B-014

関数間のデータのやり取りとバグ混入の関係性の考察
A study of the relationship of data exchange between functions and bug inclusion

高崎 椋太† 猪股 俊光†
Ryota Takasaki Tosimitsu Inomata

1. はじめに

ソフトウェア開発において，レビューやテストの工数は工数全体の大きな割合を占めることから，これらの作業を効率的に行うことは，工数削減のために重要である．そこで，バグの混入が予測される箇所が特定できれば，レビューやテストの効率化が期待される．そのためには，ソフトウェアの品質を定量的に評価する指標が必要であり， CBO メトリクスは，Imagix 4D や Understand などのソースコード解析ツールに利用されている[1][2].

CBO メトリクスは，結合度という観点により，ソースコードの複雑さを評価するソフトウェアメトリクスである．結合度とは，関数やクラスなどのモジュール内部で呼び出している外部モジュールの数によって表され，その値が高いほどソースコードの複雑度が高いとされる．モジュールの呼び出し時には，モジュール間でデータのやり取りが行われることがあるが，CBO メトリクスでは，それらのデータのやり取り内容については考慮していない．外部モジュールの呼び出し時にデータのやり取りを行う場合と，行わない場合では，ソースコードの複雑度に与える影響が異なる可能性がある．複雑度が高いソースコードほど，バグの混入が予測されることから，複雑度をより正確に測るためには，呼び出し時のデータのやり取りの仕方も考慮すべきであると考えられる．

そこで，本研究では C 言語に代表される関数間のデータのやり取りで計算が行われるソースコードを対象として，関数の呼び出し時のデータのやり取りが，ソースコードの複雑度とどのような関係性があるのかについて定量的に考察した．

2. 結合数とソースコードの複雑度

2.1 結合数

ソースコード内の関数の集合を F，関数 $f \in F$ のなかで呼び出されている関数の種類数を「結合数 $n(f)$」とする（再帰呼び出しは除く）．さらに，ソースコード全体の結合数を *Ncall* とする．

$$Ncall = \sum_{f \in F} n(f)$$

関数の集合 F = {*funA, funB, funC, funD*}における関数間の呼び出し数と n(f)の値の例を表 1 示す.

表 1

	funA	*funB*	*funC*	*funD*	*n(f)*
funA		1	0	1	2
funB	0		0	0	0
funC	3	2		1	3
funD	1	0	0		1

2.2 データのやり取りの分類

ソースファイルにおける関数間のデータのやり取りを，関数呼び出しに伴う値の受け渡しと定め，「戻り値によるデータのやり取り」と「引数の参照渡しによるデータのやり取り」の２種類に分類する。本研究では，C ソースコードを静的に解析することでこれらの出現箇所を特定する．

2.2.1 戻り値によるデータのやり取り

戻り値によるデータのやり取りとは，関数 *funX* が *funY* を呼び出して，*funY* の戻り値を利用する操作である．ソースコードにおけるこのようなデータのやり取りの総数を *Nreturn* とする．具体的には，ソースコード中の次の I,II の構文が該当する．

I. 代入文の右辺に関数呼び出しが現れる．
II. 条件式の判定に関数呼び出しが現れる．

2.2.2 参照渡しによるデータのやり取り

参照渡しによるデータのやり取りとは，関数 *funX* が *funY* を参照呼び (call by reference)する操作である．ソースコードにおけるこのようなデータのやり取りの総数を *Nptr* とする．具体的には，ソースコード中の次の I の構文が該当する．

I. 関数呼び出しの実引数の値がアドレスで，呼び出された関数の仮引数がポインタである．

2.3 ソースコードの複雑度と変更数

複雑度が高いソースコードほど，開発や保守作業のなかでバグが混入し，修正の対象になる可能性が高くなるとの予想から，ソースコードが修正された回数に着目することとした．すなわち，修正された回数が多いほど，複雑度が高くなるとして，ソースコードの修正回数をソースコードの複雑度 *Complex* とする．

2.4 ソースコードの複雑度の評価方法

CBO メトリクスでは，2.1 で定義したソースコードの結合数のみを用いて，ソースコードの複雑度を評価している．

これに対して，本研究では，ソースコードの結合数に加え，関数間のデータのやり取りにも着目して，ソースコードの複雑度を評価した．

†岩手県立大学大学院ソフトウェア情報学研究科
Graduate School of Software and Information Science, Iwate Prefectual University

3. 評価実験

3.1 実験方法

2 章で述べた定義に基づき，GitHub で公開されているソースコードを対象に，ソースコードの結合数とデータのやり取りの回数を示す *Ncall,Nreturn,Nptr* の値と，ソースコードの複雑度の関係性について，回帰分析を用いて数量的に分析する．

実験の手順を以下に示す

① プロジェクト内の過去のコミットを解析することで，すべての C ソースコードの変更数について調査し，その値を *Complex* とする

② 第 2 節で述べた定義に基づき，プロジェクト内の各ソースコードにおける *Ncall,Nreturn,Nptr* を構文解析によって求める．

③ 各 C ソースコードについて目的変数を *Complex* としたとき，説明変数として *Ncall* のみを用いた場合と，*Ncall,Nreturn,Nptr* を用いた場合の回帰分析を行い，結果を比較し，考察する．

3.2 実験結果

3.2.1 データセットの生成

実験では，GitHub で公開されているオープンソースのソフトウェアとして zlib を取り上げた．zlib は，テストコードを除き合計 15 個のファイルからなり，それらのなかに計 190 個の関数が含まれている．

zlib のファイルを解析することによって，*Ncall, Nreturn, Nptr, Complex* の各値を数え上げ，データセットを生成した．データセットのうち，*Complex* の値が大きい 4 つのファイルにおける各値の例を表 2 に示す． 一列目の値はファイル名である．

表 2

ファイル名	*Nreturn*	*Nptr*	*Ncall*	*Complex*
trees	3	24	27	52
iflate	48	55	72	68
inftrees	0	0	4	83
deflate	51	115	138	126

3.2.2 回帰分析

3.2.1 によって得られたデータセットを用いて回帰分析を行う．

① *Complex* を目的変数，*Ncall* を説明変数としたときの回帰分析を行い，その結果を E1 とする

② *Complex* を目的変数，*Ncall,Nreturn,Nptr* を説明変数としたときの回帰分析を行い，その結果を E2 とする．

実験の結果と，その比較を表 3 に示す．

表 3

	重相関 R	重決定 R2	補正 R2	標準誤差
E1	0.69626	0.48576	0.44515	22.395
E2	0.84123	0.70767	0.62794	18.339
E2/E1	+20%	+46%	+40%	−18%

4 考 察

本実験では，GitHub で公開されているオープンソースのソフトウェアである zlib において，結合数だけでなく,データのやり取り数を用いてソースコードの複雑度を評価する手法が有効であるかどうかを検証した．3.1 の結果により，以下のことが分かった．

① ソースコードの結合数と，データのやり取り数を表す *Ncall,Nreturn,Nptr* はソースコードの複雑度と強い相関がある

② E2 による回帰式は E1 による回帰式に比べ，重相関 R や補正 R2 の数値が高い．よって *Ncall* だけでなく *Nreturn, Nptr* を用いることで，より高い精度でソースコードの複雑度の予測ができる．

以上により，zlib において，*Ncall* の値だけではなくデータのやり取り内容に注目することで，より正確にソースコードの複雑度を表現できることが分かった．このことから，データのやり取り内容に注目してソースコードの複雑度を評価する手法は有効だと考えられる．

今回、得られた知見は zlib に対する実験によるものである．そのため，上記手法について，様々なプロジェクトに対して検証を進めていくことで，プロジェクトの構造などが予測精度に与える影響などを検証していく必要がある．

4. おわりに

本研究では，関数の呼び出し時のデータのやり取りが，ソースコードの複雑度とどのような関係性があるか，ならびに，これらの関係性がソースコードの複雑さの評価に有効かどうかについて考察した．公開されているソースコードを対象として，関数呼び出しの数や内容によって分類した各データのやり取りの数と，ソースコードの複雑度（変更回数）の関係性について，回帰分析した結果，関数呼び出し時のデータのやり取り内容は，ソースコードの複雑度と強い相関があることが明らかになった．このことから，関数呼び出し時のデータのやり取りをもとにソースコードの複雑度を評価する手法は有効であると考えられる．

今後は，対象とするソースコードの種類を増やしながら，ソースコードの複雑度をもとにしたバグ混入の予測が，ソフトウェア開発の工数削減に与える影響についても検討していきたい．

謝 辞

本研究の一部は JSPS 科学研究費補助金(課題番号 22K11957) の援助を受けている．

参考文献

[1] Imagix Corp, “Imagix 4D”, https://www.imagix.com/index.html.

[2] TechMatrix, “Understand”, http://www.techmatrix.co.jp/product/understand/index.html.