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.
AI98-82, KBSE98-52(1999-03)

デバッグ時間の短縮を目的とする二人によるデバッギングの実験的考察

伊藤 充男　森崎 修司
門田 暁人　松本 健一　鳥居 宏次

奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 奈良県 生駒市 高山町 8916-5
Tel: 0743-72-5312{atsuo-i, shuuji-m, akito-m, matumoto}@is.aist-nara.ac.jp

あらまし　本稿では，デバッグ時間の短縮を目的として，二人をデバッグ作業に投入する場合の作業者間の役割分担，および，知識交換の方法について，実験を通して考察する．実験では，二人の作業者が 4 種類の条件の下でソフトウェアに含まれる 1 個のバグの特定，および修正を行った．実験の結果，(1) バグ位置を絞り込む，(2) プログラムを理解する，という役割分担を行うことがデバッグ時間の短縮に効果的であることが分かった．また，共有ファイルを介した非同期型の知識交換が効果的であることを示す結果が得られた．

キーワード　デバッグ時間の短縮，二人によるデバッギング，作業分担，知識交換

An Experimental Evaluation of 2-Person Debugging

Atsuo Ito　Shuuji Morisaki
Akito Monden　Ken-ichi Matsumoto　and Koji Torii
Graduate School of Information Science
Nara Institute of Science and Technology

8916-5 Takayama, Ikoma, Nara 630-0101　　Tel: 0743-72-5312
{atsuo-i, shuuji-m, akito-m, matumoto}@is.aist-nara.ac.jp

Abstract　In this paper, we evaluate methods for sharing debug activity and exchanging knowledge about bug, aiming at reducing debugging time, when two persons modify the same bug cooperating. Four experiments that two person detect and modify single bug are conducted to reveal effective methods. An experiment indicates that taking either task of : (1)exploring the bug location in the source code, and (2)understanding the program is effective to reduce debugging time, and that exchanging knowledge using shared file asynchronously is effective as well.

key words　reducing debugging time, 2-person debugging, sharing tasks, exchanging knowledge

1 はじめに

稼働中のソフトウェアや出荷直前のソフトウェアに故障が発生した場合，たとえ多くの人員を投入してでも，故障の原因となるソフトウェア中のバグ（fault：ソフトウェア誤り）を速やかに発見し，除去することが望ましい．例えば，銀行のオンラインシステムのような，故障すると社会に大きな影響を与えるソフトウェアが故障した場合，できるだけ短時間でバグを除去する必要がある．また，納期が間近に迫っているソフトウェアに新たな故障が発生した場合，納期までに故障の原因となるバグを取り除かないと，発注側と請負側の双方が多大な損害を受ける可能性がある．

ただし，多くの人員をデバッグ作業に投入するにあたっては，その具体的な方法や，予想される問題点や効果について，あらかじめ十分に検討しておく必要がある．デバッグ作業においては，多くの人員を投入しても短時間でバグが除去できるとは限らない．場合によっては，一人でデバッグした方が早いこともある．作業の分担が困難であったり，作業者間のコミュニケーションに時間を要するためである．

本研究の目的は，複数の人間がデバッグを行う際の，作業の切り分け（役割分担），および，作業者間のコミュニケーションについて，望ましい方法を明らかにすることである．その第一歩として，二人によるデバッギングについて，役割分担と知識交換の両面からモデル化を行う．そして，モデルに基づいて4通りのデバッグ条件を設定し，それぞれ実験的に考察した結果を報告する．

関連する研究としては，文献 [2][3] では，多数のバグが含まれているソフトウェアから，できるだけ多くのバグを取り除くことを目的として，複数人でレビューを行う方法が紹介されている．本研究では，1個の故障が発生しているソフトウェアから，故障の原因となるバグを取り除くという作業を対象としており，想定している状況や目的が異なる．

2 二人によるデバッギング

対象とするソフトウェアとバグについて整理し，二人によるデバッギングで考慮すべき事柄を検討する．

2.1 対象とするソフトウェアとバグ

本研究で対象とするソフトウェアは，稼働中のソフトウェアや，出荷直前のソフトウェアである．つまり，結合テストや導入テストにおいて充分にテストされており，多数のバグを含まない．また，ソースコードの文法の誤りやスペルミスなどの単純なバグも含まないものとする．

ソフトウェアには1個の故障が発生しているものとする．また，簡単のため，故障の原因となるバグ（fault）は，ソフトウェア中に1個だけ含まれるものとする．つまり，複数のバグが複合して1個の故障が発生しているような状況は想定しない．なお，ここでいう故障とは，仕様書に定義されている動作と異なるようなソフトウェアの動作のことである．

2.2 対象とするデバッギング

本研究で対象とするデバッギングは，図1に示すように，二人がデバッグ作業を分担し，各人が作業の過程で，バグの発見に役立つ知識を蓄積する．そして，各人が得た知識を交換する．

本研究では，デバッグプロセスを，対象ソフトウェアやバグに関する知識を獲得していくプロセスであるとみなし，知識を積み重ねていった結果としてバグの発見に至ると考える．このプロセスにおいて二人が役割分担を行うということは，知識の獲得を分担して行うことを意味する．知識の獲得をどのように分担するかは，デバッグの効率（バグ発見に至るまでの時間の大小）に大きく影響すると考えられる．そこで，次章では，まず，デバッグ作業中に獲得される知識についてのモデル化を行う．そして，モデル上で，知識獲得の分担方法について検討する．このモデルは，知識獲得モデルと呼ぶことにする．

一方，バグの発見に至るためには，作業の結果得られた知識をお互いに交換する必要があるため，知識獲得プロセスの途中において知識交換のフェーズが生じる．知識交換の時期やタイミングの違いによっても，デバッグの効率に違いがでると考えられる．そこで，次章では，知識交換の時期やタイミングについても，特に同期性／非同期性に着目してモデル化を行う．そして，モデルを用いて知識の交換方法について検討する．このモデルは，知識交換モデルと呼ぶことにする．

3 二人によるデバッグのモデル

3.1 作業分担についてのモデル

デバッグ作業者は，ソースコードを読んだりデバッガを走らせていく過程で，対象とするソフトウェアについての様々な知識を獲得する [1]．例えば，

- 故障が発生した瞬間に，ソフトウェアのどの部分が実行されていたのか．
- 誤った（仕様と異なる）値が入っている変数はどれか．

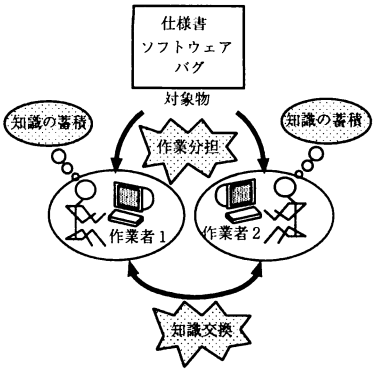

図 1: 二人によるデバッギング

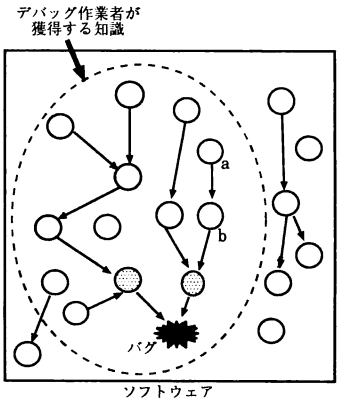

図 2: デバッグにおける知識獲得モデル

- どのような入力を与えた時に故障が発生するのか.
- モジュールXにはバグは存在しない.

などの知識である.

知識間の関係をモデル化したものを図2に示す. 図中の1個の丸は1個の知識に対応している. 網掛けされた丸は, バグの発見に直接的に関係のある知識である. 丸と丸の間の矢印は, 知識の獲得についての順序関係を示している. 図中の知識bを得るためには, それに関する知識aを先に獲得しておくことが重要となる. 例えば, (a) あるモジュールの仕様についての知識, と (b) そのモジュールの出力値が誤っているかどうかについての知識, との間には順序関係がある. モジュールの仕様が分からなければ, そのモジュールの出力値が誤っているかどうかを判断することは難しい.

図2中に点線で描かれた楕円は, デバッグ作業者が獲得する知識の範囲を表している. デバッグ作業者は, バグの発見に有用な知識も獲得するが, そうでない知識も獲得する.

二人によるデバッギングでは, 重複がないように手分けして知識を獲得することが望ましい. 図3に示すように, まとまりのある知識の獲得に各作業者を割り当てることができれば, デバッグ効率が良いと思われる. 一方, 図4 (上) のような分担は, 獲得する知識の重複が大きく, 複数人によるデバッグの効果が小さいと考えられる. また, 図4 (下) のような分担は, 獲得する知識の重複は小さいが, 作業者2はバグの発見に役立たない知識だけを獲得することになるので, 効率が悪いと考えられる.

表 1: 四つの実験条件

	役割分担	知識交換
実験A	主従型	同期型
実験B	固定	非同期型 (チャット)
実験C	役割なし	スイッチ付き同期
実験D	固定	非同期型 (共有ファイル)

3.2 知識交換についてのモデル

知識交換のタイミングには, 同期型と非同期型の両方が考えられる. 同期型の知識交換では, 一方の作業者が知識交換を要求すると, 他方の作業者は即座に反応し, 知識の交換が行われる (図5上). 一方, 非同期型の知識交換では, 知識を一時的に貯めておく「共有バッファ」が存在する. 一方の作業者が知識を伝達しようとすると, その知識は一旦共有バッファに書き込まれる. 他方の作業者は, 知識を受け取りたいときに共有バッファを参照する. そのため, 知識の伝達と受け取りが異なる時間に行われる (図5下).

同期型の知識交換モデルを図6 (上), および, 図6 (下) に示す. また, 情報交換の時期を制御できるスイッチ付きの同期型モデルを図6 (中) に示す. スイッチ付き同期型モデルでは, スイッチがONになっている間だけ知識の交換が許される.

図6 (上) の同期型モデルに当てはまる例としては, 口頭の会話による知識交換がある. 同期型の知識交換の利点は, 交換したい知識を実時間かつ短時

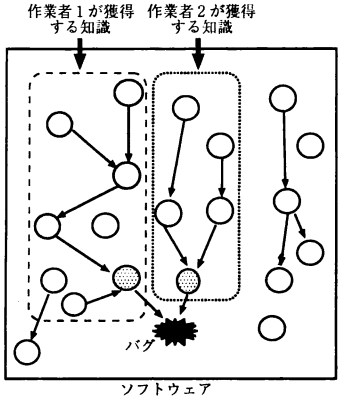

図 3: 効率の良い役割分担

間で相手に伝えたり，相手から受け取ることができることである．その反面，相手に話し掛ける際に強制的に相手の作業を中断させてしまうという欠点がある．

図 6（中）のスイッチ付き同期型モデルに当てはまる例としては，知識交換を行う時間帯を強制的に限定してしまう方法がある．例えば，1 時間おきに 10 分間だけ知識交換を行うなどの方法が考えられる．スイッチのない同期型モデルと比べた利点の一つは，知識交換により作業が中断する時間帯が限られているため，作業の集中を妨げないことである．その反面，相手から欲しい時に欲しい知識が得られないという弱点がある．

図 6（下）の非同期型モデルに当てはまる例としては，計算機上でのチャットシステムを利用する方法がある．この場合，チャットウインドウが共有バッファに該当する．一方の作業者がチャットのウインドウに知識を書き込むことで，もう一方の作業者に知識を伝えることができる．非同期型の知識交換の利点は，知識を伝達する際に，相手の作業を妨げない点である．その反面，共有バッファへの知識の書き込みに時間を要するという欠点がある．

4 実験

前章で述べた内容をもとに，知識獲得，および知識交換について，四つのデバッグ条件を設定し，二人の被験者にデバッグを行ってもらった．

4.1 四つの実験条件

実験 A ～実験 D の条件を表 1にまとめる．

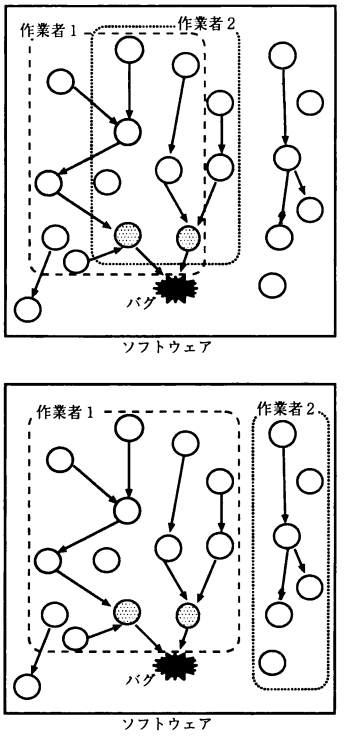

図 4: 効率の悪い役割分担

[実験 A：主従型の役割分担，会話] 実験 A では，主従型の役割分担を行った．これは，一人がリーダーシップを発揮してデバッグ作業を進め，必要に応じてもう一人に仕事を依頼する方法である．各人の役割（作業内容）は，リーダーとなる作業者によってデバッグ中に決定される．

知識交換の方法は同期型である．頻繁に会話できるよう，隣合った計算機でデバッグを行ってもらった．

[実験 B：役割固定，チャット] 実験 A では，各人の作業の決定は，リーダーによってデバッグ中に臨機応変に決定されたが，役割がうまく決定できなかったり，役割の決定に多くの時間が費やされる可能性がある．そこで，実験 B では，デバッグ前に，あらかじめ各作業者の役割を決定した．用意した役割は，プログラムのデータフローを追う，プログラムの制御フローを追う，の二つである．

知識交換の方法は，非同期型モデルを採用した．二人の被験者は，チャットシステムで全ての知識を交換する．知識交換の記録を残すことが可能であるた

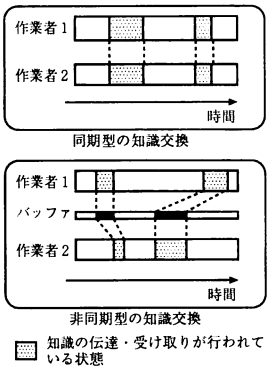

図 5: 知識交換：同期と非同期

め，会話の内容を忘れた場合にも，相手に聞き直す必要がない．

[実験 C： 役割分担なし，定期的な会話] 実験 A や B では，実験前や実験中に各人の役割を決めてもらうことにした．しかし，役割を決めるということは，自由にデバッグできなくなる，すなわち，各人が自分本来のデバッグスタイルを貫けなくなるという恐れがあり，デバッグの効率が悪くなる可能性がある．そこで，実験 C では，役割分担は決めずに，知識交換だけを行ってもらうことにした．

ただし，自由に会話してもらうと，会話による時間のロスが大きくなる可能性がある．そこで，スイッチ付き非同期型モデルを採用し，会話（知識交換）を許す時間帯と許さない時間帯を設けた．具体的には，20 分置きに会話が許されるようにした．まず，20 分間はデバッグ作業に集中してもらい，その後，自由に会話を行ってもらう．これを繰り返してもらった．

[実験 D： 役割固定，共有ファイル] 実験 D では，実験 B と同様，あらかじめ役割を決めておくことにした．一人は，プログラムを読んだりデバッガを実行させることで，バグの位置を絞り込んでいくという役割である．もう一人は，ソースコードを詳細に読み進め，プログラムを理解することに努める．どちらの被験者も，最終的にはバグを発見することを目的として作業を進めるが，前者は，バグの位置を絞り込むことを優先し，後者は，プログラムを理解することを優先する．

知識交換の方法は，非同期型モデルを採用し，プログラムのソースコードを共有バッファとした．プログラムを理解する役割の作業者は，獲得した知識

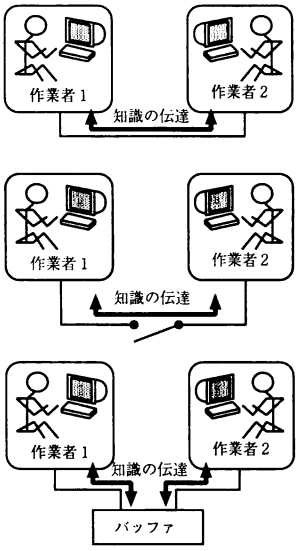

図 6: 知識交換モデル

をソースコード上の該当する場所にコメント文として書き込む．この共有ファイルは，エディタ（emacs-20.3）で閲覧・編集される．emacs では，1 個のファイルを異なる計算機のディスプレイ上に同時に表示することが可能である．

4.2 実験の環境と手順

4.2.1 実験の環境

実験では，それぞれの被験者に UNIX マシンを割り当て，計算機上でデバッグしてもらった．2 台のビデオカメラを用いて，被験者の計算機のディスプレイと被験者の音声 (発話) を記録した．

デバッグに使うツールには，コンパイラに gcc-2.8，デバッガに gdb-4.16，エディタに emacs-20.3 を選んだ．被験者は，これらのツールを普段から使用している．

被験者は，5 ～ 6 年のデバッグ経験をもつ二人の大学院生を選んだ．二人のデバッグ技術に大きな差はない．

プログラムは，4 回の実験において，それぞれ 200 ～ 400 行の規模の C 言語のプログラムとテストデータを用意した．被験者はこれらのプログラムの作成には関与していない．各実験で用いたプログラムの仕様は同一であり，酒屋の在庫管理を行うプログラムであった．ただし，プログラムごとにデータ構造

などの設計が大きく異なっており，仕様が同じことによる学習効果がデバッグ時間に与える影響は小さいと考えられる．

各プログラムにはそれぞれ1つのバグが混入しており，特定の入力に対し，誤った実行結果を出力するものであった．各プログラムのバグは，実験者が故意に混入したものである．バグの種類は，if文の条件式の誤り，ポインタの誤り，配列の添字の誤り，変数のデクリメント忘れである．

4.2.2 実験の手順

各実験では，被験者に以下の順に作業してもらった．

1. 仕様書の理解： 被験者はプログラム仕様書を読み，理解する．制限時間は設けない．
2. 故障の症状の理解： 実験者はプログラムにテストデータを与えて実行し，故障が発生していることを被験者に示す．テストデータは被験者に与えられ，デバッグ中にいつでも参照できる．
3. デバッグの開始： 二人の被験者がデバッグを同時に開始し，デバッグ時間の計測を開始する．デバッグ時間には，被験者が要求仕様書を読む時間，及び，バグの症状を見る時間を含めない．
4. デバッグの終了判定： 被験者のうち1人がバグを取り除くことができたと判断したら，もう1人の被験者と実験者に知らせる．バグを除去できたと実験者が判断したら実験を終了する．

5 実験結果と考察

5.1 実験結果

四つの実験全てにおいて，ソフトウェアの中からバグが正しく取り除かれた．各実験においてデバッグに要した時間と被験者の主観的な評価を表2に示す．表2の各列は，左から順に，デバッグに要した時間，デバッグの効率，バグの難易度，ソースコードの行数を表す．ただし，デバッグの効率とバグの難易度については，実際にバグを取り除いた被験者の主観的な評価である．

デバッグに要した時間は，実験Aが最も長かった．一方，実験Dは，A～Cと比較して半分以下の時間であった．また，被験者の感じたデバッグ効率については，実験AとBは，二人によるデバッグの効果がなく，一人でやったほうが良かったという評価「×」が得られた．実験Cでは，一人のデバッグと同程度の効率であるという評価「△」が得られた．実験Dでは，二人によるデバッグの効果が極めて高かったという評価「◎」が得られた．以上の結果から，二人によるデバッギングの効果があったのは，実験Dだけであった．

表2: 作業時間と被験者の主観的評価

	時間	効率	難易度	行数
実験A	85分	×	やや難	423行
実験B	57分	×	ふつう	285行
実験C	68分	△	やや難	193行
実験D	31分	◎	ふつう	299行

5.2 考察

4種類の作業形態での実験結果について，実験中の被験者の発話，および，実験後の被験者へのインタビューを分析した結果に基づいて考察する．

5.2.1 役割分担についての考察

[実験A：主従型の役割分担] 実験Aにおける知識獲得の様子を図7に示す．図7に示すように，被験者1が必要な知識の主要な部分の獲得を担当し，被験者2は，補助的な知識の獲得を行った．

二人の会話内容の分析の結果，被験者1から被験者2への作業の依頼は，必ずしも成功していないことが分かった．被験者1は，ごく単純な作業については，被験者2にうまく依頼することができた．例えば，あるグローバル変数の意味を調べて欲しい，というような依頼である．しかし，より複雑な作業については，依頼に失敗することがあった．例えば，被験者1は，被験者2にデバッガを使ってもらおうとしたが，被験者2は被験者1ほどにはデバッガの使い方を熟知していなかった．また，被験者1は，プログラムのある部分にバグがないかどうかを被験者2に調べてもらおうとしたが，被験者2はその部分に関係のある部分の知識がないために，いきなり作業を依頼されてもうまく作業を進めることができず，逆に被験者1にいろいろ質問することになった．

[実験B：役割固定] 実験Bにおける知識獲得の様子を図8に示す．被験者1は制御フローを追跡しており，図8に示すように，被験者2が獲得した知識をほとんど利用することなくバグを発見することができた．被験者2はプログラム上の故障発生箇所から逆向きにデータフローを追跡しており，デバッグ作業が進むにつれて，被験者1が獲得した知識を必要としていた．

2人の獲得した知識に重複は少なかったが，被験者1が得るべき知識がないと被験者2の作業が進ま

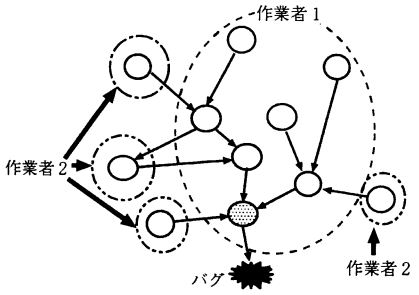

図 7: 実験 A における知識獲得

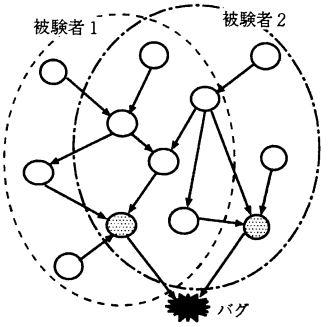

図 9: 実験 C における知識獲得

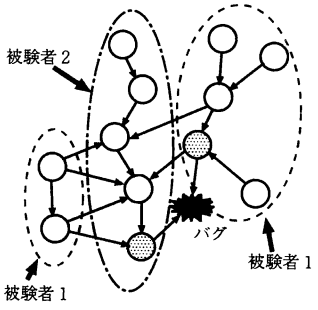

図 8: 実験 B における知識獲得

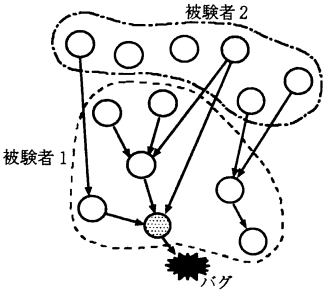

図 10: 実験 D における知識獲得

ない状態が起こった．このため，お互いの知識を上手く利用することができなかった．

[実験 C：役割なし] 実験 C における知識獲得の様子を図 9に示す．定期的な対話による知識交換では，2 人ともほぼ同じモジュールにバグがありそうだと感じていることを確認できた他は，お互いの作業にプラスになるような知識の交換はほとんどみられなかった．これは，図 9に示すように，2 人が獲得した知識に重複が多かったためと考えられる．結果として，一人によるデバッグと大差がなかった．

[実験 D：役割固定] 実験 D における知識獲得の様子を図 10に示す．被験者 1 は，バグのありそうなモジュールを絞っていったが，途中で，いくつかの構造体に関する知識が必要となった．その際に，共有ファイル中に被験者 2 が書き込んだコメントを読むことで，被験者 2 が得た知識を利用することができた．

被験者 2 は，構造体などの大域変数や，モジュールの仕様（入出力の関係）に関する知識を獲得し，ソースコード中にコメントとして書き込んでいた．被験者 2 が大域変数にコメントを付け終わり，いくつかのモジュールについての知識を獲得している段階で，被験者 1 がバグを除去して実験は終了した．

被験者 1 がバグの位置を大まかに絞り込む際には，プログラムの各部の詳細な知識は必ずしも必要でないと思われる．しかし，バグのありそうな位置をほぼ推定できた後で，実際にバグを発見するためには，バグがありそうな部分の変数の意味を詳しく知ることが重要である．被験者 2 が獲得した知識は，バグがありそうな部分で使われている変数の意味を知る上で，被験者 1 の役に立った．

[結論] 知識獲得における役割分担の点から考察すると，実験 D での役割分担の方法が有効であることが示された．

5.2.2 知識交換についての考察

各実験における会話時間（知識交換に要した時間）の割合と，交換された知識の有用性についての主観的な評価を表 3に示す．表 3の右端の「知識の有用

性」の列は，被験者1と被験者2のそれぞれの評価を示す．ただし，実験Dでは，知識の伝達が一方向（被験者2から1へ）であったため，被験者1による評価のみが記されている．

[実験A：会話] 隣合った被験者らが口頭で会話する同期型の知識交換では，一方の被験者の要求に他方の被験者が即座に応じることができるという利点が見られた．しかし，表3に示すように，知識交換に多大な時間を要した．デバッグをしている時間の半分の時間は会話していたことになる．口頭による会話は知識の伝達が容易である反面，不用意に相手に話しかけてしまう場面がたびたび見られた．また，役割分担の問題でもあるが，被験者1から被験者2への作業を依頼する際に，依頼内容の説明に多大な時間を要した．

[実験B：チャット] チャットウィンドウに表示される情報はウィンドウ内に残るため，情報の受け手側の被験者は即座に応答する必要はない．しかし，結果的には，チャットウィンドウに相手からのメッセージが表示されると，作業を中断して見てしまうことが多かった．チャットシステムは，バッファとしての役目を果たしていなかったと言える．また，表3に示すように，チャットウィンドウへの書き込みには多大な時間を要した．

[実験C：定期的な会話] 実験Cでは，知識交換をする時間帯と知識獲得の時間帯とが明確に分けられていたため，相手からの知識交換の要求によって知識獲得中に不意に作業を中断されることはなかった．しかし，相手に聞きたいときに聞けないという被験者の不満が，実験後のインタビューで聞かれた．また，前節で述べたように，有用な知識はほとんど交わされなかった．

[実験D：共有ファイル] 共有ファイルを用いて知識交換を行った実験Dでは，知識の伝達によって相手の作業を中断することはなく，非同期型の利点が見られた．プログラム理解の役割を担った被験者2は，獲得した知識を共有ファイルに書くことに多くの時間を費やしたが，そのことによって他方の被験者1の作業が待ち状態になることはなく，被験者1はデバッグ作業に集中できた．また，被験者1は，必要な時にだけ共有ファイル中のコメント文を見ることで，被験者2の知識を即座に得ることができた．

[結論] 実験AおよびBでは，知識の交換がデバッグ時間の増加につながるという悪い結果となった．また，実験Cからは，定期的な会話が効果的であるという積極的な知見は得られなかった．一方，実験Dにおける共有ファイルを用いた非同期式の知識交換の方法が効果的であったことが示された．

表 3: 実験の結果と考察 (知識交換)

	知識交換のタイミング・手段	会話時間の割合	知識の有用性
実験A	随時会話	0.5	△・×
実験B	随時チャット	0.3 ～ 0.5	×・△
実験C	定期的な会話	0.14	△・△
実験D	共有ファイル	0	◎・−

6 おわりに

本研究では，バグを短時間で除去することが優先されるソフトウェアを対象として，二人でデバッグすることによりデバッグ時間の短縮を図る方法について，実験的に考察した．実験の結果，(1)バグ位置を絞り込む，(2)プログラムを理解する，という役割分担を行うことがデバッグ時間の短縮に効果的であることが示された．また，共有ファイルを介した非同期型の知識交換が効果的であることを示す結果が得られた．

今後は，被験者を変えて同様の実験を継続する予定である．また，他の役割分担の方法や情報交換の方法についても検討する予定である．

謝辞

実験を通して貴重な御助言，御協力を戴いた奈良先端大博士前期課程 大和正武氏に深く感謝する．

参考文献

[1] K. Araki, Z. Furukawa, and J. Cheng, "A general framework for debugging," IEEE Software, pp. 14–20, May 1991.

[2] V. R. Basili et al., "The empirical investigation of perspective-based reading," Intl. J. of Empirical Soft. Eng., Vol. 1, No. 2, pp. 133–164, 1996.

[3] A. A. Porter, and L .G. Votta, "An experiment to assess different defect detection methods for software requirements inspections," Proc. of 16th Intl. Conf. on Soft. Eng., pp.103-112, 1994.