules on which the target module depends. When the target module uses functions of other modules, the former is considered to depend on the latter. **Interpretation:** The smaller the value, the smaller the number of dependent modules, indicating high reusability of the module.
Is the division and allocation of responsibilities appropriately?	MFl04	Percentage of functions without parameters	**Definition:** The percentage of functions without parameters among all functions in the file. **Interpretation:** The greater the value, the smaller the amount of (dependent) data required for use, indicating high reusability of the file. Note that a too high value (e.g., 1.0) indicates difficulty in providing data from outside and thereby difficulty in setting, which may decrease reusability.
Are external components that affect functions appropriately limited?	MFn05	Number of parameters	**Definition:** The number of parameters declared within the argument list of the function. **Interpretation:** The smaller the value, the smaller the amount of (dependent) data required for use, indicating high reusability of the function. Note that a too small value (e.g., 0) indicates difficulty in providing data from outside and thereby difficulty in setting, which may decrease reusability.
	MFn06	Number of readings of external variables	**Definition:** The number of readings of external variables (for C language, external global variables) by the function. When the same single variable is read twice, it is counted as 2. **Interpretation:** The smaller the value, the more limited the dependent external variables, indicating high reusability of the function.
	MFn07	Number of function calls	**Definition:** The number of function calls **Interpretation:** The smaller the value, the more limited the dependent functions, indicating high reusability of the function.

MFn06 are somewhat related to conventional complexity metrics "fan-in"[17] and IF4[18]; however these metrics derived in our study are fine-grained and specific to answer corresponding questions in the obtained GQM model.

3.2 Definition of reuse rate

A specific and detailed analysis of the extent of reuse can be accurately performed by evaluating the reuse rate, i.e., the percentage of components reused without modification (or completely reused) out of the entire adopted components, rather than simply by evaluating the frequency and amount of reuse. Here we define "reuse" and "adopt" in the followings.

- "Reuse" means the use of original components without any modification in the development of other programs.
- "Adopt" means the use without modification, the use with modification, or the use of extracted components; therefore conceptually, "adopt" includes "reuse".

On the basis of the above concept, we defined the reuse rate that can be measured in the context of development with software modification and adoption, and used it to represent the reuse results.

In our study, the reuse rate is defined as the ratio of the number of components reused without modification to that of components adopted in some way during the development of a new project by adopting the entire product of an existing project. A higher reuse rate indicates that the reuse of the product of the original existing project as the reuse source was much easier in such reuse and modification-based new development (i.e. derivative development).

In Figure 1, for example, an existing project as the reuse source is composed of several components, such as A, B, C, D, and E, among which, A, B, and C are assumed to be reused in a new project. Whereas A and B are adopted with some modification, for example, because of a difference in functional or nonfunctional requirements, C is reused without modification; therefore, C is counted in calculating the reuse rate in accordance with the definition of reuse. D and E in the existing project are not adopted at all and are excluded from the calculation of the reuse rate because it is difficult to specify the reason for not adopting D or E, i.e., whether it is because functional or nonfunctional requirements differ in the new project and the existing project (namely, reuse itself is not needed) or because D and E are difficult to adopt in the new project.

On the basis of the above concept, we defined the following three reuse rates: line, function, and file.

$$\text{Line reuse rate} = \frac{\begin{array}{l}\text{Total number of lines in the files reused in}\\ \text{the new project from the existing project}\end{array}}{\begin{array}{l}\text{Total number of lines in the files adopted in}\\ \text{the new project from the existing project}\end{array}}$$

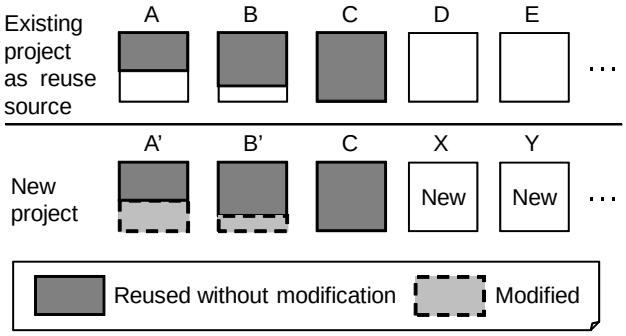

Fig. 1. Concept of reuse rate

$$\text{Function reuse rate} = \frac{\text{Total number of functions in the files reused in the new project from the existing project}}{\text{Total number of functions in the files adopted in the new project from the existing project}}$$

$$\text{File reuse rate} = \frac{\text{Total number of files reused in the new project from the existing project}}{\text{Total number of files adopted in the new project from the existing project}}$$

Given that component in Figure 1 denote files, three files are adopted in the new project, one of which is reused without modification. Therefore, the file reuse rate is 1/3.

4 Evaluating Effectiveness of Reusability Metrics

We statistically evaluated whether the above-mentioned seven reusability metrics were effective for the actual measurement of reusability using a certain size of obtained values and reuse results.

In the following subsections, we explain the projects used to obtain the values of the metrics and reuse results, the process of evaluation based on the comparison of the obtained values and reuse results, and the obtained results.

4.1 Target projects

Data on metrics and reuse results were selected. Specifically, ten existing projects on embedded software development for some instruments (P1–P10) in the same domain were first selected from various projects in a company, in which corresponding succeeding developments (P1'–P10') based on P1–P10 were conducted and from which the above-mentioned three reuse rates (as reuse results) could be obtained. In the company, Px' could be recognized as an extension and new

release of corresponding Px. There was no significant and architectural change between Px' and Px; however it can be said that the software environment of Px' is changed from that of Px because these are different projects with different functional and non-functional requirements. These projects contain relatively recent and reliable data and were selected so that the greatest reuse rate differences could be obtained with the aim of acquiring statistically significant results.

The reuse rates from the above existing projects, P1–P10, to the new derivative projects P1'–P10', respectively, were calculated and used as reuse results. Program source code of P1–P10 were used as the target of application of the reusability metrics. Our expectation is that the higher the reusability of P1–P10, the more components of P1–P10 have been reused in P1'–P10'.

Here, the files of the existing projects include files that were not reused at all in the new projects subsequently developed. Table 2 summarizes the project data. The directories, files and lines that were not reused in the subsequent developments were excluded from evaluation targets because it was unclear whether the reason for the lack of reuse was the difference in the dynamic/static characteristics of the target projects or because reuse was not needed in the functional requirements.

Table 2. Data on scale of ten projects (P1–P10)

	N. directories	N. files	N. effective LOC
Total amount	213	10,298	5,291,096
Reused	173	7,940	3,734,614

4.2 Evaluation process

The process and result of evaluation are chronologically described below. Because there were many parameters that affected the results of our statistical analysis, we carefully examined each step as follows: 1) selection of measurement level, 2) selection of reuse rate and data format of metric value, and 3) selection of explanatory variables (metrics).

Selection of measurement level First, multiple linear regression analysis using the reuse rate as the objective variable and all our metrics as the explanatory variables was carried out for all the reuse rates and the different three measurement levels: file, directory, and system. In this analysis, the objective variable was subjected to the logit transformation[20] because it was proportional data originally so that it is preferable to average the roughness of variation in raw data. For the explanatory variables, three types of data, i.e., raw data, proportional data, and log data, were used, as described later.

For the measurement levels of file and directory, no statistically significant differences were observed regarding all the reuse rates. This implies that the features that affect the reusability of the developed projects exist at the level

of the project itself (namely, the entire system) and that no features that can be observed as statistically significant differences exist in the levels of file and directory.

New projects may require the addition of functions and other items. Therefore, it is appropriate to measure the reusability of the entire system rather than that of some component levels, which depend on individual requirements.

Selection of reuse rate and data format of metric value From the above results, the measurement level was fixed at the system, and multiple regression analysis was carried out for all combinations of the reuse rate and the actual metric value in each data format. The following three types of metric data format were examined:

- Raw data
- Proportional data obtained by normalizing the raw data using an appropriate parameter
- Log data obtained by a log transformation to average the roughness of variation in raw data.

The analytical results revealed that the combination of the function reuse rate and the proportional data of metric values has the highest contribution rate that is adjusted for degrees of freedom and therefore this combination is valid. This is considered to be because raw data are strongly affected by the scale of the project, whereas proportional data have been normalized in accordance with the scale.

Table 3 shows correlation coefficients for all of combinations among the proportional data of seven metric values and the three reuse rates. In the table, it can be seen that the correlation coefficients between the function reuse rate and several metrics proportional data are high compared with other combinations.

Figure 2 shows scattergrams for the function reuse rate and each of seven reusability metrics. From the figure, it is thought that MMd01 and MFn07 are strongly correlated to the function reuse rate compared with other five metrics.

Selection of explanatory variables (metrics) In the above analyses, all seven metrics were used as the explanatory variables; however, the number of target projects is ten, which is small relative to the number of explanatory variables (the number of target projects should preferably be at least double the number of explanatory variables). Therefore, the reliability of the regression equation obtained by multiple regression analysis may be low.

When all seven metrics were used as the explanatory variables, the contribution rate after adjustment for degrees of freedom was 0.812, and the obtained regression equation had positive partial regression coefficients for five of the seven explanatory variables.

Table 3. Correlation coefficient for each combination among the proportional data of seven metric values (MMd01, MFl02, MMd03, MFl04, MFn05, MFn06 and MFn07) and the three reuse rates (LR, FnR and FlR)

	MMd01	MFl02	MMd03	MFl04	MFn05	MFn06	MFn07	LR	FnR	FlR
MMd01	1									
MFl02	0.693	1								
MMd03	0.332	-0.012	1							
MFl04	0.744	0.429	0.491	1						
MFn05	-0.719	-0.354	-0.586	-0.956	1					
MFn06	0.538	0.968	-0.15	0.299	-0.19	1				
MFn07	0.518	0.733	-0.177	-0.11	0.163	0.721	1			
LR	-0.792	-0.561	0.009	-0.328	0.299	-0.446	-0.699	1		
FnR	-0.792	-0.768	0.125	-0.357	0.275	-0.693	-0.792	0.944	1	
FlR	-0.761	-0.715	0.14	-0.38	0.328	-0.639	-0.701	0.915	0.964	1

LR: Line reuse rate after the logit transformation
FnR: Function reuse rate after the logit transformation
FlR: File reuse rate after the logit transformation

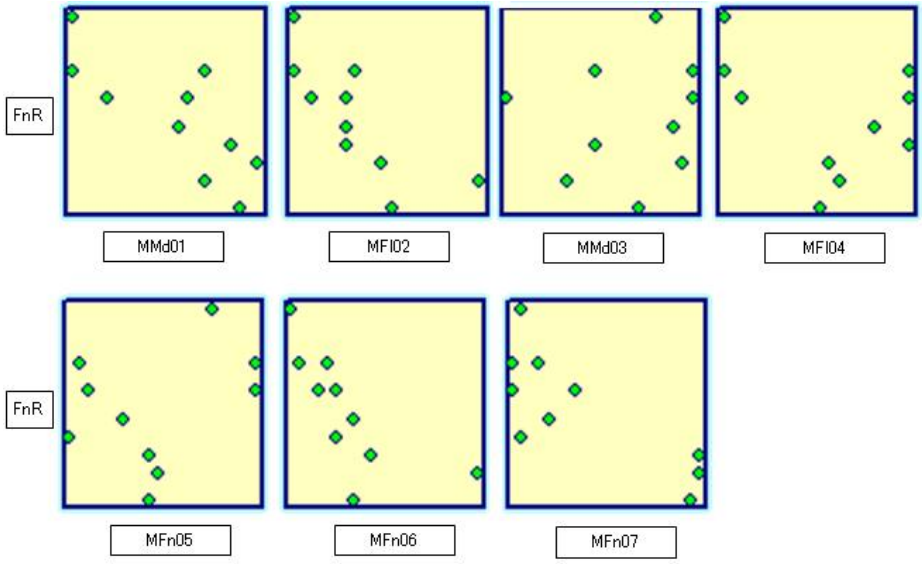

Fig. 2. Scattergrams of the function reuse rate and the seven metrics

As explained above, the six metrics other than MFl04 shown in Table 1 were derived by assuming that the smaller the value, the higher the reusability. The above analytical result was in disagreement with this assumption. This may be because there was originally some correlation among the seven metrics we adopted. As a result, multicollinearity occurred in the multiple regression analysis.

To solve this problem, appropriate explanatory variables were interactively selected using a statistical analysis tool so that no multicollinearity was observed, and we obtained the combination that gave the highest contribution rate after adjustment for degrees of freedom. It was clarified that the best regression equation with a contribution rate of 0.827 after adjustment for degrees of freedom was obtained when MMd01, MMd03, and MFn07 were selected.

Tables 4 and 5 summarize the basic statistics of the analysis and the statistics for each explanatory variable selected, respectively. In Table 4, the variance ratio is high and the level of significance is 1%, indicating that the obtained regression equation is valid. From Table 5, the partial regression coefficients are negative for MMd01 (proportional data) and MFn07 (proportional data), which is in agreement with the initial assumption that the smaller the metric value, the higher the reusability.

Here, the partial regression coefficient for MMd03 (proportional data) is positive. This may be because the three explanatory variables used in multiple regression analysis were not completely independent. However, MMd03 has the smallest standardized partial regression coefficient among the three explanatory variables and thereby has the smallest effect on the objective variable; hence, its effect is considered to be negligible.

Figure 3 shows a scattergram of the function reuse rate estimated from the obtained regression equation with three reusability metric values (MMd03, MMd01 and MFn07), and the actual function reuse rate for ten projects. In Figure 3, the multiple correlation coefficient is as high as 0.941. The estimated function reuse rate at the level of the system obtained using the regression equation is in good agreement with the actual function reuse rate.

Table 4. ANOVA table of basic statistics after selecting explanatory variables

Factor	Sum of squares	Degree of freedom	Dispersion	Dispersion ratio	Test
Regression	10.713	3	3.571	15.34	Level of significance = 1%
Residual error	1.397	6	0.233		
Total	12.11	9			

4.3 Summary of evaluation results

The results of the evaluation of effectiveness are summarized below.

Table 5. Statistics for each explanatory variable selected

Variable	Partial regression coefficient	Standard error	t-value	Standard partial regression coefficient	Tolerance
Constant term	3.446	1.769	1.948		
MMd03 (proportional data)	0.362	0.207	1.744	0.284	0.723
MMd01 (proportional data)	-8.453	2.31	-3.66	-0.687	0.546
MFn07 (proportional data)	-1.029	0.48	-2.14	-0.386	0.594

1. Program source code are reused at the level of function.
   For reuse results based on the reuse rate, the function reuse rate was most strongly related to the values of reusability metrics. This is considered to be because C language program source code are mostly reused at the level of function.
2. Ease of reuse can be estimated from the data at the level of the entire system.
   As the level of reusability measurement, the entire system is more suitable for the evaluation of reuse rate than individual components. This is probably because the features and tendencies that affect the reusability of projects are at the level of the entire project.
   In other words, such features and tendencies do not necessarily exist in individual files or directories of the entire system; in the experiments such features and tendencies are not concentrated in particular files and directories of the system.
3. The more limited the externally used files, the higher the reusability (MMd01).
   Multiple regression analysis and the correlation analysis shown in Figure 2 revealed that when the percentage of externally used files in the directories of the system is smaller, namely, the API and interfaces that are used externally at the level of directory are more limited, the function reuse rate at the level of the system tends to be higher.
   For example, let us consider directories Ma and Mb, as shown in Figure 4. The values of MMd01 are $3/5 = 0.6$ and $1/5 = 0.2$ for Ma and Mb, respectively. In this case, the number of externally used files in Mb is smaller, or more limited, than that in Ma; a system composed of such directories with limited entrance will be more reusable. A directory in which only one file is used by external modules, similarly to Mb, is considered to be a result of applying the Facade pattern[21], which defines an unified and higher-level interface to a set of interfaces in a subsystem.
4. The smaller the number of function calls, the higher the reusability (MFn07).
   Multiple regression analysis and the correlation analysis shown in Figure 2 also revealed that when the number of function calls in each function is smaller, namely, the function under evaluation is called more often than it makes calls, and is closer to the end of call, the function reuse rate at the level of the system tends to be higher.
   In Figure 5, for example, the values of MFn07 are 3, 1, and 0 for `fa()`, `fb()`, and `fc()`, respectively. In this case, `fa()` mainly has the role of calling other

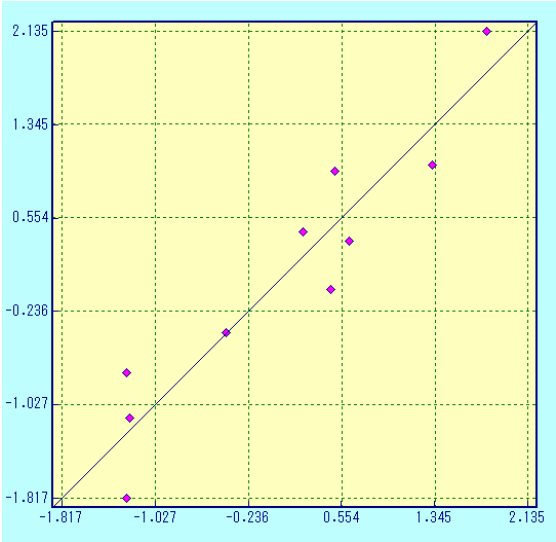

Fig. 3. Scattergram of function reuse rate at the level of the system for 10 projects after the logit transformation (X-axis, estimated values obtained using regression equation; Y-axis, metric values); multiple correlation coefficient = 0.941.

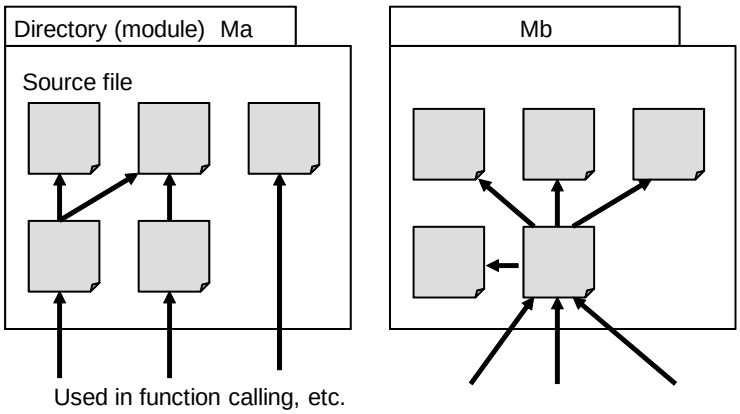

Fig. 4. Example of measuring MMd01 (percentage of externally used files)

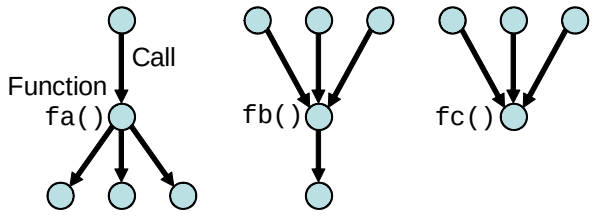

Fig. 5. Example of measuring MFn07 (number of function calls)

functions, whereas `fc()` is just called by other functions. Therefore, a system having more such `fc()` end functions in the call chain has greater reusability.

5. Ease of reuse is more significantly affected by the percentage of files used by external modules (MMd01) than by the number of function calls (MFn07). Furthermore, multiple regression analysis revealed that the percentage of files used by external modules significantly affects reusability.
   On the other hand, the effect on reusability of the number of dependent modules (MMd03) remains unclear and will be examined in the future; currently we cannot find any significant correlation between MMd03 and the function reuse rate.

As mentioned above, these evaluations were carefully carried out through several regression analyses, in which three types of detailed data were used for reuse results as the objective variable. Therefore, the validity of the proposed reusability metrics was accurately evaluated, thus resolving Problem A. In addition, the magnitude of effect on reuse results was individually analyzed and determined for three reusability metrics, thus resolving Problem B.

According to the above-mentioned goal, these validated metrics could be used for supporting development for reuse and development by reuse, such as estimating the effort necessary for reusing existing source code.

4.4 Threats to Validity

We used the reuse rate between corresponding two projects as a proxy for reusability of original program source code. We believe that the reuse rate defined in the subsection 3.2 reflects the reusability; however there might be several other factors affecting the reuse rate, such as requirement changes. It could be a threat to internal validity; in the future, we will inspect the similarity of requirements among target 20 projects.

Regarding threats to external validity, the evaluation was done on projects on embedded software development for some instruments in the same domain. We expect that the characteristics of all projects used in our experiments, such as the data on scale shown in Table 2, could help readers utilize the evaluation results. Moreover the evaluation was limited to derivative developments involving the reuse of entire architecture; in the future we will consider the generalizability of

the obtained results by applying the metrics to completely new developments involving reuse of existing components.

5 Conclusion and Future Work

For C language program source code, a set of metrics considered to be effective for measuring reusability were qualitatively identified. Through accurate analyses using several types of data on reuse results, it was statistically clarified that three of the identified metrics tend to have different effects on actual reusability. In these analyses, we defined the reuse rate, which can more accurately reflect the actual state of reuse than the frequency of reuse.

The main contribution of this paper is the development of procedures for accurately evaluating the validity and effectiveness of reusability metrics and for analyzing the magnitude of their individual effects on actual reusability. As a result, a set of reusability metrics, the effectiveness of which has been evaluated and which can be used for C language program source code, were proposed together with an evaluation of the magnitude of their individual effects on reusability.

The followings are future works.

- Further review and expand the metrics (particularly MMd03) by increasing the number of project data and repeating the analysis, and analyze the generality of the metrics. Moreover it is necessary to consider the effect of various files (such as XML files[22]) on implicit module dependencies.
- Expand the definition of the reuse rate, which is used for comparison with metric values during the evaluation of their validity, so that the reuse rate can be applied even when original existing projects do not necessarily correspond to new derivative projects on a one-on-one level (e.g., the reuse rate when a project product is reused by various new projects).
- Analyze the relationship between metric values and other reuse result data such as the frequency of reuse in a certain time frame.
- Examine the applicability of the metrics, the validity of which has been verified, to source code written in other program languages considering that the questions used to derive the reusability metrics are independent of the program language.
- The reusability metrics proposed in this paper could constitute part of a practical framework (such as [6, 7]) for measuring the internal quality including reusability.

References

1. Jeffrey S. Poulin, "Measuring Software Reuse: Principles, Practices, and Economic Models," Addison-Wesley, 1996.
2. ISO/IEC 9126-1: 2001, Information technology – Software product evaluation: Quality Characteristics and Guidelines for their use

3. Tom DeMarco, “Controlling Software Projects: Management, Measurement & Estimation,” Yourdon Press, 1982.
4. Norman Fenton, Robin Whitty and Yoshinori Iizuka, “Software Quality Assurance and Measurement: A Worldwide Perspective,” Thomson Computer Press, 1995.
5. Linda M. Laird and M. Carol Brennan, “Software Measurement and Estimation: A Practical Approach,” John Wiley & Sons, 2006.
6. Hironori Washizaki, Rieko Namiki, Tomoyuki Fukuoka, Yoko Harada and Hiroyuki Watanabe, “Practical Framework for Evaluating Quality of Program Source code,” IPSJ Journal, Vol.48, No.8, pp.2637-2650, 2007.
7. Hironori Washizaki, Rieko Namiki, Tomoyuki Fukuoka, Yoko Harada and Hiroyuki Watanabe, “A Framework for Measuring and Evaluating Program Source Code Quality”, Proc. 8th International Conference on Product-Focused Software Process Improvement (PROFES 2007), LNCS, Vol.4589, pp.284-299, 2007.
8. Ayatomo Kanno and Tadashi Yoshizawa, “Techniques for Assuring Software Quality towards 21st Century,” JUSE Press, Ltd., 1994.
9. Guttorm Sindre, Reidar Conradi and Even-Andre Karlsson, “The REBOOT Approach to Software Reuse,” Journal of Systems and Software, Vol.30, No.3, 1995.
10. William Frakes and Terry Carol “Software Reuse: Metrics and Models,” ACM Computing Surveys, Vol.28, No.2, pp.415-435, 1996.
11. Letha H. Etzkorn, William E. Hughes and Carl G. Davis, “Automated Reusability Quality Analysis of OO Legacy Software,” Information and Software Technology, Vol.43, No.5, pp.295-308, 2001.
12. Satoshi Nakajima, Shigeki Suguta and Yuji Hotta, “Evaluation of Metrics for Reuse of C++,” Object-Oriented Symposium, 1998.
13. Hironori Washizaki, Hirokazu Yamamoto and Yoshiaki Fukazawa, “A Metrics Suite for Measuring Reusability for Software Components,” Proc. of the 9th IEEE International Symposium on Software Metrics (Metrics 2003), IEEE CS, pp.211-223, 2003.
14. Masayuki Hirayama and Makoto Sato, “Evaluation of Usability of Software Components,” IPSJ Journal, Vol.45, No.6, pp.1569-1583, 2004.
15. Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita and Shinji Kusumoto, “Ranking Significance of Software Components Based on Use Relations,” IEEE Transactions on Software Engineering, Vol.31, No.3, pp.213-225, 2005.
16. Victor R. Basili and David M. Weiss, “A Methodology for Collecting Valid Software Engineering Data,” IEEE Transactions on Software Engineering, Vol.10, No.6, 1984.
17. Sallie M. Henry and Dennis G. Kafura, “Software Structure Metrics Based on Information Flow,” IEEE Transactions on Software Engineering, Vol.7, No.5, 1981.
18. Martin Shepperd and Darrel Ince, “Metrics, outlier analysis and the software design process,” Information and Software Technology, Vol.31, No.2, 1989.
19. Kilsup Lee and Sung Jong Lee, “A Quantitative Software Quality Evaluation Model for the Artifacts of Component Based Development,” Proc. 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Paralle/Distributed Computing, and 1st ACIS International Workshop on Self-Assembling Wireless Networks, 2005.
20. Winifred Diana Ashton, “The logit transformation with special reference to its uses in bioassay”, Hafner Pub. Co., 1972.
21. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software,” Addison-Wesley, 1994.
22. Siim Karus and Harald Gall: “A study of language usage evolution in open source software,” 8th Working Conference on Mining Software Repositories (MSR), 2011.