ソフトウェア工学　88－６
（1992. 11. 10）

次世代計測システム・ソフトウェアの信頼性は、どのパラダイムによって改善されるか？

信頼性モデルをもちい、『手続き型とオブジェクト指向型の信頼度』の
テスト範囲・時間を数値で比較する。

今井　聡*　、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

あらまし

ソフトウェア製品の８５％のコストは機能追加と保守の現状である。では、次世代の大規模ソフトウェアの開発・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？
近年、プログラムの再利用の目的で『オブジェクト指向技術』が着目されてきているが、数学的な基礎づけが弱く、定量的な評価がされていない。このため、信頼性モデルをもちい、『手続き型とオブジェクト指向型の信頼度』のテスト範囲・時間を数値で比較した。この結果、オブジェクト指向型は手続き型に比べ、よいオブジェクト指向分析により信頼度を向上させテスト時間を短縮できることがわかったので報告する。

和文キーワード　　信頼性モデル、手続き型プログラミング、オブジェクト指向型プログラミング

Which Paradigm Can Improve the Reliability of Next-Generation Measurement System Software ?

Reliability in Procedural and Object-Oriented Programming

Satoshi IMAI*　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 Parkway
Santa Clara, CA 95054, USA

Abstract

The cost of software, which has become an important element in measurement system, is on the increase. Experience has shown that with traditional procedural programming, 85% of software development cost is incurred in implementation. Object-oriented programming (OOP) , by enabling reuse of program modules and using natural expressions, offers hope of reduced costs. The authors used reliability models to evaluate the potential of OOP in the development of new measurement systems, the results indicating with proper object-oriented analysis (OOA) the OOP techniques can improve software reliability and shorten software testing times.

英文 key words　　reliability models, procedural programming,object-oriented programming

Introduction

Measurement system software is becoming increasingly important as digital processors become more powerful and high-speed, multifunction measurement systems become more complex. The cost of software, therefore, is likely to be an increasing proportion of the overall system cost. Experience has shown that, using traditional procedural programming method, 85 percent of the cost of the software development is incurred during implementation (design, coding, and testing)[1] . Should we continue to use procedural programming methods with which we are very familiar for next generation systems, or is there a better, more efficient way to develop and maintain software?

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. However, OOP lacks mathematical formalism and its quality has not been evaluated. There is also the disadvantage that programmers need time to learn new object-oriented techniques.

This paper reports an investigation in which we used reliability models, to evaluate the possible use of OOP in the development of new measurement systems. The results indicate that OOP techniques can improve software reliability and can shorten the time required to test software.

The Object-Oriented Concept

The object-oriented concept has two basic features:

● **Abstract Data Type**

Abstraction distinguishes external behavior from internal implementation by separating data, procedures, and message-passing functions.

● **Inheritance**

The idea of inheritance is that an abstract data type can be considered as a specialized form of some more general data types, and that the more specialized type should perform all the operations performed by the more general types.

The concept can be stated as:

Object-oriented concept = **Abstract data type** + **Inheritance**

The advantages of abstract data type are information hiding and encapsulation.

Study of the Reliability of Procedural and Object-Oriented Programming

Software reliability is defined as:

The probability that software, or a constituent element thereof, will operate without the occurrence of software failures in the prescribed environment for the intended period of time.

We assume that software failures are caused by software errors.

Assume a program flow as shown in **Figure 1**.

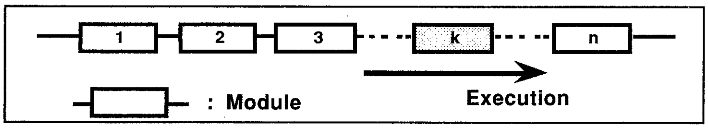

Figure 1: System under evaluation

If the reliability of the i^{th} module is R_i, the total reliability of the system R_{Total} can be determined by Equation 1.

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

We used this concept to investigate and compare the reliability and testing time of a procedure-based system and an object-based system for the case in which the k^{th} module (class) is modified.

Reliability of Procedure-Based Systems

When the k^{th} module is modified, the system reliability is given by Equation 2. In this equation, $R_{k,i}$ is the reliability of module i when the k module is modified.

$$R_{Total|k} = R_{k,1} \bullet R_{k,2} \bullet \dots \bullet R_{k,k} \bullet \dots \bullet R_{k,N} = \prod_{i=1}^{N} R_{k,i} \qquad (2)$$

It is not possible in a procedural system to specify the range of the side effects and error propagation felt by other modules when the k^{th} module is modified. Therefore, the testing time increases faster than the total number of modules[6].

Reliability of Object-Based Systems

In an object-based system, the system is implemented using the principles of encapsulation and inheritance. Classes are organized into an inheritance tree, as shown in **Figure 2.**

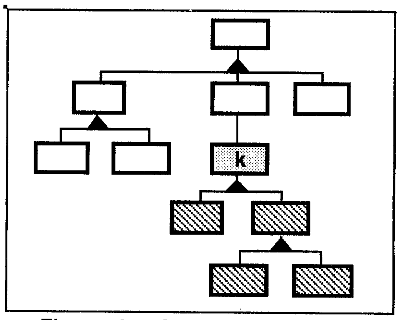

Figure 2: An inheritance tree

Let the number of descendants of the k^{th}class be L (where $L \leqq N$). When the k^{th} class is modified, the system reliability $R_{Total|k}$ is given by Equation 3.

$$R_{Total|k} = (R_{k,1} \bullet R_{k,2} \bullet \dots \bullet R_{k,k-1}) \bullet (R_{k,k+L+1} \bullet R_{k,k+L+2} \bullet \dots \bullet R_{k,N}) \bullet \{R_{k,k} \bullet R_{k,k+1} \bullet \dots \bullet R_{k,k+L}\} \qquad (3)$$

In this equation, $\{R_{k,k} \bullet R_{k,k+1} \bullet \dots \bullet R_{k,k+L}\}$ are the descendent classes of $R_{k,k}$, which are the children and grandchildren of the k^{th} class in the inheritance tree. In OOP, only the descendent classes are subjected to side effects due to modifications of class k. For this reason, classes that are not descendants of class k have the reliability they had before that class was modified. Therefore, Equation 3 becomes:

$$\begin{aligned} &R_{Total|k} \\ &= (R_1 \bullet R_2 \bullet \dots \bullet R_{k-1}) \bullet (R_{k+L+1} \bullet R_{k+L+2} \bullet \dots \bullet R_N) \bullet \{R_{k,k} \bullet R_{k,k+1} \bullet \dots \bullet R_{k,k+L}\} \\ &= \prod_{i=1}^{k-1} R_i \bullet \prod_{i=k+L+1}^{N} R_i \bullet \{\prod_{i=0}^{L} R_{k,k+i}\} \end{aligned} \qquad (4)$$

It is apparent, therefore, that when reusing software, if class k is modified, the classes affected are limited to the descendent classes of the modified class. Testing time is proportional, not to the total number of classes, but only to the number of descendent classes.

Summary of Theoretical Results

Table 1 shows a comparison of the reliability and test time for procedural programming and object-oriented programming.

Immediately after completion of a programming project, it's reliability is likely to be the same whether object-oriented or procedural programming techniques are used[7]. Because it is virtually certain that even small software systems contain bugs even after extensive testing and debugging. The reliability of an object-based system immediately after development is also given by Equation 2.

Table 1 Reliability and Test Time for Procedural and Object-Oriented Programming

(N... total number of modules, L ...the number of descendants)

		Procedural Programming	Object-Oriented Programming
Immediately after development	Reliability	$\prod_{i=1}^{N} R_{k,i}$	
	Test Time	$\prod_{i=1}^{N} T(R_{k,i})$ Proportional to 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	$\prod_{i=1}^{N} T(R_{k,i})$ Proportional to the total number of modules, N	$\{\prod_{i=0}^{L} T(R_{k,k+i})\}$ Proportional to the total number of descendants, L $L \leqq N$
	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

When software is reused in an object-based system, it is necessary to make the number of descendants (L) very small compared to the total number of modules (N), in order to achieve an advantage over procedural programming. **Essentially, it is important at the OOP analysis stage to achieve modeling with a clean, well-balanced inheritance tree which makes L as small as possible with respect to N.**

Case Study

We looked at a simulated example of a medium-scale system having 100 modules, assuming the reliability of the individual modules to be **99.99** percent. The overall system reliability is $(.9999)^{100}$, or 99.00 percent.

After modification of module k, we assumed a unit test of the module showed a reliability of **99.99** percent. We also assumed that the average reliability of the modules affected by the modification of module k is **99.90** percent. With these assumptions, we proceeded to determine the reliability of the overall system after modification of module k for a procedure-based system and an object-oriented system.

For the procedure-based system, because it is not possible to specify a limit to the side effects, Equation 2 gives the overall reliability as:

$R_{Total|k} = (.9990)^{99} \bullet (.9999)^{1} = 90.56\%$.

For the object-oriented system, the classes subject to side effects when class k is modified are only the descendants of that class. **Figure 3** shows the relationship of reliability to the total number of descendants for a system having a total of 100 classes.

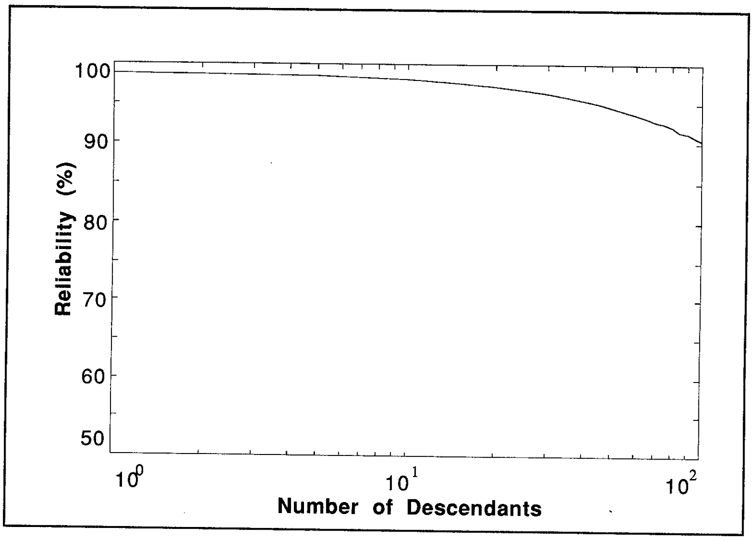

Figure 3: Reliability versus number of descendants

Assuming there are 30 descendent classes, the overall system reliability is:

$R_{Total|k} = (.9999)^{70} \bullet (.9990)^{30} = 96.37\%$

Whereas the reliability dropped **8.44** percent for the procedure-based system, it dropped only **2.63** percent for the object-oriented system .

Another case study

Software development is an incremental process. Most of the software cost is not in the initial implementation but in the enhancements that will be made during its life cycle. As they say, "software never dies; it evolves."

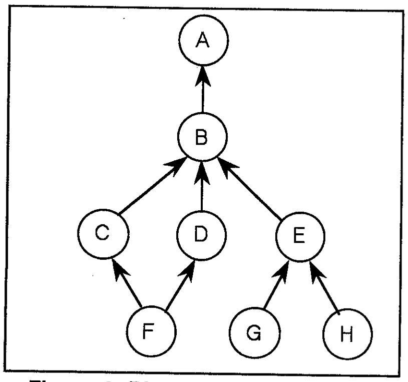

Figure 4: Directed Acyclic Graph

Let us look at the Directed Acyclic Graph (DAG) above. It can be a call graph in procedural programming and an inheritance hierarchy in object-oriented system. There is a big difference between an inheritance hierarchy and a call graph. An arrow in a call graph implies a call ("USE-A" relation), while in inheritance it implies an "IS-A" relation.

Assume that we wish to add some data to A and we need to have that data visible to H.

In the procedural method we have to change modules B and E in order to propagate data to H. But changing B implies changing C and D, and then F. Also changing E implies changing G as well. As a result we had to change all modules just because of the small enhancement that we wanted to make. Therefore, all of the modules have to be tested.

In an object-oriented system all we need to do is to add data (member data or member function) to A, and it will be readily available for H to use because our DAG is an inheritance relationship. Therefore, we have only changed two classes, and as a result, only those need to be tested.

Object-oriented programming not only protects the programmers from themselves, but it also makes upgrading and enhancement of the software (the biggest cost in software development) much easier and more reliable, and that is the beauty of it.

Conclusion

This is the preliminary report of a quantitative comparison of the test scope and testing time required for procedural and object-oriented programming. The results show that, with proper analysis, OOP can provide an improvement in reliability and shortened test time compared with procedural programming. It seems, therefore, that OOP is appropriate for implementing large-scale systems.

To apply OOP to the development of new measurement systems, it is necessary as a first step to apply object-oriented analysis to current measurement system hardware and software.

Unfortunately, OOP lacks a procedure to verify the optimum modeling method for measurement systems. Applying OOP to measurement systems requires re-engineering those systems. **Object-oriented programming alone cannot provide all the answers.**

●

Acknowledgements

We would like to thank the people of ADVANTEST Sendai Laboratories Ltd., and Hiroshi Ohura, president of ADVANTEST Corp., for his encouragement.

Technical discussions with Mr. Hira Ranga and Mr. Gordon Padwick are gratefully acknowledged. Special thanks go to Ms. Alice Salpeter and Ms. Anne Faveur for their kind checking of the technical translation.

References

1. Robert B. Grady and Deborah L. Caswell, Software Metrics: Establishing a Company-Wide Program, Prentice-Hall Inc.,1987.
2. Hajime Makabe, RELIABILITY DATA ANALYSIS, Iwanami Shoten, 1987 (in Japanese).
3. Shigeru Yamada, SOFTWARE RELIABILITY EVALUATION TECHNIQUES, HBJ Publishing, 1989 (in Japanese).
4. Jun Aoki, "Object-Oriented Software Development", FXIS, 1990 (in Japanese).
5. Peter Coad and Edward Yourdon, OBJECT-ORIENTED ANALYSIS, Prentice-Hall Inc.,1991.
6. Software testing can be assumed to exercise possible combinations - argument1, argument2 and argument3; module1, module2 and module3; path1 .1, path2.1 and path3.1 (ex. if...else...). Then it becomes a problem of combination. How long does it take to complete the test ? If it is combination of two (three, j), it takes $time(N^2)$ $(time(N^3)$, $time(N^j))$.
7. John A. Lewis, Sallie M. Henry, Dennis G. Kafura and Robert S. Schulman, "An Empirical Study of the Object-Oriented Paradigm and Software Reuse,"