ソフトウェア工学　121－14
（1998. 11.　6）

定期的インタビューを取り入れたデバッグプロセス実験の考察

内田眞司†　工藤英男†　門田暁人††

†奈良工業高等専門学校
††奈良先端科学技術大学院大学

a0140@info.nara-k.ac.jp　kudoh@info.nara-k.ac.jp　akito-m@is.aist-nara.ac.jp

あらまし

本研究は，実験により熟練者のデバッグ戦略を明らかにすることを目的とする．実験では，デバッグ中の作業者に対して定期的にインタビューを行うことにより，作業者がプログラム中のどの辺りにバグがありそうだと考えているかを追跡した．また，作業者が参照したモジュールの履歴を記録し分析した．
その結果，デバッグの初心者と熟練者では，バグがあると推定するモジュール，および個々のモジュールの参照時間に違いがあることが分かった．

キーワード
定期的なインタビュー，デバッグプロセス，プログラム理解

An analysis of the debugging process based on changes in impressions of functions

Shinji Uchida†　Hideo Kudo†　Akito monden††

†Nara National College of Technology
††Nara Institute of Science and Technology

a0140@info.nara-k.ac.jp　kudoh@info.nara-k.ac.jp　akito-m@is.aist-nara.ac.jp

Abstract

The purpose of our study is to clarify experts' debugging strategies through controlled experiment. In the experiment, we have used a periodic interview method to analyze the module that subjects think there is a bug in it during debugging. We have also analyzed the differences among subjects in time spent for reading each module. The result of experiment showed the differences between novices and experts in both the module that subjects think there is a bug in it, and the time spent for reading each module.

key words
periodic interview，debugging process，program comprehension

1 はじめに

ソフトウェア開発において，デバッグは方法論や技法が最も確立されていない作業であると言われている [1]．効率的にデバッグを行うためには，長期間の熟練や経験を要するのが現状である．

デバッグ作業者の様子を観察，分析することにより，デバッグの効率に影響する要因を明らかにしようとする研究が行われている．文献 [2] ではプログラム理解を対象とした作業者の意図を追跡する研究が行われている．プログラムを理解する作業を行う時の行動を分析した実験 [3] では，参照するモジュールの視点の遷移が，初心者と熟練者の間で異なることを示している．

また，対象をデバッグ作業とした [4] では，デバッグ作業者は，バグの位置や内容に関する様々な仮説を立てながらデバッグを進めていることが観測されている．そして，それらの仮説を強く意識しながら，デバッグを行うか否かが，デバッグの効率に影響し得ることを示している．

本研究では，実験により熟練者のデバッグ戦略を明らかにすることを目的とする．この熟練者のデバッグ戦略を明らかにする事で，デバッグ効率の向上が期待できる．予備実験として，4 人の被験者を対象として実験を行い [5]，さらに被験者を増やして，定量的な分析を行った [6]．

筆者らは，バグの存在する場所を早く特定することがデバッグ時間の短縮に繋がると考える．つまり，デバッグの熟練者 (バグを除去できた被験者) は，初心者 (バグを除去できなかった被験者) に比べて，バグの位置をより正確かつ迅速に推定できると予想される．「どこにバグがありそうか」という判断は，インタビューにより明らかにできる．しかし，この判断はデバッグ中に頻繁に変化するかもしれないし，また実験終了後に聞いても覚えていないかもしれない．

そこで，今回の実験では，各モジュールについて，それぞれどの程度バグがありそうかを，5 分毎にインタビューすることにした．ここでのインタビューとは，アンケート用紙による聞き取り作業を指すこととする．また，各モジュール間のデバッグ作業者の注目している場所（以下，注視点と記す）の移動の分析を試みた．

大規模なソフトウェアでは，全体を読んで理解すると，多大な時間と労力を要するが，全くプログラムを理解せず，直接バグを見つけることは非常に困難であると思われる．そこで，デバッグにおいて，熟練者は必要な部分だけを熟読して理解を深め，バグを取り，初心者は必要な部分の絞り込みができず，余計なところを多く読んでしまい，なかなかバグがとれないと推測する．

本研究では，被験者がどのようにプログラムを読み，プログラムを理解し，バグを除去したかという一連の作業を，デバッグの戦略としてとらえる．デバッグの戦略は，バグ位置の特定と除去作業における時間的効率に影響を与えると予想される．この戦略を知るためには，少なくとも被験者がどこを読んだかを知る必要がある．デバッグの作業においては，命令文単位ではなくモジュール単位で，読み進めて行くと思われるので，どのモジュールを読んだかという履歴を取ることにした．

分析の結果，デバッグの初心者と熟練者では，バグがあると推定するモジュール，および個々のモジュールの参照時間に違いがあることが分かった．一般に，デバッグ作業者は，プログラム中のどこにバグがありそうかを推定しながらプログラムを読み進めており，この推定が正確であるほど効率的にデバッグを行うことができると予想される．

2 実験方法

2.1 デバッグの実験

高専と大学院の学生を被験者として，バグを除去してもらう実験を行なった．1 回の試行では，1 名の被験者に，1 個のバグの入った小規模なプログラムのソースファイルと，その仕様書を与えた．

次に，実際のデバッグ作業と同様にバグの位置は知らせずに，それが原因で発生したエラーの症状だけを詳しく伝えて，バグを除去するように指示した．プログラムや仕様については説明を与えずに，プログラムと仕様書だけで理解してもらった．そして，ソースファイルを変更するだけでなく，コンパイルしたり，プログラムを実行してテストしたりすることも行なってもらった．また実験中，被験者が「どこにバグがありそうか」と考えているかを追跡する為に，定期的にインタビューすることにした．なお，インタビューには，指定のインタビュー用紙を用い，5 分毎に実験を中断して記入してもらい，記入が終わり次第デバッグを再開してもらう事を繰り返した．

被験者には，マルチウィンドウが開けるUNIX計算機上で作業を行なってもらった．実際のデバッグ作業に近づける為に，被験者が使用するエディタやデバッガのようなツールについては特に制約を与えなかった．

本実験では，1人の被験者(A)がemacsを使用したが，その他はviを使用した．なお，全被験者がデバッガを使用した被験者はいなかった．この理由はおそらく，与えられたプログラムが小規模だったのでデバッガを用いる必要がないと被験者が判断したからである．また，行動が観測されている事を被験者に事前に知らせておいた．実験は1時間までとし，被験者がバグを修正して,「デバッグを終了した」と自発的に宣言するまで継続したが，実験開始から30分経過後，これ以上デバッグを続けてもバグを取れる見込みがないと被験者自身が宣言した場合には，実験を終了してもらった．1回の試行に要した時間は，25分から60分であった．

2.2 被験者

この実験に参加してもらった被験者は，大学院情報科学研究科1年生が1人(以下，被験者Aと記す)と高専専攻科電子情報工学専攻1年生が5人(同様に，被験者B，C，D，E，F)，高専情報工学科5年生が3人(同様に，被験者G，H，I)の合計9人である．大学院生は，C言語，Lispの経験がありプログラミングの熟練者である．高専の8人は1，2年時にPascalを，3，4年時にはC言語を修得していて，基本的に100～300行程のプログラミング経験を有している．5年時の卒業研究では，約1000行程のプログラミングを経験していた．

2.3 プログラムとバグ

被験者に与えたプログラムは，C言語で書かれている三目並べゲームで，行数は約250行，総モジュール数は15個であった．実験に用いたプログラムの階層構造図を図1に示す．図中のモジュールにふってある番号は，リスト上での出現順である．

また，モジュール間の注視点の移動の分析を容易にするために各モジュール間には15行の空行を挿入した．このプログラムには1個のバグが入っていた．事前のテストによって，これ以外のバグがない事が確認された．これはプログラム作成の際に実際に混入したバグである．挿入されていたバグは，配列の添字の間違いである．なお，バグが実際に存在したモジュールはNo.12である．

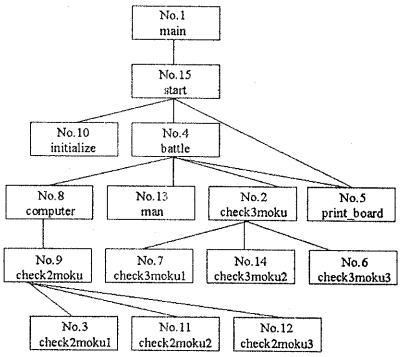

図 1: プログラムの階層構造図

2.4 データ収集

各試行において，被験者がどのモジュールを見ているかを観測するために，ビデオカメラを使用した．また，被験者が「どこにバグがありそうか」と考えているかを作業中に定期的にインタビューすることにした．

実験設備を図2に示す．実験中，被験者後方にはビデオカメラが設置されていて被験者が操作している計算機の画面をビデオで録画した．そして，デバッグ開始から終了までの間，5分ごとに作業を中断し，その都度インタビューを行った．インタビューでは，モジュール名が書いた表を被験者に与え，各モジュールについてどの程度バグがありそうかを“+3”～“-3”の7段階の評価値で記入してもらった．なお，被験者がまだ見ていないモジュールは“0”とした．実験終了後，録画したビデオを分析し，被験者が，いつ，どのモジュールを参照していたかを秒単位で記録した．

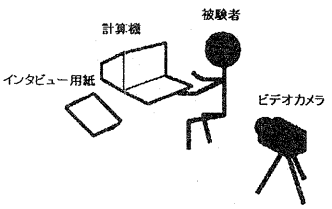

図 2: 実験設備

3 データの分析

3.1 デバッグの成否

9人の被験者のうちA，B，C，D，E，Fの6人はバグを発見することができたが，被験者G，H，Iの3人はバグを発見できなかった．

3.2 インタビュー結果

9人の被験者について，5分間隔のインタビューで得た15個のモジュール毎の「バグがありそうか」のアンケート集計を行った．バグを発見できた被験者のインタビュー結果の例を図3に，バグを発見できなかった被験者のインタビュー結果の例を図4に示す．図中，縦軸は評価値，横軸はインタビュー回数を表す．

被験者が最後まで見ていないモジュールを黒色で，「バグがなさそう」と判断したモジュールを斜線で，「バグがありそう」と判断したモジュールを網目で示した．

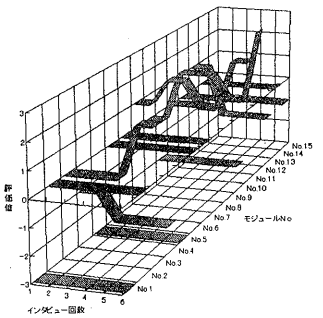

図 3: 被験者Aの評価値の変化

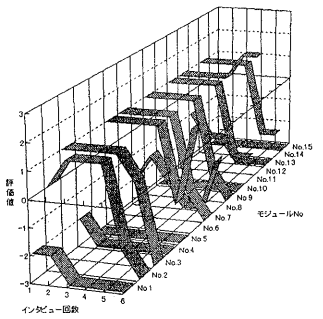

図 4: 被験者Gの評価値の変化

被験者Aの特徴は，最後まで判断していないモジュールがあることである．この特徴は，バグを発見できた被験者全員にみられた．これに対し被験者Gは，すべてのモジュールに何らかの評価を与えている．また，「バグがなさそう」と判断するモジュールの個数に違いがみられた．

次に，バグの存在するモジュール(No.12)についての各被験者のインタビュー結果を，図5に示す．図5より，9人の被験者全てが，バグの存在するモジュールについて，評価値が異なるがバグがありそうだと判断していることが分かる．

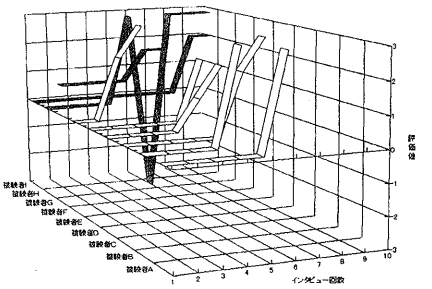

図 5: 被験者ごとの評価値の変化(No.12)

各被験者について，インタビューの評価値が0以下であったモジュールの平均個数とデバッグに要した時間の関係を図6に示す．図中，縦軸はデバッグに要した時間を表し，横軸は評価値が0以下であったモジュールの平均個数を示す．

相関係数を計算すると-0.46と負の相関がみられた．これはバグを早く発見できた被験者ほど，バグがなさそうと思う場所を増やしているからと思われる．

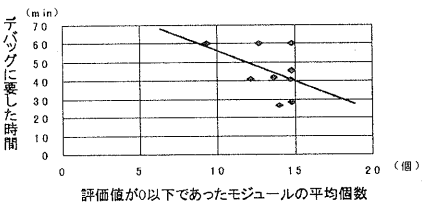

図 6: 実験時間と評価値が0以下のモジュールの平均個数

3.3 各モジュールの注視点の移動に関する結果

3.3.1 各モジュールの参照回数

各被験者について，モジュール間の注視点の移動回数を5分ごとに調べた結果を図7に示す．図中，棒グラフは視線の移動回数を表し，折れ線グラフは視線の移動の累積回数を表す．

注視点の移動回数は，最初の5分間は6人とも多く，次の5分間では，デバッグに成功した被験者(A，B，C，D，E，F:熟練者)は，注視点の移動回数が減っているのに対し，デバッグに失敗した被験者(G，H，I:初心者)は減っていない．

また，実験開始から15分以降では，後者も注視点の移動回数は減っているが，前者ほどは減っていない．累積回数のグラフからも，デバッグに成功した者より失敗した者の方が注視点の移動が多いことが分かる．

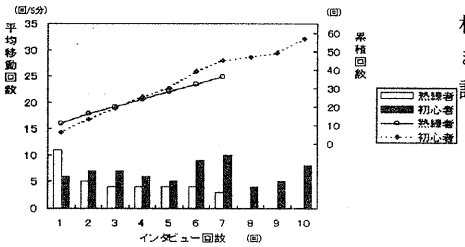

図 7: 注視点の移動回数

3.3.2 モジュール参照時間

各被験者について，モジュールの平均参照時間とデバッグに要した時間の関係を図8に示す．図中，縦軸はデバッグに要した時間を表し，モジュールの平均参照時間を表す．

相関係数を計算すると-0.55と負の相関がみられた．これはバグをはやく発見できた被験者ほど，モジュールの平均参照時間は長くなっている事を示している．

次に各モジュールの累積参照時間とデバッグに要した時間の相関係数を表9に示す．正の相関が出たモジュールは，バグをはやく発見できた被験者ほど熟読するモジュールとなる．例えば，No.12はバグが存在するモジュールなので熟読されているのがわかる．次にNo.12と関連のあるモジュールをみるとNo.8の相関が高くなっている．No.8を呼び出しているモジュールNo.4は，4つのモジュール(No8,13,2,5)を

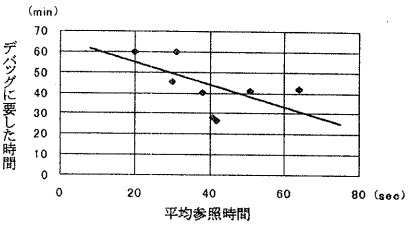

図 8: 実験時間とモジュール平均参照時間

呼び出していて，熟練者はこの4つのモジュールのうち，特にNo.8にバグがありそうと考えて熟読していると考えられる．逆に負の相関が出たモジュールは，バグをはやく発見できた被験者ほど見ないモジュールとなる．図4を見るとNo.6,No.7,No.14についてバグがありそうと判断しているので，これらのモジュールを熟読する傾向にある．この3つのモジュールは負の相関がでている．つまり，バグをはやく発見できた被験者はバグと関係の深いモジュールを熟読していると考えられる．

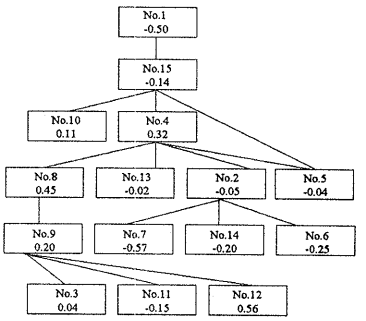

図 9: 各モジュールにおける実験時間とモジュール累積参照時間の相関係数

3.3.3 モジュールの参照履歴

被験者が，どのようにプログラムを読んで，理解しバグを除去したかを，モジュールを参照した順序を示したグラフで示す．デバッグに成功した例として，被験者Cの結果を図10に，デバッグに失敗した例として，被験者Eの結果を図11に示す．なお，図中のプロットにおいて，■は1分以上，▲は30秒以上，●は10秒以上参照していたことを表す．

図10では，インタビュー回数が増えるにつれ

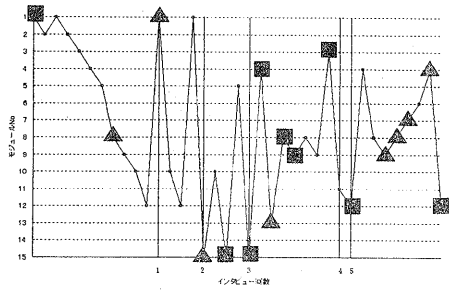

図 10: 参照順序（被験者 C）

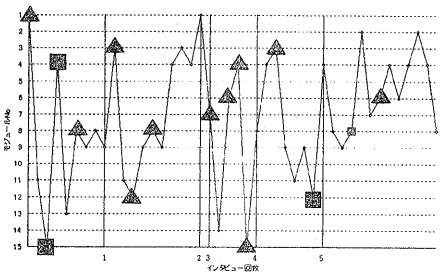

図 11: 参照順序（被験者 E）

間隔が狭くなっている．これはモジュール間の移動回数が減少している事を示していて，前半は大局的に，後半は局所的に読んでいるのがわかる．また熟読している部分がデバッグの後半に集中している．あるモジュールを熟読した後，そのモジュールにバグがなさそうと判断した時はインタビューで「バグがなさそう」と記している．バグのなさそうなモジュールの数を増やして参照すべきモジュールを絞っていく戦略を取っているのが分かる．

これに対し，図11では，モジュール間の移動回数が減少しない．つまり，読む場所を特定せずに大局的に読んでいる．なかなか熟読できずにいるので熟読回数も少ない，モジュールを熟読しないで，または全く参照しないで「バグがありそう」と判断しているモジュールがある．

4 考察

本研究では，バグの位置に関するデバッグ作業者の思考を，5分ごとのインタビューとモジュール間の注視点の移動により追跡した．ここではバグを発見できた被験者を熟練者，バグを発見できなかった被験者を初心者として考察する．

インタビュー結果から，初心者も熟練者と同様，バグの存在するモジュールについては，バグがありそうだと判断していることが分かった．初心者はバグの位置を特定することができないのでデバッグを完了できないという，当初の予想に反する結果が得られた．しかし，初心者はバグの存在するモジュール以外についてもバグがありそうと判断していた．

また，バグがなさそうだと判断したモジュールの個数に違いがあった．熟練者ほどバグがなさそうと判断したモジュールを増やしている傾向がみられた．バグがなさそうだと判断したモジュールの平均個数とデバッグに要した時間の関係には負の相関がみられた．これは，バグがなさそうと判断したモジュールを増やしながらバグの位置を絞っていると推測される．

一方，モジュール間の注視点の移動回数に，熟練者と初心者の間に違いが見られた．デバッグ開始当初は両者とも注視点の移動回数は多いが，時間経過とともに前者は半分以下に減少するが，後者はなかなか減少しない．また，各被験者のモジュール平均参照時間は熟練者ほど長い時間参照している．熟練者も初心者もデバッグ開始当初は，プログラムを制御フロー順に読んでプログラム理解に努めているために注視点の移動回数が同程度に多くなっている．

しかし，熟練者は徐々に読むべきモジュールを決めてプログラムを局所的に読み続けているのに対して，初心者は読むべきモジュールをなかなか特定できずに，もしくは，特定せずに，大局的にプログラムを読み続けているからと推測される．

また，各モジュールの累積参照時間とデバッグに要した時間の関係はそれぞれのモジュールによって異った．熟練者はあるモジュールを読んだ時に，そのモジュールから呼び出されている他のモジュールの機能が的確に想像できる．そして，バグの症状から判断して，バグに関係のありそうなモジュールを熟読していくと考えられる．

以上の考察はデバッグを効率的に行う上での重要な点を示唆している．まず，バグを発見する上ではバグがありそうなモジュールを探すだけでなくバグがなさそうと判断するモジュールを増やしていくということである．

これは，言い換えればバグのありそうな場所を絞っていく事と同じである．またプログラムを読む上では，ある程度読む場所を絞って熟読する必要がある．その時，バグの症状から判断して読む場所を絞ることが重要である．実際のデバッグ作業においてプログラム全体を熟読する事は，膨大な時間を要して効率が悪くなるのは明らかである．従って，読む場所を絞って熟読する事は非常に重要である．

5 おわりに

本研究では，プログラム中のバグを除去するデバッグ作業を観察し，デバッグの効率に影響する要因を明らかにしようとした．定期的なインタビューやビデオの記録を分析する事により，次の要因が確認された．

- バグを発見する上ではバグがありそうなモジュールを探すだけでなくバグがなさそうと判断するモジュールを増やしていくこと．
- プログラムを読む上ではバグの症状から判断して読む場所を絞り，熟読する必要があること．

今後の課題としては，デバッグプロセスのモデル化，実験データを自動的に収集できるような実験システムの開発などがあげられる．

謝辞

本研究の遂行にあたって，ご指導を頂いている奈良先端科学技術大学院大学ソフトウェア計画構成学講座の鳥居宏次教授，松本健一助教授に深く感謝致します．

参考文献

[1] Araki K., Furukawa Z., and Cheng J: “A General Framework for Debugging”, IEEE Software, pp.14–20 (1991).

[2] 吉川 :” プログラム理解過程を調べた認知実験報告 (1)”，人工知能学会研究会資料，SGI-IES-9202-7,pp.53-61(1992).

[3] 飯尾，新井，古山: ” モジュール間の視点の移動に着目したプログラム理解過程の実験と分析”，人工知能学会研究会資料、SIG-KBS-9402,pp9-16(1994).

[4] 門田，高田，鳥居 :” 視線追跡装置を用いたデバッグプロセスの観察実験”，電子情報通信学会技術報告，ソフトウェアサイエンス，SS96-5,pp.1-8(1996).

[5] 内田，工藤，門田 :” デバッグ作業者の思考を定期的なインタビューと視点の移動により追跡する実験”，情報処理学会全国大会講演論文集 (1),1C-1,pp.203-204 (1998).

[6] 内田，工藤，門田 :” 定期的なインタビューを取り入れたデバッグプロセスの実験と分析”，ソフトウェア技術者協会 ソフトウェアシンポジウム’98 論文集,pp.53-58(1998).