3L-1

ソフトウェア理解支援のためのデータ構造の重要度評価手法

竹治 勲[†]　大久保 弘崇[‡]　粕谷 英人[‡]　山本 晋一郎[‡]　齋藤 邦彦[§]

[†]愛知県立大学大学院情報科学研究科　[‡]愛知県立大学情報科学部
[§]滋賀大学経済学部

1　はじめに

1.1　背景

ソフトウェアの開発・保守プロセスにおいて，機能維持や拡張といった作業はソフトウェア理解が前提となっている．そのため，ソフトウェア理解の重要性は高い．ソフトウェア理解は一般的に，大局的な流れを追った後，詳細の動作を見ていく．大局的な流れを見る場合，機能やモジュールといった単位でソフトウェアの相互関係を把握する．しかし，近年のソフトウェアの大規模化・複雑化により，ドキュメントやソースコードが増加しているため，ソフトウェアの全てを理解するにはコストや時間がかかる．機能レベルでの理解支援の方法に Feature Location があるが，多くの機能から構成されるソフトウェアに対しては，どの点から着目すればよいかが分かりにくい．

そこで，ソフトウェアの本質的な機能に注目することができれば，大局的な動作の理解に有効であると考えられる．

1.2　目的

我々の研究の目的は，プログラム中のデータ構造に対して重要度を評価する手法を提案することである．データ構造は，大規模化により大局的な理解が困難になったプログラムの全体構造や相互関係を理解するのに有用な指標となる．理解の対象であるプログラムの本質的な機能に関わるデータ構造を重要なデータ構造とし，本手法を用いて重要なデータ構造を提示する．本手法を用いることで，大規模なソフトウェアでもプログラム中の本質的な部分に注目できる．

理解対象は，C プログラムとする．C 言語では，データ構造を表現するために構造体が利用される．このため本論文において**データ構造**とは，**構造体**のことを指す．

1.2.1　重要なデータ構造

本論文において，重要とは，本質的な機能に関わっていることを指す．また，本質的な機能とは，ソフトウェアが対象とする特定領域の問題に関わる機能を指す．本質的な機能の具体例は，圧縮・展開ソフトは圧縮と展開，コンパイラはコンパイル，OS はプロセス管理と資源管理が挙げられる．したがって，重要なデータ構造とは，本質的なデータ構造のことを意味する．

一方，入出力処理，ファイル処理，エラー処理といった処理は，多くのソフトウェアで行われるため，本質的な機能ではない．

1.2.2　重要な構造体の例

重要な構造体の例として，圧縮・展開ソフトウェアの一つである gzip-1.2.4 (以下 gzip) の例を示す．gzip では次に示す 6 種類の構造体型が定義されている (表 1).

構造体型	役割	重要
tree_desc	圧縮 (データ)	○
ct_data		○
config	圧縮 (設定)	○
huft	展開	○
stat	ファイル処理	
option	オプション処理	

表 1: gzip の構造体型

先ほど述べた重要なデータ構造の定義から，重要なものには，tree_desc，ct_data，config，huft が該当し，重要でないものは stat と option となる．

2　重要度評価

2.1　動的情報の利用

ソースコードのような静的情報は，網羅的に結果を得ることができるが，入力に依存した処理を考慮することができない．動的情報は，入力に依存した情報を扱えるが，入力サンプル全てを得ることは難しい．

本論文では，動的情報から重要度を計測することを考える．動的情報は，実際に利用されたという事実から，重要度に利用できると考えられる．

2.2　コマンドラインオプションによる動作の変化

コマンドラインオプションにより動作を切り替えることができるプログラムでは，オプションによって大きく動作が切り替わることが多い．プログラム実行時にアクセスされる変数は重要である可能性がある．したがって，様々なオプションで実行した際に利用される変数は，変数のアクセス回数とその型の重要度との間に相関があると推測できる．

2.3　入力による動作の変化

プログラムの動作は，入力の違いによっても変化する．この入力による変数のアクセス回数の違いは，重要度に影響すると考えられる．変数のアクセス回数と入力サイズとの間の相関関係の違いが，データ構造の性質を分類する目的に利用できると予想される．

3　アクセス回数カウントに対するアプローチ

2 節の方法で重要度を評価する場合，プログラム実行時の変数のアクセス回数を取得する必要がある．

To Measure Importance of Data Structures in C Program for Software Comprehension

[†]Isao Takeji, Graduate School of Information Science and Technology, Aichi Prefectural University.
[‡]Hirotaka Ohkubo, Hideto Kasuya, Shinichiro Yamamoto, School of Information Science and Technology, Aichi Prefectural University.
§ Kunihiko Saito, Faculty of Economics, Shiga University.

3.1 デバッガによるカウント

実行時の変数のアクセス回数をカウントする場合，gdb では，ウォッチポイントを用いる．ウォッチポイントとは，監視対象の変数に何らかの操作が行われたときに停止するブレイクポイントである．

gdb の場合，ウォッチポイントは優先的にハードウェアによるウォッチとして設定される．しかし，ハードウェアウォッチは，アーキテクチャに依存するため，設定できる数に限りがある．したがって，構造体のようにサイズが大きい変数を監視する場合は，

- 監視する変数をプログラムの実行毎に設定し直す
- メンバ変数を監視し，構造体型変数のアクセス回数を求める

といった方法を取る必要がある．そのため，現実的な時間で大規模なプログラムの構造体型変数全てのアクセス回数を求めることは難しい．

3.2 アスペクト指向を利用したログ情報の取得

ログ取得のアスペクトをソースコード中に埋め込むことで，アクセス回数のを取得することができる．ログ情報から正確にアクセス回数を求められる．しかし，ソースプログラムを解析し各変数に対してアスペクトを埋め込む必要があるので，手間がかかる．

3.3 Sapid の動作状況モデルによる解析 [1]

Sapid[2] で提案されている動作状況モデルを利用することで，プログラムの開始から終了までの変数のアクセス回数や関数呼び出しなどの動作状況を全てリポジトリに格納するため詳細な情報が得られる．本論文は最終的なアクセス回数を求めたいが，[1] は解析のための時間がかかるため，コストが高い．

3.4 本ツールのアプローチ

本論文では，効率と正確さのトレードオフから，より効果的にアクセス回数を求めるためのツールを実装した (図 1)．本ツールは Sapid[2] による静的解析と gcov[3] による動的解析を用いる．Sapid は，ソースプログラム上に出現する変数とその位置を得ることができる．gcov はコンパイル時に行の情報を埋め込むことで，プログラム実行時のラインカバレッジを得ることができるツールである．これにより，どのファイルの何行目を何回実行したかという情報を得る．

Sapid による静的解析結果と gcov による動的解析結果の XML ファイルを入力とし，変数の位置情報と実行された行位置を照合することで変数のアクセス回数をカウントする．

本ツールは，メンバ変数へのアクセス回数はカウントしない．構造体のアクセス回数をカウントすることを考慮すると，構造体型変数のカウントをすればよいため，メンバ変数のカウント情報は必要ない．また，3 項演算子のように行の中で一部実行されない箇所がある場合も，アクセスしたものとしてカウントするが，近似できていると考えられる．

変数のアクセス回数をカウントするのに，gdb のように何度もプログラムを実行する必要がない．

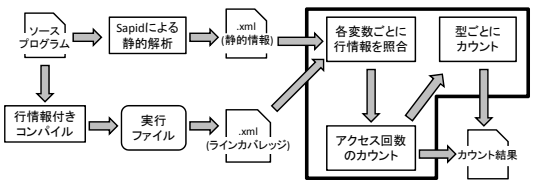

図 1: ツールの概観

4 適用実験

gzip-1.2.4 を対象に，重要度評価の実験を行った．プログラム中の各構造体型変数が何回読み書きされたかという動作情報をカウントし，各構造体型ごとに合計する．以下の場合についてカウントした．

- 単独で使用できるオプションは，単独で使用する
- 他と組み合わせる必要があるオプションの場合は，できるだけ少ないオプションで済むように指定する
- カウントする構造体型変数は，グローバル変数，ローカル変数，関数の引数の 3 種類とする

gzip のオプションは 24 個存在する．そのうち対象としたのは 17 個のオプションであり，上記の条件から 17 通りを実行した．また，オプション指定なしでの実行を含め，18 通りを取得する動作情報の対象とした．

入力ファイルは，テキストファイル (Alice's Adventures in Wonderland: 163.6KB) と バイナリファイル (gzip-1.2.4: 57.1KB) を選択した．また，入力サイズを揃えるため，両方の入力ファイルを 57KB にした．

構造体型	読み取り回数		書き込み回数	
	テキスト	バイナリ	テキスト	バイナリ
tree_desc	296	555	24	45
ct_data	392823	619572	183570	242507
config	32	32	0	0
huft	434920	572345	6630	21760
stat	304	304	11	11
option	53	53	0	0

表 2: gzip の集計結果

表 1 と比較すると，圧縮・展開に関わる構造体のうち tree_desc, ct_data, huft は，入力の種類が変わるとアクセス回数が異なる．また，ct_data と huft は全体的にアクセス回数が多い．

圧縮・展開に関わらない構造体である stat, option は，入力の種類に関わらずアクセス回数が一定であり，アクセス回数は全体的に少ないことが分かる．

5 まとめ

C プログラムを対象として，構造体型の重要度を得るための指標について議論した．また，効率と正確さを考慮し，重要度を算出する中心的なパラメータとなる変数のアクセス回数をカウントするツールを実装した．

5.1 今後の課題

構造体のメンバ変数が構造体型である場合，現状ではメンバ変数のカウントをしていない．そのため，そのような構造体型の実際のアクセス回数を反映することができない．この点を改善する必要がある．また，取得したアクセス回数を利用して重要度というメトリクスを求めることを今後の課題とする．

参考文献

[1] 日高 隆博, 山本 晋一郎, 阿草 清滋, “ソフトウェアの動作状況のモデル化に関する研究”, 電子情報通信学会ソフトウェアサイエンス研究会, Vol.97, No.83, pp.49–56, 1998.

[2] Sapid, http://www.sapid.org/

[3] gcov, http://gcc.gnu.org/onlinedocs/gcc/Gcov.htm