Evaluation of a Business Application Framework Using Complexity and Functionality Metrics

Hikaru Fujiwara[1], Shinji Kusumoto[1], Katsuro Inoue[1], Toshifusa Ootsubo[2] and Katsuhiko Yuura[2]

[1]Graduate School of Engineering Science, Osaka University 1-3 Machikaneyama, Toyonaka, Osaka, 560-8531, Japan Tel: +81-6-6850-6571, Fax: +81-6-6850-6574
[2]Business Solution Systems Division, Hitachi Ltd.
h-fujiwr@ics.es.osaka-u.ac.jp

Abstract. This paper experimentally evaluates the usefulness of a business application framework from a viewpoint of saving cost and quality of the software in a company. Here, we conducted two case studies. In the case studies, four kinds of applications are developed. Each of them is developed in two ways: using framework-based reuse and conventional module-based reuse. Then, we evaluate the difference among them using the several functionality and complexity metrics. As the results, the framework-based reuse would be more efficient than the module-based reuse in the company.

1. Introduction

As the size and the complexity of software increase, it becomes important to develop high-quality software cost-effectively within a specified period. In order to attain it, several software engineering techniques have been developed. Reuse is one of the most famous techniques among them.

The definition of reuse is the use of an existing software component in a new context, either elsewhere in the same system or in another system [4]. It is generally said that software reuse is improved productivity and quality, resulting in cost saving. Several research studies have demonstrated the actual benefits[2][7][9].

On the other hand, object-oriented development intends to construct the software by reusing several software components which have high cohesion and are easy to combine. By reusing the components which have high quality, the improvement of the quality of the software will be attained and development period will be also decreased [8]. As in development with object-oriented languages, developers reuse a particular library called a framework [3]. A framework is collection of classes that provide a set of services for a particular domain; a framework thus exports a number of individual classes and mechanisms which clients can use or adapt. In a typical development with the framework, at first, developers take out the primitive parts of a program. Then, they develop the other parts needed for the developing application, and combine them into a program. The Microsoft Foundation Classes (called MFC) is a framework for building applications that conform to the Microsoft Windows user interface standards.

In a department of Hitachi Ltd., a business application framework for a specific domain has been developed and introduced. However, it is difficult to transfer the new framework to the development because developers in the department use the original reuse technique that is a conventional module-based reuse. From a viewpoint of saving cost and improving the productivity, it is clear that framework-based reuse is more appropriate than the module-based reuse. For effective technical transfer, it is necessary to motivate the developers who will use the new technique by showing the benefit of using it quantitatively.

This paper experimentally evaluates the usefulness of the framework from a viewpoint of saving cost and quality of the software. Here, we conducted two case studies. In the case studies, four kinds of applications are developed. Each of them is developed in two ways: using framework-based reuse and conventional module-based reuse. Then, we evaluate the difference among them using the several functionality and complexity metrics. As the results, the framework-based reuse would be more efficient than the module-based reuse in the department.

Section 2 introduces the characteristics of the target application software and the proposed framework. Then, Section 3 describes the case studies and Section 4 analyzes the empirical results. Finally, Section 5 concludes and mentions about future research topics.

2. Preliminary

2.1 Target application software

In a department of Hitachi Ltd., a series of on-line application software for a local government has been developed. Here, “series” implies that there exist multiple projects where similar applications are successively developed for several local governments. The functions of the application software includes such as a family registration, payment of various taxes, treatment of health insurance. Since the data handled by each local government are different, the details of each applications are different, but the fundamental processing of each function is the same. To put it simply, it controls the processing of conventional database operations.

So far, to develop the series of the applications, conventional module-based reuse has been used. Figure 1 shows the typical example. Figure 1 (a) is the module structure of an application that makes a new record for a health insurance for a local government A. Module A_1 is a main module and controls the transition of the screens. Users manipulate the application through the screens that are changed based on the users' input. Each of the modules A_{11}, A_{12} and A_{13} corresponds to the process of data inquiry, showing the inquiry result and updating the record. In the conventional module-based reuse, each of the modules is reused in the next development. Figure 1 (b) shows the module structure of the same application for a local government B. Here, modules B_{11}, B_{12} and B_{13} are reused by modifying the corresponding modules A_{11}, A_{12} and A_{13}. The reused modules are modified in a skillful way in the department.

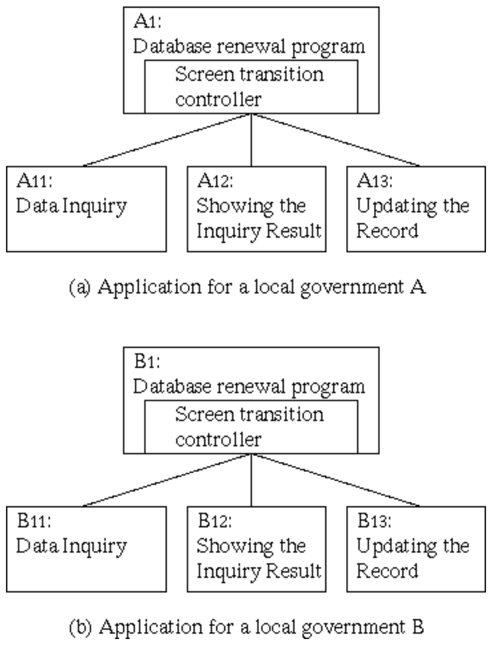
A1:
Database renewal program
Screen transition controller
A11: Data Inquiry
A12: Showing the Inquiry Result
A13: Updating the Record
(a) Application for a local government A
B1:
Database renewal program
Screen transition controller
B11: Data Inquiry
B12: Showing the Inquiry Result
B13: Updating the Record
(b) Application for a local government B

Fig. 1. Conventional module-based reuse

2.2Proposed framework

Recently, many companies have started to introduce object-oriented technology into their software development environments. The application software described in Section 2.1 will be also developed by object-oriented approach and implemented using Java. So, the new framework is going to be introduced. The framework is intended to reuse the processing of the transition of the screens in addition to the module-based reuse. In the framework, typical transitions of screens are prepared, that are data inquiry, data renewal, data addition, data removal and so on. For example, data renewal consists of the three kinds of screen manipulations: data inquiry, showing the inquiry result and updating the record.

For example, Figure 2 explains the framework-based reuse. Figure 2 (a) shows the application software that makes a new record for a health insurance for a local government A. F_1 is a framework where a screen transition pattern for data renewal is got together. By using F_1, the processing of data inquiry, showing the inquiry result and updating the record are simply implemented together. The specific information to the local government A is given by parameters. So, in case of developing the similar application for a local government B, it is easily done using the framework F_1 and the specific information to the local government B shown in Figure 2(b).

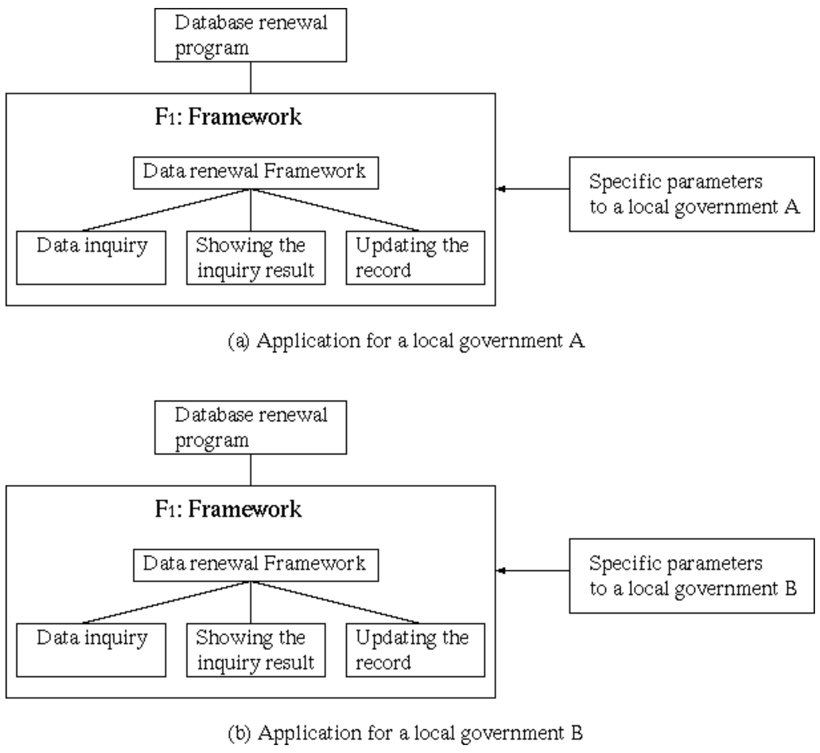
Database renewal program
F1: Framework
Data renewal Framework
Data inquiry
Showing the inquiry result
Updating the record
Specific parameters to a local government A
(a) Application for a local government A
Database renewal program
F1: Framework
Data renewal Framework
Data inquiry
Showing the inquiry result
Updating the record
Specific parameters to a local government B
(b) Application for a local government B

Fig. 2. Framework-based reuse

3. Case Studies

Here, we explain the case studies to evaluate the usefulness of the proposed framework.

3.1 Design of case study

We conducted two case studies: Case studies 1 and 2. In case study 1, four kinds of programs are developed in two ways: conventional and framework-based reuse. Here, C_a, C_b, C_c and C_d represent the four kinds of programs developed using the framework-based reuse. On the other hand, P_a, P_b, P_c and P_d represent the ones developed using the conventional reuse. The functions implemented in C_i and P_i (i=a,b,c,d) are the same. We compare C_i and P_i from the viewpoint of the productivity and quality.

In case study 2, a program is also developed in two ways. In this case, each of four functions (called f_a, f_b, f_c and f_d) are continuously added to the program. That is, using the framework reuse, at first, a program (called C_a), includes only f_a, is developed and then a program (called C_{a+b}) is developed by adding the function f_b to C_a. Similarly,

C_{a+b+c} is developed by adding f_c to C_{a+b} and finally $C_{a+b+c+d}$ is completed by adding f_d to C_{a+b+c}. On the other hand, using the conventional reuse, programs P_a, P_{a+b}, P_{a+b+c} and $P_{a+b+c+d}$ are developed. The functions between C_i and P_i (i=a, a+b, a+b+c, a+b+c+d) are the same. We compare the differences between the two successive programs from the viewpoint of the productivity and quality.

3.2 Metrics

Unfortunately, we could not collect the actual development effort and the number of faults injected in the development. So, we used the following metrics to indirectly evaluate the productivity and quality: OOFP (Object-Oriented Function Point)[5] and C&K metrics (Chidamber and Kemerer's metrics)[6].

Function point (FP) was proposed to help measure the functionality of software systems and to estimate the effort required for the software development. However, the original function point is not proposed to object-oriented program. Recently, OOFP has been proposed in [5] and relatively easy to calculate from the programs. So, we used the OOFP to indirectly measure the productivity.

On the other hand, C&K metrics is one of the most famous metrics to evaluate the complexity of object-oriented software. Several studies reported the relationship between the values of C&K metrics and the number of faults injected. For example, Basili et al. empirically evaluated that C&K metrics show to be better predictors of fault-proneness of the class than the traditional code metrics [1]. Thus, we used C&K metrics to indirectly measure the quality of the applications.

3.3 OOFP

Caldiera et al. have defined an adaptation of traditional FP, called OOFP, to enable the measurement of object oriented analysis and design specifications[5].

In traditional developments, the central concepts used in counting function points are *logical files* and *transactions* that operate on those files. In OO systems, the core concept is no longer related to file or data bases; instead the central concept is the “object”. A class is the candidate for mapping logical files into the OO paradigm. In the FP method, logical files (LFs) are divided into *internal* logical files (ILFs) and *external* interface files (EIFs). Classes within the application boundary correspond to ILFs. Classes outside the application boundary correspond to EIFs. For each ILF/EIF, it is necessary to compute the number of DETs (Data Element Types) and RETs (Record Element Types). Simple attributes, such as integers and strings, are considered as DETs. A complex attribute, such as an attribute whose type is a class or a reference to another class, is considered as RET.

Transactions in FP method are classified as *inputs*, *outputs* and *inquiries*. In OOFP, they are simply treated as generic Service Requests (SRs). Each service request (method) in each class in the system is examined. Abstract methods are not counted. Concrete methods are only counted once (in the class they are declared), even if they are inherited by several subclasses, because they are only coded once. If a method is to be counted, the data types referenced in it are classified: (1) simple items are simple data items (such as integers and strings) referenced as arguments of the method, and simple global variables referenced in the method (2) complex items are complex (class or reference to class) arguments, objects or complex global variables referenced by the method. Simple items are considered as DETs, and Complex items are considered as File Types Referenced (FTRs).

Finally, OOFP is calculated by summing up the number of ILF, EIF and SR weighted by each complexity based on DET, RET and FTR.

3.4 C&K metrics

Chidamber and Kemerer's metrics are based on measurement theory, and reflect the viewpoints of experienced object-oriented software developers. Chidamber and Kemerer defined the following six metrics[6]:

1. WMC(Weighted Method per Class) : Assume that a class C includes methods $M_1, ..., M_n$. Let $c_i(i = 1..n)$ be the static complexity of method M_i. Then, WMC(C)= $\sum_{i=1}^{n} c_i$. If it can be supposed that all methods of the given class C are equally complex, then WMC(C) is simply the number of methods defined in the class C [1].
2. DIT(Depth of Inheritance Tree of a class) : DIT(C) is the depth of class C in the inheritance tree of a given program. If it can be supposed that the whole inheritance graph is a tree, then DIT(C) is the length of path from the root to the class C [1].
3. NOC(Number Of Children) : NOC(C) is the number of immediate sub-classes subordinated to a class C in the class hierarchy of a given program.
4. CBO(Coupling Between Object class) : CBO(C) is the number of couplings between a class C and any other class. The coupling means that the class C uses member functions and/or instance variables in any other class.
5. RFC(Response For a Class) : Let $M_s(C)$ be a set of methods in a class C, and define $M_r(C) = \{M_j \mid M_j$ is a method called by any $M_i \notin M_s(C)$ and $M_j \notin M_s(C)\}$. Then RFC(C)= $| M_r(C) \cup M_s(C) |$.
6. LCOM(Lack of Cohesion in Methods) : Assume that a class C includes methods $M_1, M_2,...,M_n$. For each i (i = 1..n), let I_i be a set of instance variables used by method M_i. LCOM(C) is defined as the number of such pairs of method (M_i, M_j) that $I_i \cap I_j = \phi$, minus such pairs of method (M_k, M_l) that $I_k \cap I_l \neq \phi$, when the former is greater than the latter. Otherwise, LCOM(C) is defined to be zero.

3.5 Application of the metrics

Here, we explain how to apply the metrics to the programs. In order to evaluate the effect of the framework, we pay attention to the newly developed part of the programs. Figure 3 shows rough structure of the C_a and P_a in case study 1. In C_a, FW represents the framework. On the other hand, Figure 4 shows rough structure of the C_a, P_a, C_{a+b} and P_{a+b} in case study 2. In both cases, the values of metrics are calculated from the newly developed part of the programs (shaded portion).

Tables 1 and 2 show the measurement results of case studies 1 and 2. The values of C&K metrics are the average values per a class. Here, the values of NOC and DIT are omitted since these applications were developed without class inheritance and the values became 0.

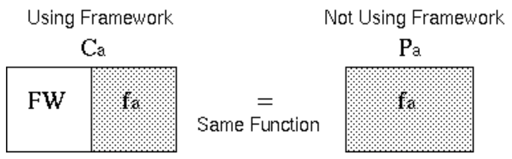

Fig. 3. Measurement in case study 1

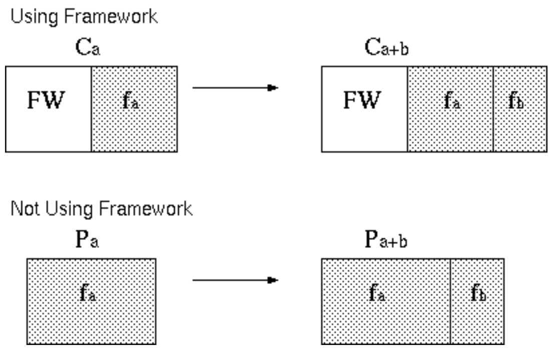

Fig. 4. Measurement in case study 2

4. Analysis

4.1 Case study 1

With respect to OOFP, the values of programs developed using framework are smaller than ones of programs developed using conventional reuse[1]. Especially, the values of C_a, C_b and C_d are much smaller than ones of corresponding programs.

Next, we analyze the values of C&K metrics:

CBO, RFC: The values of C_i are higher complexity than ones of P_i. It means that the newly developed classes of C_i include a lot of method calls. We examined the classes of C_i that have high CBO and RFC values and found that the called method are mostly included in the classes in the framework. So, if the classes of the framework have high quality, the complexity does not affect the quality of the overall application programs.

WMC, LCOM: The values of C_i are higher than ones of P_i. We also examined the classes of C_i that have high WMC and LCOM values and found that there are many simple methods, that set/get the values of attributes in the class of

[1] Since P_is have been developed scratch, they actually had no reused part.

framework. The size of these methods is about one or two lines of codes. Therefore, the complexity does not affect the quality of the overall application programs, too.

Table 1. Result in case study 1

	OOFP	CBO	RFC	WMC	LCOM
C_a	176	3.8	14.4	7.4	21.4
C_b	180	5.8	18.2	7.6	23.2
C_c	418	5.4	33.1	8.1	28.7
C_d	252	4.1	17.8	6.4	13.8
P_a	526	2.1	5.8	3.3	2.1
P_b	526	2.3	5.9	3.3	2.1
P_c	671	3.0	7.4	3.4	2.0
P_d	672	2.4	8.6	4.3	15.7

4.2 Case study 2

Using Table 2, we evaluate the amount of increased values of the metrics. Here, Δ_b, Δ_c, Δ_d represent the increased values of OOFP by adding the function f_b, f_c and f_d, respectively. For C_i (using the framework), the values of Δ_b, Δ_c and Δ_d are 75, 327 and 165, respectively. On the other hand, for P_i (not using the framework), the values of Δ_b, Δ_c and Δ_d are 89, 234 and 235. For Δ_c, the values of C_i has higher value than ones of P_i. This is because in order to implement the function f_c, a lot of data items must be handled compared to the function f_b and f_d. Also, it is found that the adjustment between the function f_c and the framework is not good. One way to cope with it is to add the new components to the framework to fit the function like fc.

Next, we analyze the values of C&K metrics. Here, (δ_b, δ_c, δ_d) represent the increased values of C&K metrics by adding the function f_b, f_c and f_d, respectively.

CBO, WMC: There is few difference between the increased values in C_i and P_i.

RFC: The increased value of δ_c in C_i is 13.4 (= 30.1 - 16.7) and much larger than one (1.7 = 8.6 - 6.9) in P_i. Using the framework, the number of method call is increased. The called method are mostly included in the classes in the framework. So, if the classes of the framework have high quality, the complexity does not affect the quality of the overall application programs.

LCOM: There are not so much difference among C_i. On the other hand, the increased value in δ_d (8.2 = 10.0 - 1.8) in P_i is larger that in δ_b and δ_c.

For $C_{a+b+c+d}$ and $P_{a+b+c+d}$, the values of OOFP are 743 and 1084. So, totally, by using the framework, the effort of the development would be reduced. On the other hand, with respect to C&K metrics, the complexity of $C_{a+b+c+d}$ is higher than $P_{a+b+c+d}$. However, the complexity is caused by the interaction among the newly developed class and the framework classes. So, we consider that if the framework has high quality, the complexity does not affect the quality of the overall application programs.

Table 2. Result in case study 2

	OOFP	CBO	RFC	WMC	LCOM
C_a	176	3.8	14.4	7.4	21.4
C_{a+b}	251	5.0	16.7	7.6	20.1
C_{a+b+c}	578	5.9	30.1	8.3	28.6
$C_{a+b+c+d}$	743	5.8	28.3	7.9	25.6
P_a	526	2.1	5.8	3.3	2.1
P_{a+b}	615	2.8	6.9	3.3	2.0
P_{a+b+c}	849	3.7	8.6	3.3	1.8
$P_{a+b++c+d}$	1084	4.0	10.5	3.9	10.0

5. Conclusion

We have experimentally evaluated the usefulness of the framework from a view point of the effectiveness of the saving cost and the quality of the software. As the results of two case studies, the framework-based reuse is more efficient than the module-based reuse in the department.

Actually, the value of FW's OOFP is 1298. However, as the result of case study 1, the value of OOFP in framework-based reuse about 2.5 times as effective as in conventional reuse. So, it is expected that the department developing the series of project will save the effort after three or four applications have developed, whereas the investment for FW was spent.

In order to show the usefulness of the framework, we are going to apply the framework to many software development projects. In addition, it is necessary to add the new components to the framework to increase the applicability of it.

References

1. V. R. Basili, L. C. Briand and W. L. Melo: *"A validation of object-oriented design metrics as quality indicators"*, IEEE Trans. on Software Eng. Vol. 20, No. 22, pp. 751-761 (1996).
2. V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: *"The software engineering laboratory - an operational software experience"*, Proc. of ICSE14, pp. 370-381 (1992).
3. G. Booch: *Object-Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing (1994).
4. C. Braun: *Reuse*, in John J. Marciniak, editor, Encyclopedia of Software Engineering, vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).
5. G. Caldiera, G. Antoniol, R. Fiutem and C. Lokan: *"Definition and experimental evaluation of function points for object-oriented systems"*, IEEE, Proc. of METRICS98, pp.167-178 (1998).
6. S. R. Chidamber and C. F. Kemerer: *"A metrics suite for object-oriented design"*, IEEE Trans. on Software Eng., Vol. 20, No. 6, pp. 476-493 (1994).
7. S. Isoda: "Experience report on a software reuse project: Its structure, activities, and statistical results", Proc. of ICSE14, pp.320-326 (1992).

8. I. Jacobson, M. Griss and P. Jacobson: *Software Reuse ---Architecture Process and Organization for Business Success*---, Addison-Wesley (1997).
9. B. Keepence and M. Mannion: *"Using patterns to model variability in product families"*, IEEE Software, Vol. 16, No. 4, pp. 102-108 (1999).