ソフトウェアの理解性向上によるデバッグ時間の短縮

6H－5

内田眞司[1]，寺井淳裕[2]，武村泰宏[3,4]，阪井誠[5]，島和之[3]，松本健一[3]
[1]近畿大学工業高等専門学校，[2]株式会社管理工学研究所
[3]奈良先端科学技術大学院大学，[4]大阪芸術大学短期大学部，[5]株式会社SRA先端技術研究所

1　はじめに

ソフトウェアの保守工程では，ユーザの要求やハードウェアなどの変化に応じて，ソフトウェアを変更する必要がある．しかし保守担当者が不十分な理解のまま，一貫性のない変更を行なうと，ソフトウェアの理解が困難になり信頼性が低下する．Porterらによるコード検査の実験では，レビュー担当者が指摘した問題のうち60%が理解性の問題であった[1]．本研究では，ソフトウェアの理解性を向上させるためにソフトウェアのオーバーホールを提案している[3]．ソフトウェアのオーバーホールでは，分解されたソフトウェアを組み立てる作業を通じて各部分の働きを理解する．組立作業の過程を記録し，その作業履歴を分析することで，ソフトウェア内に潜む理解性の問題を発見する．

本稿では，ソフトウェアのオーバーホール手法によりソフトウェアの理解性が向上したかどうかを評価した実験について述べる．実験ではオーバーホール手法を施す前と施した後のプログラムについて4人の被験者にデバッグを行なってもらった結果，オーバーホール手法を施した後のプログラムの方が早くデバッグを終了することを確認した．

2　ソフトウェアのオーバーホール

オーバーホールでは，PC上に用意されたオーバーホールツールと，そのオーバーホールを行うソフトウェアの仕様を表したドキュメントが必要となる．まずツールを用いてプログラムから選択した関数内のステートメントの順序をバラバラにする．作業者は，ドキュメントを読みプログラムを理解しながら，ドキュメントの仕様に沿うように，バラバラに分解されたソースコードを正しく動作するように並べ替える．

Title: Shortening the debugging time by improvement of software understandability
Shinji UCHIDA[1], Atsuhiro TERAI[2], Yasuhiro TAKEMURA[3,4], Makoto SAKAI[5], Kazuyuki SHIMA[3] and Ken-ichi MATSUMOTO[3]
[1]Kinki University Technical College, [2]Kanrikogaku Kenkyusho,Inc, [3]Nara Institute Science and Technology, [4]Osaka University of Arts Junior College, [5]SRA Key Technology Laboratory,Inc

その際ツールではその組み立ての過程を履歴として収集している．並べ替えが完了した時点で，ソースコードの位置がもとどおりに組み立てられたかをツールにより判定する．すべてのソースコードを正しく並べ替えることができるまで作業を継続する．この時，並べ替えるのに時間を要した部分や，並べ替えができなかった部分は，作業者にとって理解が困難であったことを意味する．そのような部分を優先的に再レビュー，再設計することにより，効率的にソフトウェアを改善することができる．

3　評価実験

3.1　実験手順

実験の目的は，オーバーホールによりデバッグの効率が向上することを確かめることである．以下に実験の手順を示す．

Step1：被験者(G1,G2,G3,G4)にオーバーホールツールの説明を行なう．

Step2：4種類のフォールトが入っているプログラムを準備する(M1,M2,M3,M4)．フォールトの入った関数に対して被験者がオーバーホール作業を行う．

Step3：理解性の低い部分に対しては，コメントを挿入して理解性を向上させる(M1',M2',M3',M4')．

Step4：被験者が2つのプログラムをデバッグする(D1,D2)

表1は被験者がオーバーホールを行なうプログラムとデバッグを行なうプログラムの対応を示している．

表1　被験者へのタスクの割り当て

	G1	G2	G3	G4
オーバーホール	M1→M1'	M2→M2'	M3→M3'	M4→M4'
デバッグ1(D1)	M3	M4	M1'	M2'
デバッグ2(D2)	M4'	M3'	M2	M1

3.2　被験者

被験者は奈良先端科学技術大学院大学情報科学研究科博士前期課程の学生4人である．4人全員がC言語によるプログラミング経験が2年以上あった．

3.3 プログラムとフォールト

プログラムはC言語で書かれていて，LispのS式を構文解析しメモリ中にリストデータを生成するプログラムである[2]．約1000行，40個の関数から構成されていた．

4 実験結果

各被験者がデバッグに要した時間を表2に示す．すべての被験者において，D1に比べてD2の方がデバッグに要した時間が短くなっている．これは，被験者の慣れにより2回目のデバッグに要する時間が短くなったためと考えられる．

表2 実験結果

	G1	G2	G3	G4	平均
D1	M3 6:05	M4 4:41	M1' 1:12	M2' 0:50	3:12
D2	M4' 2:51	M3' 2:12	M2 2:29	M1 1:50	2:20

5 分析

表3はオーバーホール前と後のプログラムをデバッグするのに要した時間を示している．オーバーホールの前と後では，デバッグ時間に2時間の差がある．この差の要因として，オーバーホールの有無以外に，作業者の能力差とバグの難しさの違いも考えられる．そこで作業者の能力差とバグの難しさの違いについて仮説を立て分散分析を行なった．

仮定1：被験者の能力の差が小さい

オーバーホールの有無については，有意水準6%で有意な差があった．また，バグの違いについては有意水準14%で有意な差があった．

表3 フォールト毎のデバッグ時間

	M1	M2	M3	M4	平均
オーバーホール前	G4 1:50	G3 2:29	G1 6:05	G2 4:41	3:46
オーバーホール後	G3 1:12	G4 0:50	G2 2:12	G1 2:51	1:46
平均	1:31	1:39	4:08	3:46	

仮定2：フォールトの見つけにくさの差が小さい

オーバーホールの有無については有意水準3%で有意な差があり，作業者の違いについては6%で有意な差があった．

表4 被験者毎のデバッグ時間

	G1	G2	G3	G4	平均
オーバーホール前	M3 6:05	M4 4:41	M2 2:29	M1 1:50	3:46
オーバーホール後	M4' 2:51	M3' 2:12	M1' 1:12	M2' 0:50	1:46
平均	4:28	3:26	1:50	1:20	

デバッグ時間の差に関する3つの変動要因の中でオーバーホールの有無による差が最も大きかった．

6 まとめと今後の課題

本稿では，ソフトウェアのオーバーホールを用いてソフトウェアの理解性を向上することでデバッグ時間が短縮できることを示した．実験の結果，オーバーホール手法を施す前のプログラムのデバッグ時間と比較して，オーバーホール手法を施した後のプログラムの方がデバッグに要する時間が短くなった．デバッグ時間の差の要因として，オーバーホールの有無以外に，作業者の能力差とバグの難しさの違いが考えられるが，分散分析の結果オーバーホールの有無による差が最も大きいことがわかった．

今後の課題として，オブジェクト指向開発のソフトウェアについて同様の実験を行なう事が挙げられる．オブジェクト指向開発の再利用性が高いことを示すには複雑な実験を行なったとしても，外乱が多くて実験結果を比較しにくいという問題がある．提案している手法を用いることで理解性が向上することを確認できれば，間接的ではあるが，再利用性も向上しているといえる．

参考文献

[1] A.A.Porter, et al, "An experiment to assess the cost-benefits of code inspections in large scale software development," IEEE Trans. on Soft. Eng., vol. 23, no. 6, June 1997.

[2] M. Probst, "The lispreader library," http://www.complang.tuwien.ac.at/~schani/lispreader/

[3]寺井,内田,島,武村,松本,井上,鳥居, "ソースコードの並び替えによるソフトウェアの問題発見手法," 信学技報, ソフトウェアサイエンス研究会, Vol. 100, No. 570, SS2000-52, pp. 81-88, Jan. 22-23, 2001.