社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS96-34 (1996-11)

修正保守に対するオブジェクト指向設計の影響について

高田眞吾† 高田義広† Walcelio Melo‡ 鳥居宏次†

† 奈良先端科学技術大学院大学 情報科学研究科
〒630-01 奈良県 生駒市 高山町 8916-5
e-mail: michigan@is.aist-nara.ac.jp
‡ Centre de Recherche Informatique de Montreal

あらまし
近年，オブジェクト指向が着実にソフトウェア開発の現場に利用されるようになってきている．このため，今後オブジェクト指向ソフトウェアの保守の重要性が高まる．その保守に影響を与える要因の一つとして，プログラムがどのように設計されたかということが考えられる．そこで，オブジェクト指向設計が，いろいろな保守作業のうちの修正保守に対して与える影響を調べるために実験を行なった．本論文では，その実験について報告する．オブジェクト指向設計を計るメトリクスとしてChidamberとKemererの提案した六つを用いた結果，CBOだけが労力と統計上相関があると認められた．最後に，実験方法および解析方法についても検証する．

キーワード オブジェクト指向，メトリクス，修正保守，労力

Impact of Object-Oriented Design on Corrective Maintenance

Shingo Takada† Yoshihiro Takada† Walcelio Melo‡ Koji Torii†

† Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma-shi, Nara-ken 630-01
e-mail: michigan@is.aist-nara.ac.jp
‡ Centre de Recherche Informatique de Montreal

Abstract
With the emergence of object-oriented technology and its use in software development, the importance of maintenance of object-oriented software has grown. One aspect that may affect the cost associated with maintenance would be the design of the object-oriented program. In this paper, we report on an experiment testing the impact object-oriented design has on effort needed in corrective maintenance. We use Chidamber and Kemerer's suite of metrics, and the results show that only CBO show correlation with effort. We also inspect the experiment itself and the analysis method.

Keywords object-oriented, metrics, corrective maintenance, effort

1 はじめに

オブジェクト指向は，1980年代後半から広まり，現在様々な分野でオブジェクト指向のソフトウェアが作成されている [2] [6]．ソフトウェアのライフサイクルにおいて，ほとんどのコストは保守にかかる [13] と言われている．このことと，オブジェクト指向では，通常の手続き的手法によるソフトウェアと違った新しい保守の問題が報告されている [10] ことを合わせると，オブジェクト指向における保守の問題は非常に重要であると言える．

ソフトウェアの保守のコストに対して影響する要因として，そのソフトウェアがどのような構造をもっているか，またはどのような設計になっているかが考えられる．そこで，設計がソフトウェアの保守に対してどのような影響を与えるかを調べるために実験を行ない，本論文ではそれを報告する．保守の作業としては，エラーの訂正，機能拡張などがある [13] が，本実験では，エラーの訂正，つまり修正保守に注目した．

オブジェクト指向設計を計るために，種々のメトリクスが提案されている [1][5][11][15] が，本論文では，Chidamber と Kemerer の提案したメトリクス [5] を用いる．

本論文では，まず，2節でオブジェクト指向設計に関するメトリクスについて述べ，実験における仮説をあげる．次に，3節では，実験の手順を述べる．4節では，実験の結果・解析を記し，5節でそれに対して考察を行なう．6節では本論文をまとめる．

2 オブジェクト指向設計メトリクス

オブジェクト指向に関するメトリクスとして，様々なものが提案されている [1][5][11][15] が，ここでは，Chidamber と Kemerer により提案されたメトリクス (以下，C&K メトリクスと呼ぶ)[5] を利用する．

2.1 C&K メトリクス

Chidamber と Kemerer が提案したメトリクスは，以下の六つである．

(1) クラス毎の重み付きメソッド (Weighted Methods per Class; WMC)

(2) 継承木の深さ (Depth of Inheritance Tree; DIT)

(3) 子供の数 (Number of Children; NOC)

(4) オブジェクトクラス間のカプリング (Coupling between Object Classes; CBO)

(5) クラスに対するレスポンス (Response For a Class; RFC)

(6) メソッドにおけるコヒージョンのなさ (Lack of Cohesion in Methods; LCOM)

WMCは，メソッドの複雑度を利用することにより，クラスの複雑度を得る．各メソッドが同じ複雑度であると仮定すれば，WMCはメソッドの数に置き換えることができる．メソッドの複雑度を得る方法が定義されていないため，本研究では，各メソッドの複雑度が同じであると仮定し，WMCをクラス中にあるすべてのメンバ関数の数として扱う．

DITは，クラスの継承木における最大の深さを指す．本研究の場合，DITをクラスの先祖の数として扱う．

NOCは，クラスの直接の子供 (サブクラス) の数を指す．

CBOは，あるクラスがカップルしている他のクラスの数である．ここで，あるクラスが別のクラスのメンバ関数やインスタンス変数を利用している時，そのクラスは別のクラスとカップルしているという．

RFCは，あるクラスのオブジェクトが受けとったメッセージに対して，実行される可能性のあるメソッドの数である．本研究では，あるクラスのメンバ関数から呼ばれる関数の数として定義する．

LCOMは，次の式で得られる．

$$LCOM = P - Q$$

ここで，Pはインスタンス変数を共有していないメンバ関数の組の数であり，Qはインスタンス変数を共有しているメンバ関数の組の数である．値が負となった場合，LCOMは0として扱う．しかし，LCOMの定義は不十分であると指摘されている [4][7] ため，本研究では，対象外とする．

2.2 実験の仮説

本実験では，オブジェクト指向設計メトリクスが修正保守にかかる労力と関係があるかを検証するために，LCOMを除いた上記のC&Kメトリクスそれぞれに対して，以下の仮説を立てる．なお，「労力」は，「訂正するためにかかる時間」のことを指す．

H-WMC メンバ関数の多いクラスは，少ないクラスに比べて複雑であり，その結果として，そのようなクラスに関わるエラーを訂正するために労力が余計にかかる．

H-DIT クラス階層上，より深い位置にあるクラスは，上位クラスから継承されるメンバ関数やインスタンス変数が多くなるため，修正が難しくなり，労力が増える．

H-NOC 子供の多いクラスを訂正する場合，その訂正が子供に対しても影響を与える可能性があるため，より難しくなり，テストも多く必要となる結果，労力が増える．

H-CBO CBOの高いクラスにあるエラーは，エラーを特定し，他のクラスに対して修正が与える影響を知ることが難しいため，労力が余計にかかる．

H-RFC RFCの高いクラスは，複雑な機能を実装していることとなるため，理解・変更が難しくなり，労力が増える．

3 実験について

実験では，被験者が，C++で書かれたレンタルビデオシステム中のエラーを直した．被験者は，症状を一つずつもらい，エラーを直す過程において，種々の時間を記録した．

3.1 レンタルビデオシステム

レンタルビデオシステムは，Maryland大学でのプロジェクトの結果であり [3]，オブジェクト指向により設計され，C++で書かれていた．インタフェース作成のためにOSF/Motif，およびMotifAppというダイアログやメニューなどの作成用のクラスライブラリ [16] を利用していた．

プログラムの行数は，コメントと空行を含めて，10,145行である．クラス数は25であり，そのうちのほとんどはMotifAppライブラリ中のクラスのサブクラスであった．

3.2 被験者

奈良先端大の選択科目の中で行なわれた実験には，9名の学生が参加した．C++やOSF/Motifに対して，事前に経験を必要としなかったため，被験者の経験レベルは，初心者から上級者まで，幅広いものであった．表1は，各被験者のC++と他の最も経験のあるプログラミング言語を，その言語で書いた最も大きいプログラムの行数で経験レベルをまとめたものである．OSF/Motifに関しては，被験者Bだけが経験を持ち，他は全員初心者であった．

表 1: 被験者のプログラミング経験

被験者	C++	他言語
A	若干 (<100 行)	C (>3000 行)
B	多 (>10,000 行)	C (>10,000 行)
C	少 (>500 行)	C (>1000 行)
D	無	C (>10,000 行)
E	無	C (<100 行)
F	若干 (<100 行)	C (<100 行)
G	若干 (<100 行)	Basic (>500 行)
H	無	C (<100 行)
I	無	Basic (>500 行)

経験の少ない被験者のために，C++, OSF/ Motif, MotifAppについて，事前に講義と演習を行なった．ほとんどの被験者は，自分の経験レベルに関係なく，この講義と演習に参加した．

3.3 エラー

エラーは，合計32個あった．プログラム使用中に現れた誤った現象 (症状) 一つずつをエラーと数えた．ほとんどのエラーは独立していたが，中には，順番を考慮して直さなければならないものもあった．

学習は，ソフトウェア工学の実験の結果の正当性に対して悪影響があるため，避けるべきであると通常考えられている [9][12]．同じプログラムから複数のエラーを訂正する場合，どうしても学習が起きることを避けることができない．しかし，全体的に，各エラーが同じ程度の学習の影響を受けることになればよいと考え，順番はランダムに決めた．

さらに，エラーにはいろいろな種類があり，ある種類が後半に集中すると，それらが特別に学習を受けることになる．また，似たエラーが存在するため，それらの扱いも注意を要する．そこで，エラーを五つのグループに分け (表2)，エラーの順番を決めた．グルーピングは，著者の一人が事前に訂正した結果

表 2: 訂正のために必要な作業の種類

I	1-3 文字の訂正またはコメントアウトが必要
II	複数のメンバ関数で変更が必要
III	新しいメンバ関数が必要
IV	新しいクラスが必要
V	その他 (*if*ブロックの追加など)

症状 11： 一文字のビデオの登録

1 症状について
タイトルが一文字であるビデオを登録できない．登録できるようにすること．
2 日時
2.1 実行日
2.2 開始時刻　(症状確認時刻　)
2.3 訂正箇所発見時刻
2.4 チェック開始時刻
2.5 チェック終了時刻
3 変更箇所 (ファイル名や行なった変更)

図 1: エラーシートの例

に基づいて行った．似たエラーが順番上かたまらないように，訂正の順番は，連続した五つのエラーをとったら，基本的に各グループから一つずつあるように決めた．ただし，各グループにあるエラーの数の違いにより，常にそのようになるわけではない．

3.4 実験の手順

C++などの講義・演習後，実験の手順に慣れるために，予備実験を行なった．この後，被験者には，プロジェクトの概要，要求仕様，OMT 図 [14] を渡した．そして，システムの実行ファイルを渡し，システムが行なえる様々な機能に慣れてもらった．

この後，システムのソースコードを渡し，各被験者は次の手順でエラーを訂正した．

(1) 実験者から，訂正すべきエラーの症状等が書いてあるエラーシート (図 1) をもらう．

(2) 実行日とシートを渡された時間 (開始時刻) を記す．

(3) エラーの症状を復元して確認し，その時間を記録する (症状確認時刻)．

(4) エラーの訂正を始める．訂正すべき箇所を見つけたと思ったら，その時間を記す (訂正箇所発見時刻)．また，変更も記す．

(5) エラーを訂正できたと思ったら，その時間を記し (チェック開始時刻)，実験者に正しいかどうかチェックしてもらう．正しければ，時間を記し (チェック終了時刻)，新しいエラーシートを渡す (2 へ)．正しくなければ，いったんその時間をチェック終了時刻に記し，訂正を再開する．この間，該当すれば，随時訂正箇所発見時刻等を記す．

この作業は，被験者が全エラーを訂正したか，30 時間作業し，終了を宣言する[1]まで続けた．訂正は，クラスの時間内でのみ行なうようにし，時間外ではソースコードを見ないように注意した．

各被験者の環境は同じで，DecAlpha ワークステーションを用いた．また，コンパイラとして g++，デバッガとして gdb を利用した．

経験の少ない被験者がいたため，本実験では，「ギブアップ」を宣言することができた．もし被験者がある一定の時間内にエラーを訂正することができなかった場合，被験者はそのエラーをあきらめて，新しいエラーを始める権利があった．ギブアップするための時間は，初日は 4 時間で，二日目以降は 3 時間であった．

4 実験結果

本節では，実験結果を報告する．まず，用いたプログラムにおけるオブジェクト指向設計メトリクスの分布を示す．次に，メトリクス間の相関関係をあげ，最後に，メトリクスと労力の間の相関関係について述べる．

4.1 オブジェクト指向設計メトリクスの分布

図 2, 3, 4, 5にそれぞれ WMC, DIT, CBO, RFC に基づいた分布を示す．

[1]選択科目の単位を取得するために，最低 30 時間作業しなければならないルールを導入した．

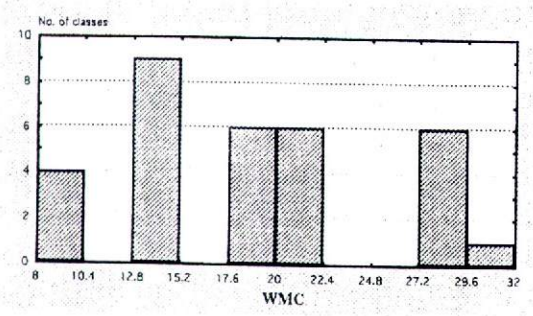

図 2: WMC による分布

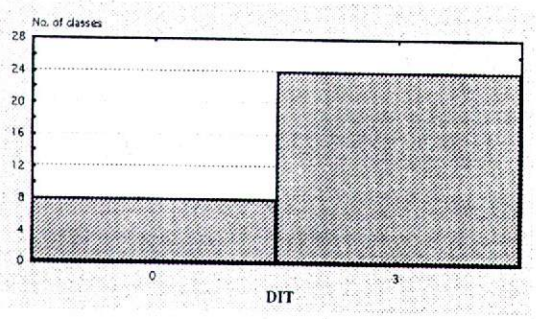

図 3: DIT による分布

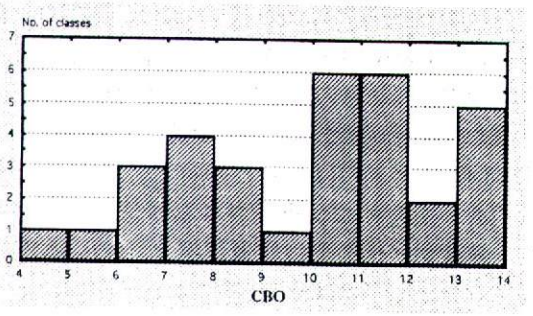

図 4: CBO による分布

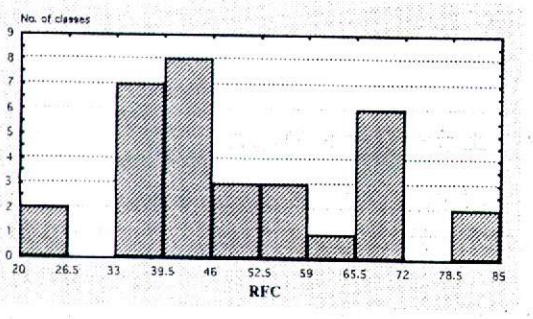

図 5: RFC による分布

表 3: C&K メトリクス間のスピアマンの順位相関係数 (有意水準 0.05)

C&K メトリクス	相関係数	検定結果
WMC と DIT	0.25	相関なし
WMC と CBO	0.31	相関なし
WMC と RFC	0.32	相関なし
DIT と CBO	0.23	相関なし
DIT と RFC	0.32	相関なし
CBO と RFC	0.42	相関あり

WMC に関しては，最低 9 のクラスから最高 32 のクラスまでと非常に幅広かった．DIT に関しては，すべてのクラスにおいて 0 または 3 であった．DIT=3 のクラスに関しては，MotifApp にあるクラスのサブクラスとして定義されたものである．CBO と RFC についても，WMC と同様に幅広い値をとっていた．いずれの結果も，正規分布に従っているわけではないので，以後の相関についての解析は，分布の正規性を仮定しないでもよいスピアマンの順位相関係数に基づく無相関検定 [8] を用いて行なう．

なお，NOC に関しては，子供をもったクラスがなかった．このため，以後の解析には NOC を含めない．

4.2 オブジェクト指向設計メトリクス間の相関関係

表 3に C&K メトリクス間のスピアマンの順位相関係数を示す．この係数を用いて，無相関検定を行なった結果，CBO と RFC の間だけが有意水準 0.05 で棄却された．つまり，CBO と RFC の間だけに相関が認められ，他のメトリクスの間では，特に相関があったわけではなかった．

4.3 C&K メトリクスと修正保守の労力の関係

実験で用いたシステムのオブジェクト指向設計と，修正保守に要した労力の間に相関があるか調べる．オブジェクト指向設計を計るために，C&K メトリクスのうちの CBO, RFC, WMC, と DIT を用いる．前述の通り，LCOM については定義に問題があり，また，NOC に関してはすべて 0 のものだけであったので，この二つに関しては，省略する．修正保守に要した労力に関しては，エラーを訂正する

表 4: C&K メトリクスと労力の間のスピアマンの順位相関係数

C&K メトリクス	相関係数	検定結果
WMC	0.27	相関なし
DIT	-0.03	相関なし
CBO	0.46	相関あり
RFC	0.23	相関なし

ためにかかった時間 (症状確認時刻からチェック開始時刻までの時間) を用いる.

表4に, C&K メトリクスと労力の間のスピアマンの順位相関係数を示す. これに基づいて, 無相関検定を行なった結果, CBO だけが有意水準 0.01 で棄却された. つまり, CBO と労力の間に相関が認められた.

これより, 2.2節であげた六つの仮説のうち, H-CBO に関しては, 実験で正しいという結論を得た. ただし, 相関係数は 0.46 とそれほど高くないため, 強い相関であると言うわけではない.

5 考察

前節で述べた通り, C&K メトリクスのうち, CBO だけが労力との間で相関が見られ, 他のメトリクスの間で相関が見られなかった. 本節では、これを一般論として結論付けられるかどうかを考察する。

具体的には, 実験または解析の方法に問題がないか検証し, 特に, 以下の事柄に注目する.

(1) 被験者の実力

(2) プログラムとエラーの選定

(3) エラーシートの記入

(4) 修正すべきクラスに到達するまでの労力

(5) エラー毎にランダムに被験者を選んでいるという解析方法

(6) 複数のクラスを訂正する必要のあるエラーのときの解析方法

5.1 被験者の実力

本実験では, 事前に C++の経験を必要としなかったため, 経験の全くないまたはわずかしかない被験者が 9 人中 7 人と大多数を占めた (表 1). さらに, そのうち, 言語を問わず, プログラミング自身をほとんど経験していない被験者もいた.

経験のほとんどない被験者については, 一つ一つのエラー訂正が新しいことであり, 設計の違いによる労力の差がなかった可能性がある. さらに, 難しいエラーに関しては,「できる」被験者しか訂正できず, 労力のデータとしては, 偏りのあるものになった. これは, 明らかに解析結果に影響を与える.

事前に C++についての講義・演習を行なったが, それだけでは上記の問題を克服できなかったことになる. つまり, 実験の前に, 演習を増やし, 被験者のレベルを上げてから, 実験を行なうか, ある程度最初から経験のある被験者に限ることが必要であった.

5.2 対象プログラムとエラーの選定

本実験で用いたプログラムは, Maryland 大学で行なわれた学生のプロジェクトの結果を用い, エラーはその中に含まれていたものを用いた. これらが問題であった可能性もある.

ほとんどのクラスは, MotifApp にある一つのクラスのサブクラスとして作成されていた. これは DIT の分布 (図 3) において, DIT が 0 と 3 のクラスしかなかったことに現れている. このため, 構造としては, C&K メトリクスとは関係なく, ほとんどのクラスが似ていた. この結果が DIT の相関係数に現れているととらえることができる. つまり, DIT=3 のクラスのほとんどが同じクラスのサブクラスであったために, 継承されるメンバ関数やインスタンス変数が同じで, 学習が起き, 仮説 H-DIT が棄却されたという解釈が可能である.

その上, 実験で用いたエラーはすでに存在していたものだけを用いたため, 幅広いものではなかった. つまり, 対象プログラムのことを考えたら, すでに存在しているエラーだけを対象にするよりも, 意図的に混入したエラーも含ませた方がよかったことが考えられる.

5.3 エラーシートの記入

本実験では, エラーシートを用い, そこで種々の時間を被験者に記入してもらった. 未記入のものは, データとして不完全であるため, 解析の対象から外しているが, 記入があっても, 定義通りの記入ではなかった場合がありうる.

労力を算出するためには,「症状確認時刻」と「チェック開始時刻」の二つを用いた．いずれも時刻の記入が遅れることが考えられるが，この場合，多くて数分ぐらいであることが予想できる．訂正するために2時間以上かかったエラーに関しては，数分は誤差の範囲と考えることができるが，全エラーの約3割は30分もかかっておらず，これらに関しては誤差であるとは言えない．このため，エラーシートの記入が影響を与えたことが考えられるが，自動的に判別するような手法を見つけない限り，これを完全に防ぐ方法はない．

5.4 修正すべきクラスに到達するまでの労力

C&K メトリクスと労力の間の関係は，あるエラーを直した時，そのエラーを直すために要した時間と，そのエラーを直すために修正したクラスのメトリクスの間の関係を調べることにより，行なった．しかし，あるエラーを直すために要した時間は，純粋に修正したクラスだけを見ていたとは言えない．例えば，修正すべきクラスに到達するまでに，他の関連クラスを見ていた可能性がある．もちろん，これは CBO や RFC の結果，他の関連クラスを見ていたと言う解釈もできる．しかし，たとえプログラムのほとんどのクラスが，あるひとつのクラスのサブクラスであり，構造が単純であっても，関係ないところを見ている可能性は否定できない．これは経験の少ない被験者が多いことからなおさら言えることであろう．

この問題を解決するためには，他の見ていたクラスを記録し，それをどの程度見ていたかまで記録する必要が出てくる．しかし，これは，被験者に負荷がかかるため，エラーシートと同様，自動的に判別する方法が望ましい．

5.5 エラー毎にランダムに被験者を選んでいるという解析方法

本研究では，C&K メトリクスと労力の間の関係を解析する際，各エラーについてランダムに被験者を選んだ．つまり，各エラーについて，被験者をランダムに選び，その被験者のデータをそのエラーに対するデータとして扱う．そして，そのエラーを直すために修正したクラスの C&K メトリクスと，そのエラーを直すためにかかった労力 (時間) のデータ対を用いた．このため，結果は一つのランダムな値の集合に対する結果であり，違う集合をとれば，違う結果を得ることが考えられる．

5.6 複数のクラスを訂正する必要のあるエラーのときの解析方法

一つのエラーを直すために，一つのクラスを直せばよいという状況であればよいが，必ずしもそうではなかった．つまり，エラー A を直すために，クラス XX とクラス YY を直すような状況が多々あった．このような場合，各クラスに要した労力は，(時間÷クラス数) で得た．例えば，エラー A を直すために1時間かかったら，クラス XX を直すために30分，クラス YY を直すために30分かかったと仮定して解析が行なわれた．しかし，必ずしもそうであったとは限らない．むしろ違うことが多かったであろう．例えば，30分30分ではなく，XX に5分，YY に55分かかったということがありうる．これが解析に多大な影響を与えたことは十分考えられる．

これは，XX を修正している時間と YY を修正している時間の境目をはっきりさせるという問題がある．エラーシートにそこまで記入するのは被験者にとって負担が大き過ぎることが考えられるため，現実的ではない．この問題を回避するためには，一つのクラスだけを修正すれば済むエラーだけを選ぶ，またはそのようなエラーだけを意図的に混入しなければならない．

5.7 まとめ

以上のように，本実験では，結果の妥当性に対して影響を与える問題点があり，統計的に得た結果を一般論として結論付けるには不十分である．特に，プログラムの構造上，H-DIT に関しては，確かめることが難しいと言える．もちろん，人間を扱っている実験である以上，たとえ考察した結果問題点がなかった場合でも，この実験を複製して，結果を確かめる必要はある．

本節であげた問題点のうち回避できるものについて，気をつけるべき点をまとめると，以下の通りである．

- 被験者の経験レベルは重要である．
- 対象プログラムの構造がワンパターンであるものは避ける．
- エラーに関しては，意図的に混入したものも含めることによって，種類を増やす．

- 一つのエラーを直すために，複数のクラスにまたがる場合は，解析が困難になるため，一つのエラーは一つのクラスを修正することにより除けるようなものにする．

6 おわりに

本論文では，オブジェクト指向設計と修正保守の労力の間の関係を調査する実験について述べた．C&K メトリクスの CBO, RFC, WMC, DIT を解析した結果，CBO だけが労力との間で相関が認められた．しかし，実験の遂行および解析の過程において，問題点を六つ上げた結果，労力と相関があるのは CBO だけであるという一般論としてまとめるには不十分である．

今後は，本実験で得られた問題点を考慮しながら，他のプログラム，他の被験者などで同種の実験を行ない，オブジェクト指向設計と修正保守の労力の間の関係を調査する必要がある．また，他のオブジェクト指向を計るメトリクスも調査する必要がある．

謝辞

実験に参加した学生に感謝したい．

参考文献

[1] F. Abreu, W. Melo: "Evaluating the Impact of Object-Oriented Design on Software Quality," Proc. of 3rd International Software Metrics Symposium (1996).

[2] J. Adamczyk: "Smalltalk Reaches Crossroads in the Insurance Industry," Comm. of the ACM, 38(10):107-109 (1995).

[3] V. Basili, L. Briand, W. Melo: "Measuring the Impact of Reuse on Quality and Productivity in Object-Oriented Systems," CS-TR-3395, Univ. of Maryland (1995).

[4] V. Basili, L. Briand, W. Melo: "A Validation of Object-Oriented Design Metrics as Quality Indicators," CS-TR-3443, Univ. of Maryland (1995).

[5] S.R. Chidamber and C.F. Kemerer: "A Metrics Suite for Object Oriented Design," IEEE Trans. on Software Engineering, 20(6):476-493 (1994).

[6] J. Grochow: "Smalltalk in the Telecommunications Industry," Comm. of the ACM, 38(10):110-111 (1995).

[7] M. Hitz, B. Montazeri: "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective," IEEE Trans. on Software Engineering, 22(4):267-271 (1996).

[8] 池田 央(編): 統計ガイドブック，新曜社 (1989).

[9] E. Kamsties, C. Lott: "An Empirical Evaluation of Three Defect-Detection Techniques," Proc. of the Fifth European Software Engineering Conf. (1995).

[10] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, C. Chen, Y. Kim, Y. Song: "Developing an Object-Oriented Software Testing and Maintenance Environment," Comm. of the ACM, 38(10):75-87 (1995).

[11] M. Lorenz, J. Kidd: *Object-Oriented Software Metrics*, Prentice-Hall (1994).

[12] A. Porter, L. Votta, V. Basili: "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," IEEE Trans. on Software Engineering, 21(6):563-575 (1995).

[13] R. Pressman: *Software Engineering: A Practitioner's Approach*, McGraw-Hill (1982).

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: *Object-Oriented Modeling and Design*, Prentice-Hall (1991).

[15] L. Wei, S. Henry, D. Kafura, R. Schulman: "Measuring Object-Oriented Design," Journal of Object-Oriented Programming, 8(4):48-55 (1995).

[16] D. Young: *Object-Oriented Programming with C++ and OSF/Motif*, Prentice-Hall (1992).

(盛 文 社)