ソフトウェア工学 92－6
（1993．5．26）

信頼度モデルをもちいたオブジェクト指向技術の技術評価

山口 隆弘* Givargis A. DANIALY**

(*) アドバンテスト仙台研究所
〒989-31仙台市青葉区上愛子字松原４８番２

(**)Advansoft Research Corporation
4301 Great America Parkway Santa Clara,
CA 95054, USA

手続き型プログラミングとオブジェクト指向プログラミングのどちらが，次世代の大規模ソフトウェアの開発・テスト・保守・機能拡張をおこなうのに適しているのであろうか？簡単なソフトウェア信頼度モデルをもちいて，これらのプログラミング・パラダイムの間に有意の差があることを示した。最も重要なポイントは分析である。適切な分析により，信頼度を向上させ，テスト時間を短縮できる。

Simplified Formulation: The Software Reliability Growth Model for Procedure-Based and Object-Oriented Programming

Takahiro YAMAGUCHI* Givargis A. DANIALY**

(*)ADVANTEST Sendai Laboratories, Ltd.
48-2 Matsubara, Kamiayashi, Aoba-ku,Sendai,
Miyagi, Zip 989-31,JAPAN

(**)Advansoft Research Corporation
4301 Great America ParkwaySanta Clara,
CA 95054, USA

In recent years, object-oriented programming (OOP) has attracted attention, primarily because it offers the advantage of reusing program modules. Using natural expressions that are intuitively easy to understand, the technique has been successful in small-scale systems. Essentially, it is characterized as having powerful modeling capabilities. However, programmers need time to learn this technique, which also lacks both a rigorous mathematical formalism and any published quantitative evaluations. For this reason, an investigation was performed using reliability growth models to determine which software technology should be used in the implementation of new measurement systems.

Since software development may be looked at as an incremental maintenance, it is very important to use a paradigm that allows intuitive, easy and reliable enhancements and bug removal.
Enhancements: we will show it is easy and intuitive to add new code to an already existing OO system.
Bug Removal: There are three major steps in fixing a software system defect. We will show that OO paradigm improves software reliability in all three steps:
A. Diagnosing the problem
B. Making changes for the defect and all side effects
C. Testing the system for integrity

1. はじめに

CPUの計算能力の飛躍的向上と複数項目を同時に測定する計測技術の高度化のため，計測分野でもソフトウェアの比重が大きくなってきている。また，ソフトウェア開発コストの85%は実現工程(設計・作成・テスト)が占める[1]。では、次世代の大規模ソフトウェアの開発・テスト・保守・機能追加をおこなうには，従来の手続き型プログラミング技法でいいのであろうか?

ここ数年，プログラムの再利用の観点から『オブジェクト指向技術』が注目されている。オブジェクト指向は，『直感的で自然な表現をもつ技術』である。すなわち，『強力なモデリング能力』が特徴である[2]。しかし，技術の習得に時間がかかり，数学的な基礎づけが弱い。さらに，オブジェクト指向技術の定量的技術評価報告が少ない。

このため，新しい計測システムを構築するために，どのソフトウェア技術を利用すべきかを信頼度成長モデルをもちい検討した。検討結果は，オブジェクト指向技術の採用により，信頼度が向上しテスト時間を短縮できることを示している。

2. 問題の定義

手続き型プログラミング技術をもちいて計測器組み込みソフトウェアを開発した経験をまとめる: I.分析，設計とプログラミングの対応を維持するのが困難である。II.全体を完全に理解していないため，開発工数を過小評価してしまう。III.大規模ソフトウェア開発では，メンバー内で最終システムについてのイメージを共有しにくい。とくに開発途中の仕様変更/追加にたいし，最終システムについての理解をチームメンバー内でコヒーレントに維持していくのは困難である。IV.優秀な人間を投入しても，「いつ終了するのか」，「どのような品質になるのか」を確信をもって報告できない。V.大規模になればなるほど品質確認のためのテスト工程が長くなり，大量のテスト労力を必要とする。

結果的に「この技術では，より大規模な開発はできない」というのが実感である。たとえば図1は，開発した計測器組み込みソフトウェアの瞬間エラー発見率の例である(約165,000行のCコード; NCSS; オン・スケジュールのデリバリー)。このテストプロセスの2/3は‘確率変数との戦い’であり，残り1/3は‘忍耐’である。

ソフトウェア開発の立場からは，オブジェクト指向技術を採用すると，I.ソフトウェア信頼度が向上するか? II.さらに，信頼度が向上するとしたら，瞬間エラー発見率であらわされるソフトウェア開発プロセスもパラダイム・シフトするか? III.オブジェクト指向技術を効果的に適用するには，なにが最も重要か? を予測し，手続き型プログラミングの適用との有意の差があるかを数値で明らかにする必要がある。

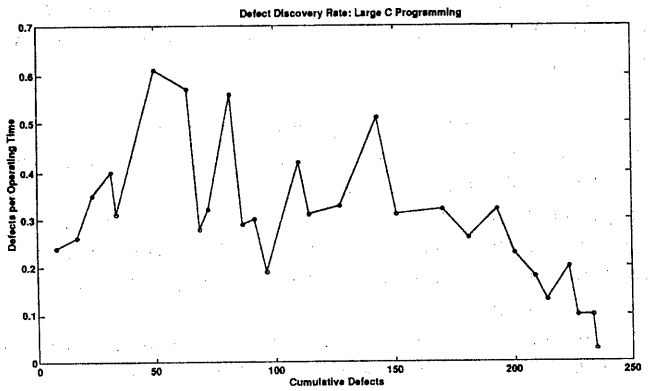

図1. 手続き型プログラミング技術ソフトウェアの瞬時エラー発見率の例

Figure 1 An example of defect discovery rate of procedural programming.

3. オブジェクト指向技術の基本原理

オブジェクト指向技術は，基本的につぎの特徴をもつ[4],[5]。

a.「抽象データ型」データ構造とそれに適用できる操作をまとめて「データ型」として 局所化し定義したもの。「抽象データ型」は，その対象データの表現や演算の実装を特定しないで，この「データ型」へのインタフェイスをあたえる。このような情報隠蔽をカプセル化(encapsulation)と呼ぶ。

b.「継承」おのおののクラスにたいし，操作集合メソッドがたいおうする。クラスが上位-下位階層を構成しているとき，メソッド継承により上位クラスで定義されたメソッドを下位クラスで利用できる。‘差分プログラミング’により，メソッド定義を単純化可能である。

分析とオブジェクト指向プログラミングの間に1対1対応をもとめる経験則"Balancing the structure between solution domain concerns and application domain concerns is an engineering consideration crucial to system maintainability and to architectural aesthetics. Failure to preserve close relationship between application domain formalism, and solution constructs, can lead to an inflexible implementation that is difficult to understand."[4]は，分析とプログラミングの間を‘同形(isomorphic)’[12]にできることを示している。また，オブジェクト指向技術についてのコメント『強力なモデリング能力』[2]や『メリットは分析とプログラミングの1対1対応である』[3]は，‘同形’の存在を語っている。

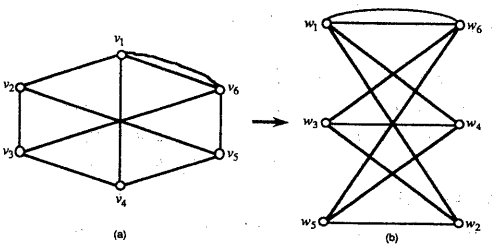

図2. オブジェクト指向分析とプログラミングの間の同形: (a)分析, (b)プログラミング

Figure 2 Isomorphic graphs: Object-oriented analysis and object-oriented programming.

この‘同形’は，従来の手続き型プログラミングでは，存在しない。

4. ソフトウェア信頼度の検討

ソフトウェア信頼度は，『ソフトウェアあるいは構成モジュールが，規定の環境のもとで，意図する期間中にソフトウェア故障が発生することなく動作する確率』と定義される[7]。ソフトウェア・エラーが，ソフトウェア故障を引き起こす。ソフトウェア信頼度$R(x|t)$は，瞬間エラー発見率$h(x)$を計測することによりあたえられる[7]。

$$H(t) = \int_0^t h(x)dx \qquad (1)$$

$$R(x|t) = \exp\left[-\{H(t+x) - H(t)\}\right] \qquad (2)$$

ここで$H(t)$は，時刻tまでに発見される総期待エラー数をあらわす。

評価対象のプログラムの流れを図3と仮定する。i番目のモジュールの信頼度R_iとすると，このシステムの信頼度R_{Total}は，(3)であらわされる[6]。ここで全モジュール数は，Nである。

$$R_{Total} = R_1 \bullet R_2 \bullet \ldots \bullet R_k \bullet \ldots \bullet R_N = \prod_{i=1}^{N} R_i \qquad (3)$$

つぎに，k番目のモジュールを変更したときの手続き型とオブジェクト指向型の信頼度を比較しよう。

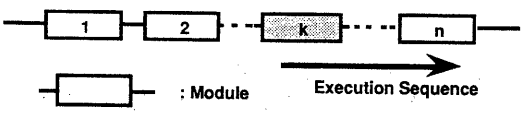

図3. 評価対象システム

Figure 3 System under evaluation.

4.1 手続き型の振る舞い

ソフトウェア・エラーは複数の要因により発生する。さらに,ソフトウェア開発プロセスは‘雑音’-仕様変更/追加/開発者の能力や注意深さの不均一性-がおおいため,因果関係‘どの要因がこの結果をひきおこすのか’をはっきり計測しにくい。このため,オブジェクト指向の考え方‘「Howなぜこのようになるかの詳細なメカニズム」にとらわれなく「What結果としてあらわれる最も重要な特徴」をとらえる’をもちいて,手続き型の振る舞いをモデリングする。

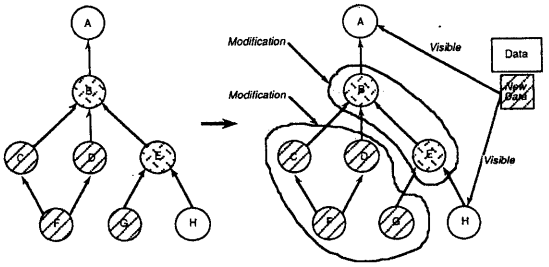

図4. 関数呼び出しグラフと修正の影響
Figure 4 Call graph in procedure-based system and the side-effects of small enhancement.

図4 (a) の有向グラフは,手続き型の関数呼び出しグラフをあらわしている。この有向グラフの矢印は関数呼び出し"USE-A"関係を意味している。「上層モジュールAのために新しいデータを追加し,下層モジュールHにもアクセスさせる」変更が必要になったとする。データをHにわたすため,まずBとEを拡張する。さらにこの修正のため,ほかの下層モジュールの修正も生じるかもしれない。このように,モジュールの修正が爆発的に伝播してしまう。すなわち,ほとんど全てのモジュールを再テストすることになる。手続き型の振る舞いの特徴は,「このデータ結合をつうじて,モジュールの独立性が“弱く”なってしまう[8]」ことである。また,独立性が“弱く”なるモジュールは,分析結果とはなんの相関ももたない。

この振る舞いを,最悪ケースの手続き型のモデルとしてもちいる。

4.2 手続き型の信頼度

k番目のモジュールを変更したときの手続き型のシステムの信頼度を導く。記号$R_{k,i}$は,k番目のモジュールを変更したとき,影響をうけるi番目のモジュールの信頼度をあらわすとする。一般的には

$$R_{k,i} \le R_i;\ \frac{R_{k,i}}{R_i} \equiv \left(1-\varepsilon_{k,i}\right) \le 1.0 \qquad (4)$$

である。

理想的変更は,k番目以外のほかのモジュールにたいして,副作用をあたえない。このときの信頼度は,つぎの式となる。

$$\begin{aligned} & R_{Total|k} \\ & = R_1 \bullet R_2 \bullet \ldots \bullet R_{k,k} \bullet \ldots \bullet R_N = \left(\prod_{i=1}^{N} R_i\right)\left(\frac{R_{k,k}}{R_k}\right) \\ & = R_{Total}\left(\frac{R_{k,k}}{R_k}\right) \le R_{Total} \end{aligned} \qquad (5)$$

実際の変更は,ほかのモジュールにたいしてゼロでない副作用をあたえる。

$$\begin{aligned} & R_{Total|k} \\ & = R_{k,1} \bullet R_{k,2} \bullet \ldots \bullet R_{k,k} \bullet \ldots \bullet R_{k,N} = \prod_{i=1}^{N} R_{k,i} \\ & = R_{Total}\left(\prod_{i=1}^{N} \frac{R_{k,i}}{R_i}\right) \le R_{Total} \end{aligned} \qquad (6)$$

$$R_{LOSS|Procedural} \equiv \frac{R_{Total|k}}{R_{Total}} \propto \mathrm{O}\left((1-\varepsilon)^N\right) \qquad (7)$$

と信頼度はあらわされる。「データ結合をつうじて,モジュールの独立性が“弱く”なってしまう」最悪ケースのときは,k番目のモジュールの変更の副作用は,システム全体に拡散する。瞬間エラー発見率パターンは,不確定事象の振る舞いをしめす。すなわち,発見されるエラー間の相関は小さく,エラーを発見するのが困難である。このとき,テスト労力は全モジュール数Nに比例する量以上となる。

4.3 オブジェクト指向型の信頼度

オブジェクト指向型では，カプセル化と継承関係によりシステムを構築できる。オブジェクト指向型の継承木を図5にしめす。k番目のクラスの子孫クラス数をLとする(L≦N)。k番目のクラスを変更したときの再利用システムの信頼度$R_{Total|k}$は(8)によってあたえられる。

$$R_{Total|k} = (R_{k,1} \bullet R_{k,2} \bullet \ldots \bullet R_{k,k-1}) \bullet (R_{k,k+L+1} \bullet R_{k,k+L+2} \bullet \ldots \bullet R_{k,N}) \bullet \{R_{k,k} \bullet R_{k,k+1} \bullet \ldots \bullet R_{k,k+L}\} \quad (8)$$

ここで$\{R_{k,k} \bullet R_{k,k+1} \bullet \ldots \bullet R_{k,k+L}\}$は$R_{k,k}$の子孫クラスであり，継承木のk番目のクラスをルートとする子や孫などのクラスである。オブジェクト指向ではk番目のクラスを変更したときの副作用は，この子孫クラスのみである。したがって，k番目のクラスと子孫関係にないクラスは，クラス“k”変更前の信頼度をたもつ。(8)は(9)のように変形される。

$$R_{Total|k} = (R_1 \bullet R_2 \bullet \ldots \bullet R_{k-1}) \bullet (R_{k+L+1} \bullet R_{k+L+2} \bullet \ldots \bullet R_N) \bullet \{R_{k,k} \bullet R_{k,k+1} \bullet \ldots \bullet R_{k,k+L}\} = R_{Total}\{\prod_{i=0}^{L} \frac{R_{k,k+i}}{R_{k+i}}\} \le R_{Total} \quad (9)$$

$$R_{LOSS|Object-Oriented} \equiv \frac{R_{Total|k}}{R_{Total}} \propto O\left((1-\varepsilon)^L\right) \quad (10)$$

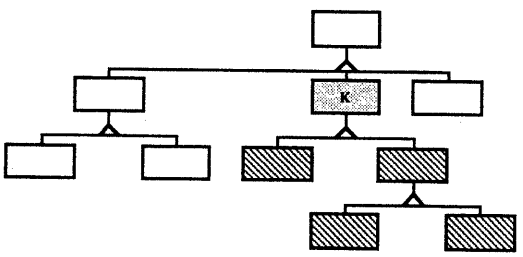

図5. 継承木
Figure 5 An inheritance tree: "IS-A" relation.

これからオブジェクト指向型の再利用において，クラス“k”を変更したとき信頼度に影響をうけるクラスは子孫クラスに限定され，テスト労力は全子孫数Lにのみ比例する量以上となるのがわかる。

この定式化から，さらにつぎのことが明らかになる。まず，“子孫テスト”というあたらしいテストがあらわれる。さらに，オブジェクト指向型のもつ‘同形’のため，発見されるエラーの分布は分析結果にたいおうする。このため，エラーは概念的な順序関係をもってくる。すなわち，「発見されるエラー間の相関は大きい。」すなわち，テスト時間の経過とともに，ソフトウェア内に残存するエラーが発見されやすい傾向をもつ(習熟S字形信頼度成長モデル[7])。

また，現実の問題領域の意味との関連をテストに利用できるから，テストは手続き型より比較的容易になる。さらに，仕様から「この部分はテストが必要」というテスト手続きの生成も原理的に可能である。

ぎゃくに，概念構造のひずみからエラーが発生するから，分析は手続き型より重要となる。

どのような技術でも，開発された最初のモジュールはテストする必要がある。したがって，開発直後のオブジェクト指向型の信頼度は，(9)でなく(6)にたいおうする[9]。

4.4 信頼度のまとめ

手続き型とオブジェクト指向型の信頼度とテスト時間の比較を表1にまとめた。

		Procedure-Based System	Object-Oriented System
Immediately after development	Reliability	$\prod_{i=1}^{N} R_{k,i}$	
	Test Time	$\prod_{i=1}^{N} T(R_{k,i})$ Increases faster than the total number of modules, N	
	Tested Modules	Unit test and integration test: $R_{k,1}, R_{k,2}, \ldots, R_{k,N}$ Number of tested modules: N	
Re-use	Reliability	$\prod_{i=1}^{N} R_{k,i}$	$\prod_{i=1}^{k-1} R_i \bullet \prod_{i=k+L+1}^{N} R_i \bullet \{\prod_{i=0}^{L} R_{k,k+i}\}$
	Test Time	$\prod_{i=1}^{N} T(R_{k,i})$ Increases faster than the total number of modules, N	$\{\prod_{i=0}^{L} T(R_{k,k+i})\}$ Increases faster than the total number of descendants, L
	Tested Modules	Unit test $R_{k,k}$ Side-effects test $R_{k,i}(i \neq k)$ Number of tested modules: N	Unit test $R_{k,k}$ Descendent classes test $R_{k,k+1}, R_{k,k+2}, \ldots, R_{k,k+L}$ Number of tested classes: L+1

表1　手続き型とオブジェクト指向型の信頼度とテスト時間の比較

Table 1 Reliability and test time for procedure-based and object-oriented system

再利用のときに手続き型と比べ，オブジェクト指向型の信頼度についての優位性を確保するには，全モジュール総数Nにたいし子孫モジュール数Lを非常に小さくなるよう設計することが必要である。

$$R_{LOSS|Object-Oriented} << R_{Procedural} \quad (11)$$

$$L << N \quad (12)$$

つまり，オブジェクト指向分析のときにどれだけ$L << N$をみたす簡潔でバランスのとれた継承木をもつモデリングをおこなえるかが最も重要である。すなわち，オブジェクト指向分析では概念のレベルがキーポイントであることを示している。

5. オブジェクト指向型の信頼度の特徴と対応原理

図6は, オブジェクト指向型の信頼度の子孫モジュール数Lにたいする依存性を計算した例である[13]。

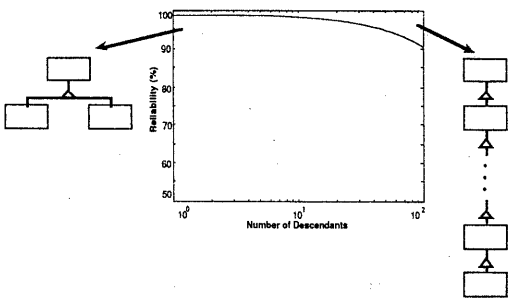

図6. オブジェクト指向システムの信頼度と
子孫モジュール数
Figure 6 Reliability of the object-oriented system
versus number of descendants.

子孫モジュール数Lが大きくなると, 信頼度の値が急激に小さくなる傾向がわかる。この図の右端は, $L \approx N$ にたいおうしている。継承木の形は, 「親以外の全てのクラスは, この親クラスの子孫」をあらわす直線になっている。これは, 「データ結合をつうじて, モジュールの独立性が“弱く”なってしまう」最悪ケースの手続き型を近似している。

$$R_{\Pr ocedural} = \lim_{L \to N} R_{Object-Oriented}(L,N) \qquad (13)$$

つぎに, このソフトウェア信頼度の違いをグラフ理論をもちいて解釈する。クラス“点”と継承“辺”は, 継承木“グラフ”Gを構成する。適切なオブジェクト指向分析は, それぞれの問題領域により決まるグラフGの構造のうち2点u,v間の路(path)の長さを最小値の‘距離$d(u,v)$’にたいおうさせる構造を発見する操作である。最悪ケースの手続き型プログラミングは, このグラフGを, 2点u,v間の路の長さを最大値‘直径$diam(u,v)$’にたいおうさせる構造を選択してしまう操作である。

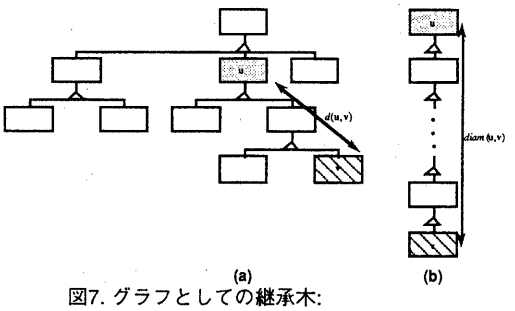

図7. グラフとしての継承木:
(a)オブジェクト指向, (b)手続き型
Figure 7 An inheritance tree as graph.

‘直径$diam(u,v)$’はグラフを構成するクラス“点”の数が増加すれば, 大きくなる。したがって, 大規模システムの開発ほど‘直径$diam(u,v)$’より‘距離$d(u,v)$’にたいおうする構造を発見できるオブジェクト指向型の採用が適切である。

図8は報告されている, Booch法のオブジェクト指向分析・設計CASEツールROSEの開発のときの瞬間エラー発見率の例である(約170,000行のC++コード; NCSS; このパターンの再現性あり)[11]。習熟S字形信頼度成長モデルの「発見されるエラー間の相関は大きい」特徴があらわれている。図1の手続き型の瞬間エラー発見率と比較すると, 有意の差を認められる。

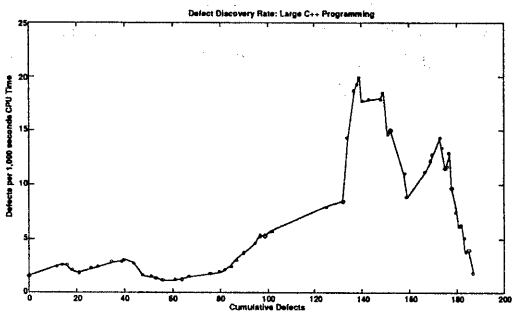

図8. オブジェクト指向プログラミング技術
ソフトウェアの瞬時エラー発見率の例
Figure 8 An example of defect discovery rate of
object-oriented programming.

6. おわりに

非常に簡単なモデルをもちいて, 手続き型プログラミングとオブジェクト指向型プログラミングの信頼度を定式化した。このとき, 手続き型プログラミング・モデルの属性としては「データ結合をつうじて, モジュールの独立性が"弱く"なってしまう」ポイントを; オブジェクト指向型プログラミング・モデルの属性としては「分析とプログラミングの間に1対1対応の'同形'」を一般化した。

この定式化から, つぎの事項を明らかにした。I.オブジェクト指向技術を効果的に適用するには, オブジェクト指向分析が最も重要である。II.オブジェクト指向型プログラミングのソフトウェアを再利用したとき信頼度はクラスの子孫クラス数に比例して劣化する。III.さらに, 瞬間エラー発見率であらわされるソフトウェア開発プロセスは指数形信頼度成長モデルから習熟S字形信頼度成長モデルへパラダイム・シフトする。

さいごに, オブジェクト指向と手続き型プログラミングのソフトウェア信頼度をグラフの'距離$d(u,v)$'と'直径$diam(u,v)$'の概念に対応させることができることを明らかにした。

謝辞

信頼度の検討方向を示していただいたASLの八鍬所長に, オブジェクト指向の概念についての議論をしていただいた中込部長に; 数ケ月まえのプロジェクトで'同形'の概念のソフトウェア開発プロセスへの有効性確認の実験をおこなう機会をあたえていただいたアドバンテストの神谷事業本部長に感謝します。

参考文献

1. Robert B. GRADY and Deborah L. CASWELL, Software Metrics: Establishing a Company-Wide Program, Prentice-Hall Inc.,1987.
2. Joshua M. DUHL, "オブジェクト指向DBの現状と課題," 情報処理学会連続セミナ, 工学院大, 11月1992年.
3. 相原敬雄, "オブジェクト指向管理ソフト/CASE現状と課題," 情報処理学会連続セミナ, 工学院大, 11月1992年.
4. James O. COPLIEN, Advanced C++ Programming Styles and Idioms, Addison-Wesley Publishing Company, 1992.
5. 青木　淳, "オブジェクト指向ソフトウェア開発", FXIS, 1990.
6. 真壁　肇, 信頼性データの解析, 岩波書店, 1987.
7. 山田　茂, ソフトウェア信頼性評価技術, HBJ出版局, 1989.
8. Tom DeMARCO, 構造化分析とシステム仕様, 日経BP社, 1986.
9. John A. LEWIS, Sallie M. HENRY, Dennis G. KAFURA and Robert S. SCHULMAN, "An Empirical Study of the Object-OrientedParadigm and Software Reuse," Conference on OOPSLA'91, Phoenix, Arizona,USA, October 1991.
10. Satoshi IMAI, Takahiro YAMAGUCHI, and Givargis A. DANIALY, "Which Paradigm Can Improve the Reliability of Next-Generation Measurement System Software ?," OOPSLA'92, Vancouver, British Columbia, CANADA, October 1992.
11. James F. WALSH, "Preliminary Defect Data from the Iterative Development of a Large C++ Program," OOPSLA'92, Vancouver, British Columbia, CANADA, October 1992.
12. 同形の定義をつぎにあたえる。Two graphs G_1 and G_2 are said to be isomorphic if there exists a one-to-one correspondence between their vertex sets and one-to-one correspondence between their edge sets so that the corresponding edges of G_1 and G_2.
13. システムは100のモジュールからなる。変更まえの各モジュールの信頼度を99%とする。k番目のモジュールは, 変更のあと単体テストにより信頼度99%をえたものとする。この変更により副作用を受けた各モジュールの平均信頼度を98%と仮定する。
14. 山口隆弘, Givargis A. DANIALY, James F. WALSH, "次世代計測システム・ソフトウェアの信頼度向上のためのオブジェクト指向技術の適用," 1993情報学シンポジウム.