

4 R-5

ビジュアルデバッガのデバッグ作業効率評価

高橋健司 下村隆夫

NTT ソフトウェア研究所

1. はじめに

プログラム実行状況図形表示機能を持つビジュアルデバッガ VIPSでは、リスト構造図形表示機能によりリスト構造を扱うプログラムのデバッグ作業の効率化を図っている[1]。

リスト構造図形表示機能の効果を調べるために、VIPSおよび既存のデバッガを実際のデバッグ作業に適用し、比較評価した。評価の結果、VIPSでは作業時間を約30%、コマンド投入数を約25%削減できることが明らかになった。デバッグ作業の分析により、VIPSを用いたデバッグ作業の手順を明らかにし、VIPSによる作業時間およびコマンド投入数の削減効果を裏付けた。

2. デバッグ作業実験

(1) デバッグ対象プログラム

リスト構造を扱うプログラムをデバッグ対象とした。以下の2つのC言語記述のプログラムをデバッグ対象とした。

プログラム1 … 四則演算の式の値を求める。規模は152S。

プログラム2 … 在庫管理を行う。規模は164S。

(2) 評価対象デバッガ

ビジュアルデバッガおよびウィンドウベースの市販のシンボリックデバッガdbxtoolを評価対象とした。VIPSとdbxtoolのコマンド体系はほぼ同じであり、機能上の主要な差異はVIPSがリスト構造図形表示機能を持つことである[図1]。

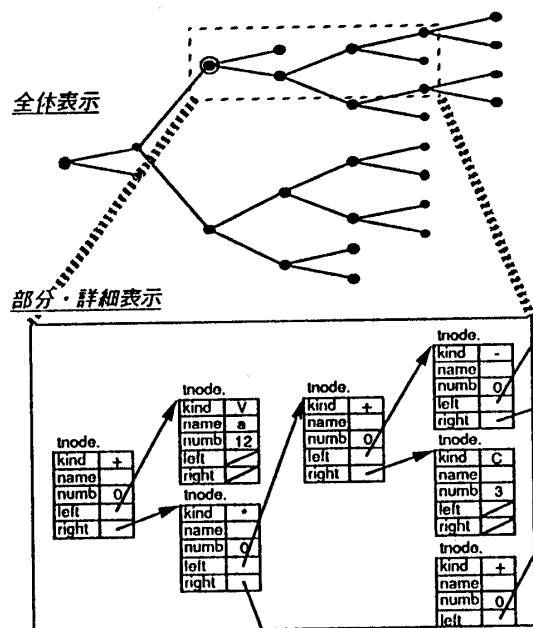


図1. VIPSのリスト構造図形表示機能

(3) 実験手順

予め埋め込まれたバグを被験者が評価対象デバッガを用いてデバッグする。そのデバッグ作業時の作業時間および作業手順を被験者自身が記録した。

デバッグ作業手順は、“ある情報をもとに次の調査対象を決定

し、デバッガの操作等の調査作業を行い、調査対象に関する情報を得るまで”を1手順として、得られた情報、判断内容、使用したコマンドおよびその実行回数を記録した。

デバッグ作業中は、作業手順の記録を行わず、作業終了後に作業時間と作業手順を記録した。作業手順の記録については、記憶に基づいて、再度同じ作業を行いつつ記録した。これは、作業時間と作業手順を同時に記録すると、作業時間を正確に計測できないためである。

(4) 被験者

C言語のプログラミングの経験が3年以上の2名(被験者A,B)。各被験者は2つのプログラムをそれぞれ6回デバッグした。毎回、1つのバグが埋め込まれおり、一人あたり合計で12個のバグをデバッグした。

3. デバッグ作業効率の分析

実験結果を基にデバッグ作業を分析し、VIPSによるデバッグ作業効率向上効果について調べた。作業効率の評価尺度として、作業時間およびコマンド投入数を用いた。

(1) デバッグ作業モデル

まず、デバッグ作業の中でデバッガに共通な部分をモデル化する。今回の実験では、“エラー箇所の候補を挙げ、それを検証する”という作業を繰り返すことによりデバッグ作業を進めている(図2)。

次に、VIPSとdbxtoolのデバッグ作業で異なる手順について考察する。VIPSとdbxtoolの作業の違いは、“リスト構造の状態を調べる”部分である。この部分について2つのデバッガを用いた作業を比較する。

・VIPSを用いた場合

リスト構造の状態を知るために、リスト構造を図形表示する(図3)。VIPSを用いた場合、被験者Aと被験者Bの作業手順の差はない。これは、今回の被デバッグプログラムでは、VIPSを用いた場合、ほぼ一回のリスト構造図形表示コマンドだけでリスト構造全体の状態を把握できたためである。

・dbxtoolを用いた場合

被験者によりデバッグ作業の手順が異なっている。

被験者Aは、リスト構造のうち比較的小規模で現在の処理対象となっている部分に注目し、この部分リスト構造の始点(参照ポインタ)を表示する方法をとっている。参照ポインタが参照する部分から全体を頭の中で再構成し、リスト構造の状態を把握している(図4)。

一方、被験者Bは、リスト構造の全ノードを表示し、紙の上で再構成することにより、リスト構造の状態を把握する(図5)。

(2) デバッグ作業時間の分析

デバッグ作業時間はVIPSでは平均6.4分、dbxtoolでは平均9.5分かかった。VIPSでは、作業時間を約30%短縮できた。

分散分析の結果、デバッグ作業時間には、デバッガ効率差、被験者およびバグ難易度の3つの要因が大きく影響することがわかった(有意水準5%)。

デバッグ作業モデルに基づき、これらの要因がどのようにデバッグ作業時間に影響を与えるのかについて考察する。

An evaluation of debugging efficiency
using Visual Debugger

Kenji Takahashi and Takao Shimomura
NTT Software Laboratories

(a) デバッグ効率差

リスト構造を変更するバグとリスト構造を変更しないバグでは、VIPSとdbxtoolの作業時間の差は、より顕著になる(表1)。dbxtoolの場合、VIPSに比べ、以下の作業が余分に必要になるため作業時間がより多くかかる。

- ・被験者A - 適切な参照ポイントの探索、参照ポイントの表示、頭の中でのリスト構造の再構成
- ・被験者B - リストノード数分の表示コマンド実行、紙の上でのリスト構造の再構成

これより、デバッグ効率差はリスト構造の表現能力の差であることがわかる。

表1. バグ種類別のデバッグ作業時間 [単位: 分/件]

	VIPS	dbxtool
リスト構造不変バグ	4.2	4.8
リスト構造変更バグ	7.1	11.0

(b) 被験者能力差

被験者Aは、被験者Bに比べ以下の点で優れているため、作業時間が短くなったと考える。能力の高い被験者の場合でもリスト構造変更バグに対してVIPSが有効であることがわかる(表2)。

- ・プログラム理解力

被験者Aは、被験者Bに比べ、プログラム読解作業が少なかった。これは、プログラム理解力の差によると考える。

- ・dbxtool使用時の参照ポイントの利用

リスト構造変更バグでは、特に作業時間の差が顕著になる。これより、被験者Aは、被験者Bに比べ、より効率的にリスト構造を把握していることがわかる。被験者Aは、参照ポイントの利用によりリスト構造を頭の中で再構成し、全てのリストノードを表示しないことにより、作業時間を短縮した。

表2. 被験者別の別のデバッグ作業時間 [単位: 分/件]

	被験者A		被験者B	
	VIPS	dbxtool	VIPS	dbxtool
リスト構造不変バグ	3.0	4.7	5.3	5.0
リスト構造変更バグ	6.2	8.2	8.0	13.8

(c) バグ難易度

バグの難易度は、エラー出力とエラー箇所との関係を推測するのがどれくらい難しいかに大きく依存している。つまり、バグの難易は以下のように分類される。

- ・デバッグが比較的困難なバグ

エラー箇所がエラー出力に直接反映されないバグは、デバッグが比較的困難である。たとえば、エラー箇所のコールスタックが深い場合や、出力されない内部変数のバグ等。

- ・デバッグが比較的容易なバグ

エラー出力とエラー箇所との関係が直接的である場合、デバッグは容易である。たとえば、出力する変数の値を直接変更するバグ、出力を直接制御する変数や式のバグ等。

デバッグ困難なバグはdbxtoolと同様 VIPSでもデバッグ作業時間が長い(表3)。これは、VIPSでもエラー出力とエラー箇所との関係の推測を直接支援していないためである。デバッグ困難なバグのデバッグを支援するためには、プログラムの実行履歴を利用した支援機能が必要である。

表3. バグ難易度別のデバッグ作業時間 [単位: 分/件]

	VIPS	dbxtool
デバッグ困難バグ	8.9	12.9
デバッグ容易バグ	4.6	7.0

(2) コマンド投入数の分析

コマンド投入数はVIPSでは平均16回、dbxtoolでは平均22回多かった。VIPSでは、コマンド投入数を約25%短縮できた。また、分散分析により、コマンド投入数にはデバッグ効率差(有意水準1%)およびバグ難易度(有意水準5%)の2つの要因が大きく影響することがわかった。

デバッグ作業モデルを基に、これらの要因がどのようにコマンド投入数に影響するかについて考察する。

(a) デバッグ効率差

コマンドを表示、実行、検索の3つに分類して、VIPSとdbxtoolにおけるコマンド数を比較すると、実行、検索コマンド数はほぼ同じであるが、表示コマンド数はVIPSの方が少ない。特にリスト構造変更バグの場合、表示コマンド数の差は大きい(表4)。これはVIPSのリスト構造図形表示機能を用いることによりほぼコマンド1回でリスト構造の状態を常時把握できることによる。

表4. コマンド種別のコマンド数 [単位: コマンド数/件]

	VIPS			dbxtool		
	表示	実行	検索	表示	実行	検索
リスト構造不変バグ	1.7	10.2	0.0	4.7	12.3	0.2
リスト構造変更バグ	2.4	14.6	0.2	9.1	13.6	0.3
平均	2.3	13.5	0.1	8.0	13.3	0.3

(b) バグ難易度

コマンド投入数はデバッグ作業時間と同様の傾向があり、VIPSとdbxtoolでコマンド投入数が多かったバグは共通している。

4. おわりに

今後、実用のプログラムのデバッグ作業についても分析を進め、デバッグ作業モデルの精密化およびVIPSの機能拡充を検討する。

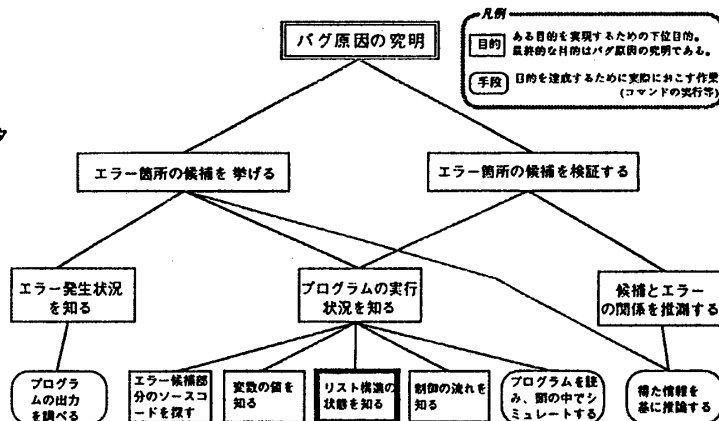


図2. デバッグ作業モデル

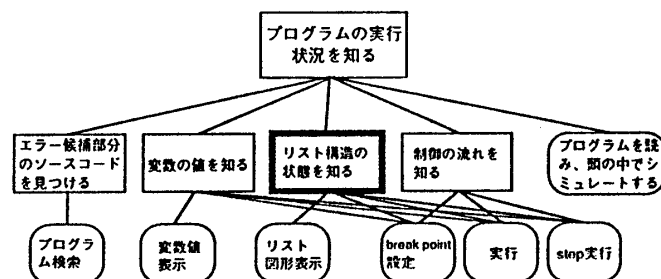


図3. VIPSを用いた場合のデバッグ作業モデル

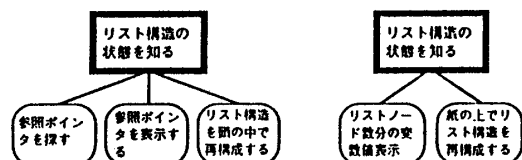


図4. dbxtoolを用いた場合のデバッグ作業モデル(被験者A)



図5. dbxtoolを用いた場合のデバッグ作業モデル(被験者B)

謝辞

研究を進めるにあたり貴重な助言を頂いたソフトウェア研究所ソフトウェア基礎技術研究部磯田定宏部長に深謝致します。

[参考文献]

[1] 高橋, 下村, 磯田, "既存のデバッグツールを利用した高速なビジュアルデバッグ方式", 2M-3, 情報38 全大, 1989, pp.1235-1236.