ソフトウェア工学 111－4
（1996．9．11）

オブジェクト指向プログラムにおける保守の最適化に向けて

高田眞吾† 中小路久美代†‡ 高田義広† 鳥居宏次†

† 奈良先端科学技術大学院大学
‡ (株) SRA

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．いろいろな保守の作業のうち，ソフトウェア内で見つかったエラーの訂正では，エラーを保守スタッフのメンバの間に分割し，各メンバは割り当てられたエラーをある順序で訂正する．この分割と順序付けを「最適化」すれば，保守にかかるコストを抑えることが可能となる．
本論文では，これの第1歩として，オブジェクト指向プログラムの訂正に関する実験について報告する．結果より，すべての被験者は学習をしたわけではなかった．また，エラーがどの程度重大なものかで分類した際，難易度に違いがあった．将来的に，保守の最適化を考慮するとき，このような事柄を考慮する必要がある．

Towards Optimization of Maintenance of Object-Oriented Programs

Shingo Takada† Kumiyo Nakakoji†‡ Yoshihiro Takada† Koji Torii†

† Nara Institute of Science and Technology
‡ Software Research Associates, Inc.

With the emergence of object-oriented technology and its use in software development, the importance of maintenance of object-oriented software has grown. From the viewpoint of corrective maintenance, the errors that were found in the software would be divided up among a maintenance staff, and each staff member would go about fixing the error in a certain order. If there is some way to achieve an "optimal" division of errors among staff and an "optimal" ordering of errors to be corrected, then the cost needed for maintenance may be reduced.
As a first step towards such a goal, this paper reports on an experiment on the corrective maintenance of an object-oriented program. The results show that not all subjects show signs of learning and there is a difference in difficulty when the errors were categorized by severity. Both of these results should be taken into consideration in future endeavors of optimizing maintenance activities.

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られている [1] [3]．このため，その保守の重要性が高まるが，オブジェクト指向の場合，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 [5]．

ソフトウェアの保守には，エラーの訂正，機能拡張などがある [8]．それぞれ重要であるが，エラーの訂正は，すでにソフトウェア中に存在している問題を扱っている点で，特に重要である．

エラーは，ソフトウェア開発チームと異なるスタッフにより訂正されることが多い．この結果，保守のスタッフは事前にプログラムに関する深い知識を持っていない．これらエラーはスタッフの間で分割され，各スタッフのメンバはエラーのあるサブセットを担当する．例えば，保守のスタッフが3人の場合，30個のエラーがあれば，これらは3人の中で分割される．しかし，30個のエラーを訂正する時間を最低限にする「最適な」分割方法はない．メンバの経験やエラーの見かけ上の難易度をどう考慮するか，各メンバが10個担当するのか，など疑問がわく．

スタッフの間でエラーを分割したら，各メンバは担当のエラーをある順番で直す．しかし，訂正するのにかかる時間を最低限にする「最適な」順序付けの方法は存在しない．ここでも，エラーの難易度，学習など最適な順序を影響する要因をどう考慮するかなどの疑問がわく．

この分割と順序付けのプロセスを最適化する方法があれば，保守にかかるコストを抑えることができる．多くの組織では，保守がプログラム開発ほど技術が必要ではないと考えられているため，保守に割り当てられるスタッフの経験が比較的浅い [8]．このことを考えると，分割と順序付けのプロセスは特に重要となる．オブジェクト指向プログラムの保守にはプログラムを理解するところに問題がある [5] と言われているため，通常の手続き的手法により書かれたプログラムよりも順序付けはさらに重要であろう．つまり，学習が最もよく行なわれるような訂正の順序であれば，おそらく訂正にかかる時間を最も抑えることができるであろう．

この最適化を探る第1歩として，オブジェクト指向に基づいて書かれたプログラム中のエラー訂正に関する実験の結果を，学習とエラーの重大さの影響に注目して，報告する．

本論文では，まず，2節で実験の手順について述べる．3節では，実験の結果を記す．その後に，4節では学習の観点から，そして5節では，エラーの重大さから考察する．6節では本論文をまとめる．

2 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．

2.1 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 [2]，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 [9] 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．

2.2 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．

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

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)

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．

2.3 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，ある順番で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が [4][6]，本実験ではどの程度学習されたかを計った．しかし，エラー訂正の順番を考える際，注意しないと計れなくなる．本実験では，次の二つの点を考慮した．

- 全体的に，各エラーは同じ程度の学習の影響を受けるべきである．
- 経験レベルなどのため，全被験者が全エラーを直すとは限らない．

前者は，訂正の順番をランダムにするべきことを意味し，後者は，エラーを直す被験者の数がすべてのエラーに対して，ほぼ同じであるように気をつけるべきであることを意味する．

この二点を考慮して，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．

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)

2.4 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 [7] 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

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

れば，いったんその時間をチェック終了時間に記し，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，全エラーが訂正されたか，30時間作業し，被験者が終了を宣言する[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．

3　実験結果

エラー訂正の順番を決める際，各エラーが同じ程度の学習の影響を持つように考慮した．こ

[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．

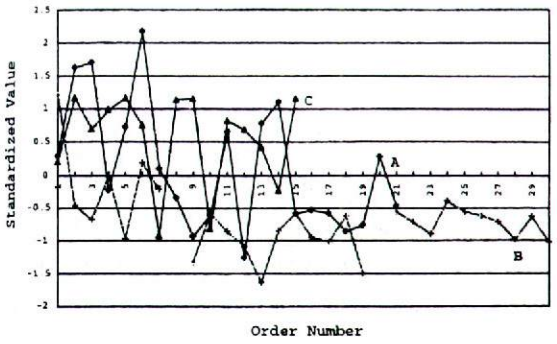

図 2: 被験者 A, B,C の正規結果

のため，結果を正規化した場合，学習の影響があれば,「正規結果」は，訂正が進む (つまり，エラー訂正の順番が増す) につれて，下がるべきである．ここで「正規結果」とは：

$$\text{正規結果} = \frac{\text{実際の時間} - \text{平均時間}}{\text{標準偏差}}$$

「時間」とは，エラーが確認されてから (症状確認時刻)，被験者がエラーを訂正できたと思った時刻 (チェック開始時刻) の間である．図2は，被験者 A, B, C の正規結果を示す．被験者 B の訂正順番 (order number) 20 のように，データが抜けているところがあるが，これはエラーシートが完全に埋まっていなかったため，時間を計算できなかったものである．

図2では，正規結果が下向きに傾いていない，つまり学習効果がなかったようである．ここでは示していないが，他の 6 人の被験者も同様であった．また，正規結果と訂正順番の間の相関係数を調べても，一番良い結果が-0.51 であるため，やはり学習効果がなかったようである．

訂正の順序は，エラーを 5 種類に分割してグルーピングした結果に基づいているため，次に図 3, 4, 5で，エラーを連続した 5 個にグルーピングした時の結果を示す．グループ番号 i は，訂正順番 $i, i+1, i+2, i+3, i+4$ の正規結果の平均である．例えば，グループ番号 10 は，訂正順番 10, 11, 12, 13, 14 の結果の平均である．

この場合，ほとんどの被験者において，最初の 10 グループの間でグラフが下向きに傾いている．明らかな例外として，被験者 F と G (共に

表 3: グルーピングした時の結果

	グループ#1-10		全エラー		学習
	相関係数	Mann-Whitney	相関係数	Mann-Whitney	
A	-0.853	OK	-0.827	OK	YES
B	-0.935	OK	-0.355	OK	YES
C	-0.777	OK	-0.656	OK	YES
D	-0.291	NO	+0.347	XX	NO
E	-0.844	OK	-0.467	NO	Maybe
F	+0.659	XX	+0.659	XX	NO
G	+0.175	NO	+0.338	NO	NO
H	+0.338	NO	-0.114	NO	NO
I	+0.620	NO	-0.289	OK	Maybe
ave	-0.790	OK	-0.958	OK	YES

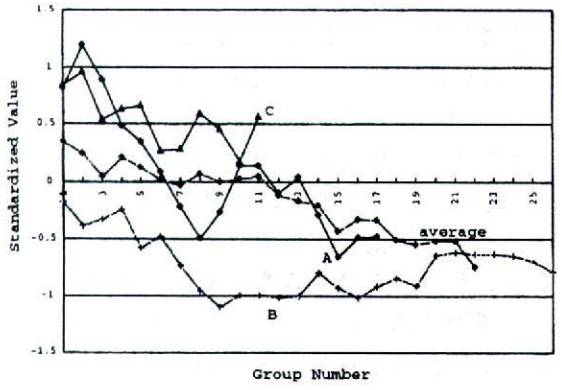

図 3: グルーピング時の結果 (被験者 A, B, C)

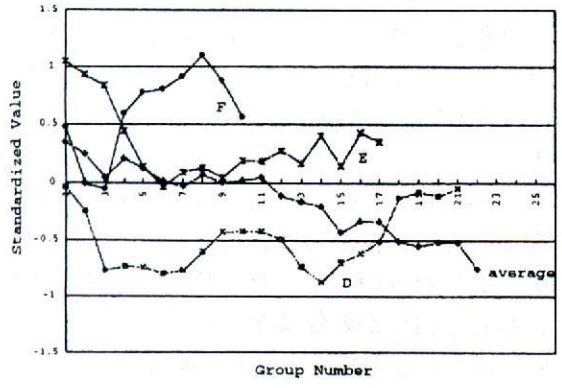

図 4: グルーピング時の結果 (被験者 D, E, F)

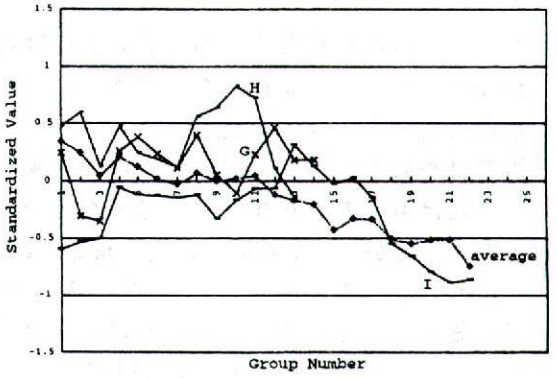

図 5: グルーピング時の結果 (被験者 G, H, I)

最初の三グループの後，結果が悪くなった)，そして最初が悪い方向であった被験者 I がいた．

表 3は，グループ番号と正規化した値の間の相関係数を示している．被験者 F が 14 個のエラーしか試行しなかったことを考慮して，結果は,「グループ#1-10」と「全エラー」に分けている．表 3は，二つの数字の集合が同じであるかどうかをチェックする Mann-Whitney テストの結果も示している.「グループ#1-10」の場合，一つの集合はグループ#1-5 であり，もう一つは #6-10 である.「全エラー」の場合，二つ目の集合は，その被験者にとっての最後の 5 グループである.「OK」は二つの集合が異なり，二つ目が

有意水準 0.05 で小さい方の結果 (つまり，二つ目の集合にあるエラーが早く訂正された) であると認められたものである.「NO」は違いがなく,「XX」は違いがあったが，二つ目の集合が有意水準 0.05 で大きいと判定されたものである.

表 3より，エラーをグループにまとめると，被験者 A, B, C は学習の影響を見せている．被験者 E は最初は見せていたが，全体的には見せていない．逆に，被験者 I は最初の 5 グループと最後の 5 グループだけを考慮すると，学習の影響を見せている．しかし，I の場合，最初の 3 グループを考慮から外すと，相関係数が-0.623 となり，いくらか学習していることがうかがえる.

全被験者の平均を見ると，学習が起きていたといえる．ただし，全被験者の全エラーの相関係数として-0.958，Mann-Whitney テストでも OK という結果を得ているが，この結果はそのまま利用してはならない．被験者の間で試行したエラーの数が異なるため,「良い」被験者の方が多くのエラーを試行した．つまり，エラーの数から見ると，実験が進むにつれて,「悪い」被験者はやめて,「良い」被験者だけの結果が残る．これはもちろん正規化した結果が下がる結果となる．しかし，全被験者は訂正順番 14 までエラーを訂正したため,「グループ#1-10」の結果は，妥当である．そのため，やはり実験が進むにつれて，訂正にかかる時間が低下したことには変わりがない.

4 学習効果

保守作業の最適化を考える時，学習の効果を考慮すべきである．保守のスタッフがソフトウェア開発スタッフと異なることが多く，保守のスタッフの経験レベルが低い可能性があるため，各メンバが効率良く学習するように，エラーを分割し，順序付けることができれば，保守の効率を上げ，保守の最適化につながる.

4.1 学習対象

学習する時，次の 3 点が対象として考えられる：(1) 実験手順に対する慣れ；(2) C++；(3) プログラム.

まず，実験手順の慣れが，学習の対象として考えられる．エラーシートの使用や実験者による訂正のチェックなど，本実験では，通常の保守と異なる環境で行なった．本実験の前に予備実験を行なったが，何人かの被験者は最初はエラーシートの記入を不自然に感じたかもしれない．また，最初の頃は，訂正のチェックの要請を遅らし，エラーの訂正にかかる時間がのびていた可能性がある.

次に C++に関しては，表 1の通り，B と C を除いて，C++の経験はあまりなかった．このため，実験が進むにつれて，C++コードを読む早さが増した可能性は十分考えられる．どの程度影響があったかは結論しにくい．例えば，C++の経験 (表 1) と学習 (表 3) の間に相関は見られない．また,「熟練した C プログラマから熟練した C++プログラマになるのに，ほとんどの場合，6 カ月以上かかる」[2]といわれている．実験に含まれたエラーを訂正するために,「熟練した」C++(や C) プログラマである必要はないが，エラーを訂正するためには，ある程度プログラミング能力が必要であった.

最後に，プログラムに対して学習したために，エラーの訂正が早くなったことが考えられる．これは，システムを作成した人達のスタイルを学習することも含む．この要因は，少なくとも訂正箇所を探索する開始地点を決める分には，大いに影響がある.

4.2 本当に学習だったのか?

結果の傾きが下向きとなった原因として学習の対象を三つ上げたが，学習の影響ではないと考えられる要因として，次の 2 点が考えられる：(1) ギブアップ；(2) エラーと関係のない部分のチェック.

被験者がギブアップしたエラーの時間は，何も変更せずに，そのまま用いた．これは，ギブアップしたものでも，その時間の間にシステムに対して何らかの作業を行なっていたからである．ほとんどのギブアップは最初の方にあった：全ギブアップの 47%は最初の 5 つのエラーの中にあり，全ギブアップの 73%は最初の 9 つのエ

[2] http://www.edge.net/algorithms/cpp6.html

ラーの中にあった．ほとんどの場合，これらのギブアップしたエラーは平均を上昇させることになるため，ギブアップの分布だけでも，正規化した時間が下向きに傾いていることを意味する．しかし，ギブアップの数の減少そのものを学習の影響と見ることができる．

被験者は，与えられたエラーと関係のない部分を見ていた可能性がある．これは最初の方に起きやすいであろう．このため，やはりこれだけで結果のグラフが下向きに傾くことになる．しかし，これも「学習の行動」とみなすことができ，そのとき見ていた部分に対して「学習」したと考えることができる．

4.3 学習と経験

C++の経験と学習の間に相関を見ることができないが，学習の影響を一番良く見せた三人を見ると，三人ともCで1000行以上のプログラミングの経験がある．C++がある意味ではCのオブジェクト指向版であるため，Cのプログラミング経験がある程度あった方が，C++を読むのに慣れやすいことを意味する[3]．

しかしながら，1000行以上のCプログラミングの経験があるにも関わらず，学習の効果を見せなかった被験者が一人(被験者D)いた．被験者Dは，最初は下向きに傾いていたが，三つ目のグループの後，一定の値になった(図4)．これは，被験者Dは最初の数エラーの間で学習がほとんど終り，その後は,「最速」で訂正していたことを示すようである．しかし，二箇所ほど傾きが上向きになるところがある：グループ番号7から9の間と，14から最後まで．エラーシートを確かめた結果，両方の場合，傾きが上向きになったのは午後2時過ぎである．これだけでは結論付けられないが，被験者Dが疲労にあった可能性がある．

このCとC++の保守の間の関係が本当の関係なのか，それとも実験で用いたプログラムの「非オブジェクト指向」の現れであるかを探るために，C++で書かれた他のオブジェクト指向プログラムでさらなる調査が必要である．

[3] もちろん，これとは逆にシステムがあまりオブジェクト指向的に書かれていないことを意味する可能性もある．

5 エラーの重大度の影響

エラーの分割と順番を考える時，手元にある唯一の情報はエラーの症状である．そこで，本節では，エラーの症状がどの程度重大か(重大度; 表4)に注目する．

表4: エラーの重大度による分類

単純	間違ったつづりなど単純なエラー
低	重大なエラーであるが，必要な機能の使用やテストを妨げない
中	必要な機能の使用やテストを妨げる
高	システムが動作しない

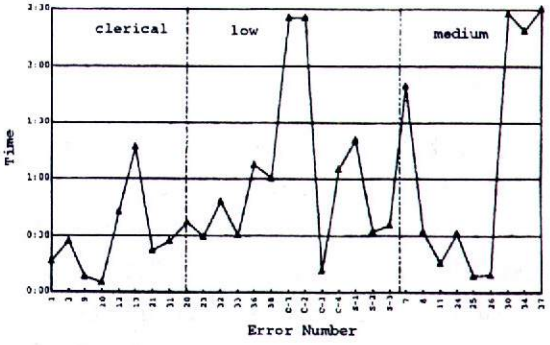

図6: エラーの重大度に基づいた結果(1)

表5: エラーの重大度に基づいた結果(2)

重大度	平均時間	ギブアップ率
単純	0:26:02	0%
低	0:59:42	18%
中	1:04:31	26%

エラーの重大度に基づいた結果を図6と表5に示す.「clerical, medium, low」は，それぞれ重大度の「単純，低，中」に相当する[4]．平均は，被験者がギブアップしたものを含んでいない．つ

[4] 「高」はなかった．

まり，この時間は，被験者がエラーを確認してから，エラーを訂正したと考え，実際に正しく直した時までの時間である．また，表5では，重大度の三種類の平均とギブアップ率を示している．

図より,「単純」が最も簡単であるが,「低」と「中」の間に特に差があるようには見受けられない．この反面，ギブアップを考慮すると，表5より,「単純」の場合は，一度もギブアップがなく,「低」の場合18%,「中」の場合26%とはっきりとした差が出た．

これらの結果より,「低」と「中」のエラーの間で正しく訂正するためにかかる時間に差はあまりないが，難易度に関しては，差があった．もちろん，これを一般化するためには，他のプログラムでさらに調査する必要があるが，他のプログラムでも同様のことが言えることができれば，保守におけるエラー訂正作業を最適化する時，この情報を利用することができる．

6 おわりに

本論文は，C++で書かれたソフトウェアのエラー訂正に関する実験の結果を示した．全体的には被験者は学習の影響を見せたが，個々の被験者を見ると，必ずしもそうではなかった．学習の影響を見せた被験者は1000行以上のプログラムをCで書いた経験のある被験者に集中していた．また，重大度により，エラーの間に違いを見ることができた．

今後の課題として，学習とCプログラミングの経験の間の関係，および難易度とエラーの重大度の間の関係を一般化できるかどうかの確認がある．これができれば，保守におけるエラー訂正作業の順序付けの視点から，これらの情報を用いて，保守の作業の最適化を行なえる可能性がある．また，事前にある情報はエラーの症状だけであるので，症状だけを利用した他の分類を考えるべきである．

謝辞

本実験を行なうにあたって，CRIMのWalcelio Meloに助言を頂いた．また，実験に参加した学生に感謝したい．

参考文献

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

[2] 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).

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

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

[5] 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).

[6] 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).

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

[8] I. Sommerville: *Software Engineering*, Addison-Wesley (1992).

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