# ソフトウェア理解性尺度に関する一実験*
## -Cプログラムへの適用-

5S－1

友本隆義　廣田豊彦　橋本正明†
九州工業大学　情報工学部‡

## 1　はじめに

ソフトウェア保守にかかる時間のほとんど半分は，ソフトウェアの理解のために費やされる．このことは，ソフトウェアの理解性がソフトウェア保守コストに大きな影響を与えることを意味している．もしもソフトウェアの理解性を予測することができるならば，ソフトウェアの保守性も評価することが可能である．我々はプログラム理解性を予測する手段として，余波複雑度[1]を提案している．今回，余波複雑度の有用性を示すため，Cプログラムのデバッグ実験を行なった．その結果，余波複雑度は他の尺度と比較して理解性と深い関連があることが分かった．本論文では実験内容及びその結果を示し，余波複雑度と理解性との関連について考察する．

## 2　余波分析と余波複雑度

余波分析[1]とは後方余波分析と前方余波分析を合わせて言う．後方余波分析は与えられた節点$n$で参照されている変数$v$を解析し，$n$よりも前に実行され$v$の値に影響を与える節点$D(v,n)$を見つけ出す．前方余波分析は節点$n$で定義されている変数$v$を解析し，$v$の修正の結果として値が変わるかも知れない変数を使用している節点$U(v,n)$を見つけ出す．後方余波が影響する節点の集合$R(v,n)$及び，前方余波が影響する節点の集合$F(v,n)$はそれぞれ次の式を満足する．

$$R(v,n) = D(v,n) \bigcup \sum R(w,m),$$
$$where \forall m \in D(v,n), \exists l, m \in U(w,l)$$

$$F(v,n) = U(v,n) \bigcup \sum R(w,m),$$
$$where \forall m \in U(v,n), \exists l, m \in D(w,l)$$

余波複雑度は，プログラム中の各変数に対して後方余波分析及び前方余波分析を行ない，その時走査した節点の合計数で定義される．したがって，余波複雑度はソフトウェア断片を変更するときの潜在的な副作用の大きさを示すと考えられる．

## 3　比較対照の尺度

ソースコード行数はもっとも単純な尺度であるが，プログラムが小さければ理解が容易であるとは言えない．そこでいろいろなプログラムの複雑度の尺度が提案されている．本実験では余波複雑度との比較対照として，McCabeの閉路複雑度[2]及びHalsteadのソフトウェア科学計量法[2]を用いた．以下にその尺度について述べる．

閉路複雑度はプログラム中の独立なパス数に等しい．グラフ$G$の閉路複雑度$CC$は以下の式で計算される．

$$CC(G) = (\text{枝数}) - (\text{節点数}) + 1$$

Halsteadのソフトウェア科学計量法は演算子とオペランドの数の計量に基づいており，プログラムの量$V$は以下の式で計算される．

$$V = N \log_2(n)$$

ただし

$N$ = 演算子の総数 + オペランドの総数

$n$ = 演算子の種類の数 + オペランドの種類の数

である．

## 4　実験

### 4.1　実験方法

我々は余波複雑度がプログラム理解に必要な労力を推定するために使えるかどうかを調べるために50行，100行及び150行程度の小規模のテスト用Cプログラムをそれぞれ2個ずつ用意した．6個のテストプログラムは以下のようなものである．test1.cは環状バッファへの入出力を行なうコード，test2.cは与えられた文字列中の特定の文字を入れ換えるコード，test3.cは数式を判断するコード，test4.cは与えられた文字列を指定通りに書き換えるコード，test5.cはスタック操作のコード，test6.cは15ゲームをコード化したものである．これらのそれぞれのプログラムにいくつかのバ

*An experiment on software understandability metrics
†Takayoshi Tomomoto Toyohiko Hirota Masaaki Hashimoto
‡Kyushu Institute of Technology

グを加え，プログラマがバグを修正するのにかかる時間を測定した．

バグについては，各プログラムに加えるバグに修正のしやすさの点で大きな差があってはならない．そこで，バグの種類 [3]，数について各プログラムとも同等にするよう努めた．またそのバグは，プログラムの内容を理解しなければ修正することができないものにした．

実験は 3 回に分け，1 回目は 50 行，2 回目は 100 行，3 回目は 150 行のプログラムについて実験を行なった．参加した 13 人のプログラマ (学生) はいずれも C の経験者である．彼らはそれぞれの回において 2 つのプログラムを順に渡され，プログラムを理解しバグを発見して修正しなければならない．プログラムが正しく修正されたかどうかは，用意された入力に対してそのプログラムが正しい出力を生成するかどうかで判定した．プログラマには回答の順序を変更することは認めなかった．

## 4.2 実験結果

実験の結果を表に示す．表中ののべ時間は，そのプログラムに対するプログラマ全員の修正時間の合計である．また ripple は余波複雑度の累計を，ripple/Node は 1 節点当たりの余波複雑度を，ripple/Def は 1 変数定義当たりの余波複雑度をそれぞれ表している．プログラム修正に要した労力は，'test1.c"test3.c"test2.c"test5.c"test4.c"test6.c' の順に大きくなっている．

各複雑度や余波複雑度を節点数あるいは変数定義数で正規化した値と，プログラム修正にかかった労力との関係に着目すると，正の相関が見られるのは 1 変数定義当たりの余波複雑度だけであった．すなわち McCabe，Halstead と余波複雑度の累計及び 1 節点当たりの余波複雑度にはそのような相関関係は見当たらなかった．

## 5 おわりに

ソフトウェアの保守性の評価が可能になれば，保守の予想コストの見積もりも可能になる．また，どのようなソフトウェアが保守性が高いかを知ることができるため，ソフトウェアの製作段階から保守性の高いものを作成する努力も可能になる．このように，保守性尺度を自動的に確立するツールは有益である．

プログラム読者がプログラムの動作や機能を理解しようとするときの思考は，余波分析の動作に類似している．我々が余波複雑度がプログラムの理解性を計測するのに使えると考える理由がここにもある．

我々は実験によって，余波複雑度がソースコードの行数，McCabe の閉路複雑度，Halstead のソフトウェア科学計量法などの標準的な尺度よりもプログラムの理解度に密接に関係していることがわかった．今後さらに，データ分析や追加実験を行ない，保守コスト (バグの修正時間) と評価尺度との関係について研究を進める予定である．

## 参考文献

[1] 廣田豊彦，東木勝治，Overstreet, C. M.，橋本正明，Cherinka, R.: ソフトウェア理解性尺度に関する一実験，電子情報通信学会技術研究報告 (知能ソフトウェア工学)，Vol. 94, No. 357, pp. 41–47 (1994).

[2] McClure, C.: *The Three Rs of Software Automation: Re-engineering, Repository, Reusability*, Prentice-Hall (1992). (邦訳：『ソフトウェア開発と保守の戦略－リエンジニアリング・リポジトリ・再利用－』ベスト CASE 研究グループ 訳，共立出版，1993 年).

[3] Beizer, B.: *Software Testing Techniques*, Van Nostraud Reinhold (1990).

| | test1.c | test2.c | test3.c | test4.c | test5.c | test6.c |
|---|---|---|---|---|---|---|
| のべ時間 (分) | 730 | 1020 | 784 | 1357 | 1262 | 1659 |
| 平均時間 (分) | 56 | 81 | 60 | 104 | 97 | 131 |
| 標準偏差 | 24.2619 | 44.703 | 32.5612 | 39.997 | 52.4968 | 65.2452 |
| line | 61(50) | 46(50) | 115(100) | 120(100) | 142(150) | 144(150) |
| ripple | 46 | 303 | 491 | 2278 | 639 | 1431 |
| ripple/Node | 3 | 22 | 8 | 58 | 22 | 51 |
| ripple/Def | 5 | 30 | 13 | 72 | 34 | 74 |
| Halstead | 1116.47 | 1216.45 | 2464.86 | 3776.39 | 3309.48 | 4891.29 |
| McCabe | 8 | 7 | 29 | 26 | 19 | 30 |

表 1: 実験結果