複雑度と機能量に基づく アプリケーションフレームワークの実験的評価

藤原 晃[†] 楠本真二[†] 井上克郎[†]
大坪稔房[††] 湯浦克彦[††]

本論文では，ソフトウェアの品質と工数削減の観点から，フレームワークの有効性を実験的に評価することを目的とする．ここでは2つのケーススタディを行なう．ケーススタディでは4種類のアプリケーションをＪＡＶＡで開発する．各々のアプリケーションは，フレームワークに基づく再利用と従来のモジュールに基づく再利用の２通りの方法で開発される．それらの間の差異を機能量と複雑さのメトリクスを用いて評価する．結果として，フレームワークを用いた再利用は従来型の再利用よりも効果的であることが確認された．

Evaluation of a Business Application Framework Using Complexity and Functionality Metrics

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

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. はじめに

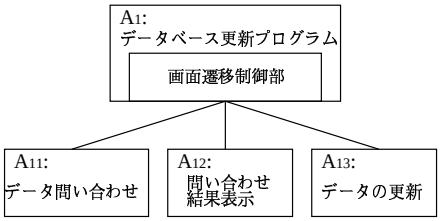
ソフトウェアの大規模化と複雑化に伴い，高品質なソフトウェアを一定期間内に効率良く開発することが重要になってきている．これを実現するために様々なソフトウェア工学技術が提案されてきている．再利用はそれらの中でも最も有効なものの一つである．

再利用は既存のソフトウェア部品を同一システム内や他のシステムで用いることであると定義されている[4]．一般にソフトウェアの再利用は生産性と品質を改善し，結果としてコスト削減するといわれている．実際の効果を報告した論文も多く発表されている[2)7)9)]．

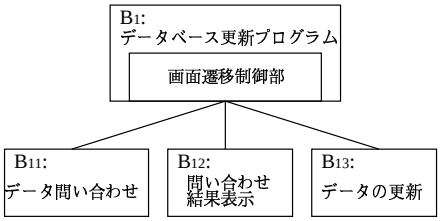
一方，オブジェクト指向開発では，凝集度が高く合成しやすいソフトウェア部品を再利用することでソフトウェアを効率よく開発することを目的としている．品質の高い部品を再利用することでソフトウェアの品質を向上することができ，開発期間も短くなると指摘されている[8]．オブジェクト指向言語での開発において開発者はフレームワークと呼ばれる特定のライブラリを再利用する．フレームワークは特定のドメインに対するサービスを提供するクラスの集合である．フレームワークを用いた典型的な開発では，まず，開発者はプログラムの基本的な部品を取り出す．そして，アプリケーションを開発するのに必要な他の部分を作成し，それらを合成して一つのプログラムにする．例えば，The Microsoft Fundation Classes (MFC) は Microsoft Windows のユーザーインターフェイス基準に準拠したアプリケーションを開発するためのフレームワークである．

日立製作所のある部署では，特定ドメイン向けのビジネスアプリケーションフレームワークを開発し，導入しようとしている．しかし，その部署では従来型のモジュール単位での再利用手法を長期間使っており，

† 大阪大学大学院基礎工学研究科
Graduate School of Engineering Science, Osaka University
†† 株式会社日立製作所ビジネスソリューション事業部
Business Solution Systems Division, Hitachi Ltd.

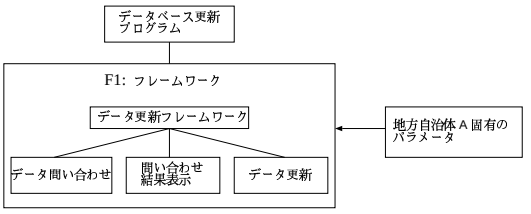


(a) 地方自治体A向けアプリケーション

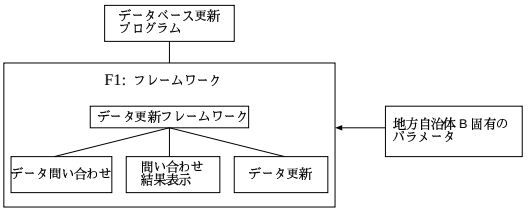


(b) 地方自治体B向けアプリケーション

図 1　モジュール単位での再利用



(a)地方自治体A向けアプリケーション



(b)地方自治体B向けアプリケーション

図 2　フレームワークを用いた再利用

新しいフレームワークを開発現場に導入するのは困難な状況にある．コストの削減と生産性の向上の観点から，フレームワークを用いた再利用が従来の再利用よりも有効であることは明らかであるが，その効果を定量的に示して導入の動機づけをすることが重要になっている．

そこで，本論文では，ソフトウェアの品質と工数削減の観点から，フレームワークの有効性を実験的に評価することを目的とする．ここでは 2 つのケーススタディを行なう．ケーススタディでは 4 種類のアプリケーションをＪＡＶＡで開発する．各々のアプリケーションは，フレームワークに基づく再利用と従来の再利用の 2 通りの方法で開発される．それらの間の差異を機能量と複雑さのメトリクスを用いて評価する．結果として，フレームワークを用いた再利用は従来型の再利用よりも効果的であることが確認された．

2. では開発対象となるアプリケーションソフトウェアと導入されたフレームワークの特徴を紹介する．3. ではケーススタディについて述べ，4. では実験の結果を分析する．最後に 5. で結論を述べ，今後の課題についてまとめる．

2. 準　　備

2.1　開発対象アプリケーション

日立製作所のある部署では，様々な地方自治体向けのオンラインアプリケーションソフトウェアを開発している．ここでは，同一のアプリケーションがいくつかの地方自治体に対して継続して開発される．アプリケーションソフトウェアの機能には，住民票の登録，税金の支払，健康保険の取り扱いなどがある．各地方自治体によって扱うデータが異なるので，各アプリケーションの細かい部分は異なっている．しかし各機能について基本的な処理は同じである．簡単にいえば，典型的なデータベース操作の処理を行う機能である．

従来，それらのアプリケーションの開発においては，モジュール単位の再利用が用いられてきた．図 1 は典型的な例を示している．図 1(a) は地方自治体 A の健康保険のデータを更新するアプリケーションのモジュール構造図である．モジュール A_1 はメインモジュールで画面遷移の制御を行う．ユーザは入力データに応じて切り替わる画面を通してアプリケーションを操作する．各モジュール A_{11}，A_{12}，A_{13} はデータ照会，照会結果の表示，データ更新処理にそれぞれ対応する．従来のモジュール単位の再利用では，各モジュールは次の開発で再利用される．図 1(b) は地方自治体 B 向けの同じアプリケーションの構造を表している．ここで，モジュール B_{11}，B_{12}，B_{13} は，それぞれ A_{11}，A_{12}，A_{13} に対応するモジュールを変更して再利用したものである．

2.2　導入するフレームワーク

近年，多くの企業でソフトウェア開発環境にオブジェクト指向技術が導入され始めている．2.1 で述べたアプリケーションもまたオブジェクト指向法で開発し，ＪＡＶＡで実装することが検討されており，新しいフレームワークが導入されようとしている．そのフレームワークはモジュール型の再利用に加えて画面遷移処理も再利用することを目的としている．フレームワー

クでは，データ照会，データ更新，データ追加，データ削除など，典型的な処理についての部品が用意されている．例えば，データ更新は，データの照会，照会結果の表示，データの更新の 3 種類の処理を行う画面から構成される．

図 2 はフレームワークを用いた再利用の一例を示している．図 2(a) は地方自治体 A の健康保険データを更新するアプリケーションの構造を示している．F_1 はデータ更新の画面遷移パターンを合わせ持っているフレームワークを表している．F_1 を用いることにより，データの照会，照会結果の表示，データの更新は簡単に実装される．地方自治体 A に固有の情報はパラメータとして与えられる．それゆえ同様のアプリケーションを地方自治体 B に対して開発する場合，フレームワーク F_1 と地方自治体 B に固有の情報を用いて簡単に開発することができる (図 2(b) 参照)．

3. ケーススタディ

ここでは，導入予定のフレームワークの有効性を評価するために行ったケーススタディについて述べる．

3.1 概　　要

ケーススタディ 1 と 2 の 2 つを行った．ケーススタディ 1 では，4 種類のプログラムを，従来のモジュール単位の再利用とフレームワークを用いた再利用の 2 通りの方法で開発した．C_a, C_b, C_c, C_d はフレームワークを用いた再利用で開発された 4 種類のプログラムを表す．一方，P_a, P_b, P_c, P_d は従来の再利用を用いたプログラムをあらわす．C_i と P_i(i=a,b,c,d) は同じ機能である．

ケーススタディ 2 では一つのプログラムが 2 通りの方法で開発される．この場合，4 つの機能 f_a, f_b, f_c, f_d は連続してプログラムに追加される．つまり，フレームワークを用いた場合は，まずプログラム f_a の機能を持つプログラム C_a を開発し，そして機能 f_b を Ca に追加することで C_{a+b} を開発する．同様に C_{a+b} に f_c を追加することで C_{a+b+c} が開発され，最後に C_{a+b+c} に f_d を追加して $C_{a+b+c+d}$ が完成する．一方，従来の再利用手法を用いて $P_a, P_{a+b}, P_{a+b+c}, P_{a+b+c+d}$ が開発される．C_i,P_i は同じ機能である．

3.2 メトリクス

残念ながら各ケーススタディにおいて実際の開発工数と開発中に見つかったバグ数を収集できなかった．そこで，OOFP(Object-Oriented Function Point) と C&K メトリクス (Chidamber と Kemerer の複雑さメトリクス) を，それぞれ生産性と品質を評価するために用いた

ファンクションポイント (FP) はソフトウェアシステムの機能量を計測し，ソフトウェア開発の工数の見積り等に利用するために提案された．しかし，元々の FP はオブジェクト指向プログラムに対して提案されたものではない．近年，OOFP が提案され[5]，これは比較的容易にプログラムから計測が可能である．そこで，OOFP を生産性を計測するために用いた．

一方，C&K メトリクスはオブジェクト指向ソフトウェアの複雑さを計測するもっとも有名なメトリクスのうちの一つである．C&K メトリクスと発見されたバグ数との関係についての研究も報告されている．たとえば Basili らは実験的に C&K メトリクスがクラスのバグの出やすさを，従来のコードのメトリクスよりよく予測していることを評価した[1]．それゆえ C&K メトリクスをアプリケーションの品質を計測するために用いた．

3.3 OOFP

Caldiera らは従来の FP をオブジェクト指向で開発された要求/設計仕様書から計測できるようにカスタマイズし，OOFP として定義した[5]．

従来，FP 計測における中心概念は論理ファイルとそれらのファイルを扱うトランザクションであった．オブジェクト指向システムでは核となる概念はファイルやデータベースとは直接関係がなく，オブジェクトが中心となる．クラスは論理ファイルをオブジェクト指向ソフトウェア上に対応させたものと考えられる．FP 法では論理ファイルは内部論理ファイル (ILF) と外部インターフェイスファイル (EIF) に分類される．OOFP では，アプリケーションの境界内にあるクラスを ILF に対応させ，アプリケーションの境界外部にあるクラスを EIF に対応させている．ILF，EIF それぞれについてデータ項目数 (DET) とレコード種類数 (RET) を算出する必要がある．整数やストリングなど単純な属性は DET とみなされ，クラス型やクラスへの参照型の属性のように複雑な属性は RET とみなされる．

FP 法におけるトランザクションは外部入力，外部出力，外部照会の 3 つに分類される．OOFP では，それらを単に一般的なサービスの要求 (SR : Service Request) として扱う．システム内のクラスのメソッドを SR として数えるが，抽象メソッドは数えない．具象メソッドは，サブクラスに継承されていても，(宣言されているクラス内で) 1 度だけ数えられる．メソッドが数えられたら，その参照するデータ型を単純な型と複雑な型の 2 種類に分類する．単純な型は引数やグローバル変数として参照している整数やストリング型を表

し，複雑な型はそのメソッドによって参照されている複雑な（クラス型やクラスへの参照型の）引数，複雑なグローバル変数，オブジェクトを表す．単純な型はDET とみなされ，複雑な型は関連ファイル数 (FTR) とみなされる．

最後に，各 ILF，EIF，SR を DET，RET，FTR に基づく複雑さで重みづけをして合計したものが OOFP として算出される．

3.4 C&K メトリクス

Chidamber と Kemerer らのメトリクスはオブジェクト指向ソフトウェアのクラス複雑度を，内部，継承，結合の 3 つの視点から評価するものであり，以下の 6 種類のメトリクスから構成される．

WMC(クラスの重み付きメソッド数; Weighted Methods per Class): 計測対象クラス C が，メソッド $M_1, ..., M_n$ を持つとする．これらのメソッドの複雑さをそれぞれ $c_1, ..., c_n$ とする．このとき，$WMC(C) = \Sigma c_i$ である．適切な間隔尺度 f を選択して $c_i = f(M_i)$ によりメソッドを重み付けをする．Basili ら[1] および Chidamber ら[6] の研究においては，すべてのメソッドの複雑さが同じであるという仮定をおいて，WMC をメソッドの数としている．本論文でも同じ仮定を用いる．

DIT(継承木における深さ; Depth of Inheritance Tree): DIT は計測対象クラスの継承の深さである．多重継承が許される場合は，DIT は継承木におけるそのクラスを表す節点から根に至る最長パスの長さとなる．

NOC(子クラスの数; Number Of Children): NOC は計測対象クラスから直接導出されているサブクラスの数である．

CBO(クラス間の結合; Coupling Between Object classes): CBO は，計測対象クラスが結合しているクラスの数である．あるクラスが他のクラスのメソッドやインスタンス変数を参照しているとき，結合しているという．

RFC(クラスの反応; Response For a Class): 計測対象のクラスのメソッドと，それらのメソッドから呼び出されるメソッドの数の和として定義される (すなわち，メッセージに反応して潜在的に実行されるメッセージの数である)．

LCOM(メソッドの凝集の欠如; Lack of COhesion in Methods): 計測対象クラス C が n 個のメソッド $M_1, ..., M_n$ を持つとする．I_i $(i = 1, ..., n)$ を，それぞれメソッド M_i によって用いられるインスタンス変数の集合とする．$P = \{(I_i, I_j)|I_i \cap I_j = \emptyset\}$ と定義し，$Q = \{(I_i, I_j)|I_i \cap I_j \neq \emptyset\}$ と定義する．もし $I_1, ..., I_n$ がすべて $\emptyset$ の時は，$P = \emptyset$ とする．このとき，$LCOM(C) = |P| - |Q|$，ただし，$|P| - |Q| < 0$ の時は，$LCOM(C) = 0$ と定義する．

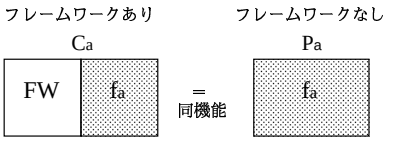


図 3 ケーススタディ 1 の計測方法

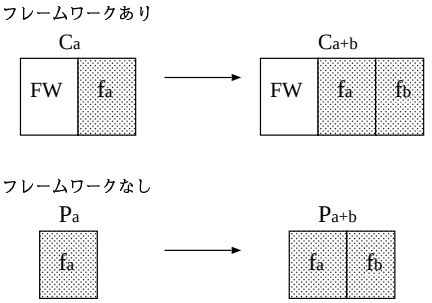


図 4 ケーススタディ 2 の計測方法

3.5 計 測 方 法

ここでは，上述したメトリクスをプログラムにどのように適用するかを説明する．フレームワークの効果を評価するために，新規開発部分に注目する．図 3 はケーススタディ 1 における C_a, P_a の構造を示している．C_a では FW はフレームワークを表している．一方，図 4 はケーススタディ 2 における $C_a, P_a, C_{a+b}, P_{a+b}$ の構造を示している．どちらのケースにおいても，メトリクスの値は新規にプログラムの開発された部分（影のついた部分）から算出する．

表 1 と 2 はそれぞれケーススタディ 1 と 2 の計測結果を表す．C&K メトリクスの値は 1 クラス当たりの平均である．ここで，NOC と DIT については値が 0 になったため省略する．これは，これらのアプリケーションがクラスの継承を用いていないためである．

4. 分　　析

4.1 ケーススタディ 1

OOFP については，フレームワークを用いて開発したプログラムの値のほうが従来の再利用手法を用いたプログラムの値に比べて小さくなっている．(P_i は新規に開発されており，再利用部分はない) とくに C_a, C_b, C_d の値は対応するプログラムの値より非常に小さくなっている．

表 1 ケーススタディ 1 の計測結果
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

つぎに C&K メトリクスの値を分析する．

CBO,RFC: C_i の値は P_i の値より高い複雑度を示している．これは C_i の新規開発クラスが多くのメソッド呼び出しを含んでいることを示している．フレームワーク内のメソッドに対する呼び出しを除いて CBO を計測すると，C_i のすべてのクラスで CBO は 0 となった．このことから，C_i のクラスのメソッド呼び出しは，すべてフレームワーク内のクラスのメソッドに対して行なわれていることがわかる．従って，フレームワーク内のクラスの品質が高ければ，結合による複雑さは一見高いがアプリケーションプログラム全体の品質に影響はないと考えられる．

WMC,LCOM: C_i の値は P_i の値よりも高くなっている．C_i のクラスのうち WMC，LCOM が平均より高いものについて調べたところ，クラスの属性を設定，参照するための set/get メソッドが約 45%を占めていた．set/get メソッドの処理は 1，2 行であった．WMC の定義から，set/get メソッド数に比例して WMC は増加する．ある属性の set/get メソッドは，一般に他の属性の set/get メソッドと参照する属性に共通するものがないため，LCOM の定義から，set/get メソッドの組合せ数に比例して LCOM は増加する．set/get メソッドが多いために WMC，LCOM が高くなっているが，set/get メソッドが単純であることからアプリケーション全体の品質には影響しないと思われる．

4.2 Case study 2

表 2 を用いて，メトリクスの増加量を評価する．ここで，$\Delta_b, \Delta_c, \Delta_d$ はそれぞれ 機能 f_b, f_c, f_d, を追加した時の OOFP の増加量を表す．C_i（フレームワーク使用）では，$\Delta_b, \Delta_c, \Delta_d$ の値はそれぞれ 75, 327, 165 であった．一方，P_i(フレームワークなし) では，$\Delta_b, \Delta_c, \Delta_d$ はそれぞれ 89,234,235 であった．Δ_c では，C_i の方が P_i の値より高かった．これは機能 f_c を実装するために f_b,f_d よりも多くのデータ項目が処理されるためである．また，機能 f_c とフレームワークとの適合性が良くないこともわかった．従って，f_c のような機能に合わせるために新たなコンポーネントをフレームワークに追加する必要がある．

次に C&K メトリクスについて分析する．ここで，$\delta_b, \delta_c, \delta_d$ はそれぞれ機能 f_b, f_c, f_d, を追加した時の C&K メトリクスの増加量を示す．

CBO,WMC: C_i,P_i の間で大きな違いは見られない．

RFC: C_i において， δ_c は 13.4(= 30.1 - 16.7) であり，P_i の値 (1.7 = 8.6 - 6.9) よりもかなり高い．フレームワークを用いると，メソッド呼び出しの数は増加する．呼ばれているメソッドはすべてフレームワークに含まれるクラスのものである．従って，フレームワークが高品質であれば，アプリケーションプログラム全体の品質に影響はない．

LCOM: C_i においては特に差はない．一方 P_i では δ_d (8.2 = 10.0 - 1.8) が δ_b や δ_c に比べて大きくなっている．

最後に，$C_{a+b+c+d}$ と $P_{a+b+c+d}$ について比較する．OOFP はそれぞれ 743, 1084 である．従って，全体ではフレームワークを用いることによって，開発工数は削減されたと考えられる．一方，C&K メトリクスについては， $C_{a+b+c+d}$ の複雑度が $P_{a+b+c+d}$ よりも高くなっている．しかし，その複雑度はフレームワークのクラスと新規開発クラスとのメッセージの送受信によるものである．従って，フレームワークが高品質であれば複雑度はアプリケーションプログラム全体の品質に影響はしないと考えられる．

5. おわりに

本論文では，工数削減の効率とソフトウェアの品質という観点からフレームワークの有効性を実験的に評価した．2 つのケーススタディの結論として，フレームワーク型の再利用の方がモジュール型の再利用よりも効果的であることが示された．

実際には FW 開発のための工数が必要であり，OOFP は 1298 であった．しかし，ケーススタディ 1 の結果からフレームワーク型の再利用での OOFP 値は従来の再利用より 2.5 倍ほど効果的であるといえる．従ってこの一連のプロジェクトを開発している部署では 3 つか 4 つのアプリケーションを開発すれば，FW への投資は解消できるものと考えられる．

今後の課題としては，フレームワークの有効性を示すために，多くのソフトウェア開発プロジェクトにフレームワークを適用していく．さらに，フレームワー

表 2 ケーススタディ 2 の計測結果
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

クの適用範囲を広げるために新しいコンポーネントを追加する必要がある.

参 考 文 献

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).