ソフトウェア工学 140−11
（２００３．３．６）

ソフトウェアの大局的可視化のための組織化メトリクス

大橋 良徳† 山本 晋一郎†† 阿草 清滋†

本論文では，ソフトウェアの全体像を把握し，理解を支援するために，ソースコードを大局的に可視化する手法を提案する．既存のソースコードの可視化手法は主にノードと線分による複雑なグラフを作成する手法であり，明瞭な表現ではなかった．提案する手法では，単純な情報ではなく，組織化し，まとめ上げたソフトウェアメトリクスを用いて可視化を行う．組織化メトリクスを利用することで，既存の手法には見られない大局的ソフトウェアビジュアライゼーションを実現することが可能となる．大局的ソフトウェアビジュアライゼーションは，簡潔な表現で多くの情報を可視化することができ，ソフトウェア理解に役立つことを示す．

A perspective software visualization using the organization metrics

Yoshinori Ohashi,† Shinichiro Yamamoto†† and Kiyoshi Agusa†

In this paper, in order to grasp the whole software image, a technique of visualizing a source code globally is proposed. By a technique proposed, it visualizes not using simple information but the software metrics systematized and summarized. With systematization metrics, it becomes possible to realize perspective software visualization which is not seen to the existing technique. It is shown that perspective software visualization can visualize many information by brief expression, and is useful to software understanding.

1. はじめに

ソフトウェアは長期にわたって変更され続ける特徴を持ち，ソフトウェアの保守には多大な費用が費やされる．保守を行うプログラマには，対象についての十分な理解が必要不可欠であり，理解のためには長期にわたる学習が必要となる．現状として，新しい開発者が保守に参入することは非常に困難である．

要求された機能を新しい持つソフトウェアを作る場合，生産性の向上を期待し，既存のソフトウェアを再利用することが多い．この場合、プログラマは新たな機能拡張を考えたり，余分な機能を省くことで高速化を図るなど，既存のソフトウェアを改変することになる．ソフトウェアの改変を行う場合も，既存のソフトウェアに対する十分な理解が必要となる．

既存のソースコードを改変するためには，ソフトウェアの仕様書のみではなく，ソースコードを読んでの詳細な理解が必要となる．しかし，ソースコードを読んで理解することは非常に困難な作業である．ソースコードは実行順に記述されているわけではなく，多くの参照を用いており，逐次的に読み進めることはできない．読み進めるためには，同一ファイル内で頻繁に参照箇所を変更したり，他のファイルを読む必要があり，混乱しやすい．このように，ソースコードは複雑に入り組んでおり，加えて今日のソフトウェアプロジェクトは巨大化する一方である．大規模なソースコードとなれば，ソースコードを書いた本人でさえ，読むことが難しくなる．他人の書いたソースコードを読むとなれば，さらに多大な労力が必要とされる．そこで，ソースコードの理解を，人間の視覚能力に働きかけることで支援する，ソースコードの可視化技術が求められている．

本論文では，ソースコードの全貌が把握できる大局的な可視化手法を提案する．ソースコードの可視化をテーマとした研究は，これまでにも数多く行われているが，未だに一般的と言われるような手法は存在しない．その理由として，大規模かつ複雑なソースコードは，図形オブジェクトの重なりを生じない配置の実現が困難であることが挙げられる．これまでの可視化手法は主にノードと線分による複雑なグラフを生成する

† 名古屋大学大学院工学研究科情報工学専攻
Dept. of Information Engineering, School of Engineering, Nagoya University
†† 愛知県立大学情報科学部情報システム学科
Faculty of Information Science and Technology, Aichi Prefectual University

手法であり，明瞭な表現とは言えなかった．提案手法では，一つの手続きのみを解析して知ることができる局所的なメトリクスではなく，ソースコード全体を解析し，手続き同士をまとめ上げた組織化メトリクスを用いて，3 次元配置での可視化を行う．大局的情報である組織化メトリクスを利用することで，既存の手法にはない大局的可視化が実現できる．大局的可視化手法は，既存の手法以上の情報量が一つの図として可視化でき，グラフ表現よりも直感的な理解支援が行えることを示す．

2. 既存の可視化手法

ソフトウェアの機能，構造，実行手順などを 2 次元の図式として表現する方法に，プログラム図式がある．プログラム図式には，フローチャート，NS チャート，HIPO 図，PAD など，様々な図があり，古くから利用されてきた．しかし，プログラム図式は設計段階での利用を目的としており，理解のための記述方法とは言えない．ソフトウェアの規模が大きくなるにつれ，図も大規模なものになりがちで，一般のソースコードから実用的な図を自動生成することは非常に困難である．このような理由から，プログラム図式にはなかった新たなソフトウェアの図的表現方法が数多く提案されている．

一般に，可視化はデータの集合 D から図的データの集合 R へ変換するマッピング T_v と規定される．

$$T_v : D \mapsto R$$

ソースコードの可視化に限らず，これまでに開発されてきた数多くの情報可視化手法は，T_v の情報表現のレベルによって，基本的，局所的，大局的，の 3 種類のレベルにに大別できることが知られている[1]．

基本的　与えられたデータそのものを描く

局所的　データの関係を考慮に入れて描く

大局的　与えられたデータ全体を解析して初めて得られる事実を描く

例えば，平面上に配置されたデータの存在を，データ間の関係を意識せずそのまま 2 次元上に点や色で表現する場合，情報表現のレベルとしては基本的である．値の等しいデータを曲線で結び，等高線を描けば，情報表現のレベルとして局所的になる．データ全てを解析しなければ分からない，極大点を与える原点を明示するような可視化手法は，大局的な情報表現レベルに相当する．

ここでは，既存のソースコードの可視化手法を，情報表現の 3 種類のレベルに沿って説明していく

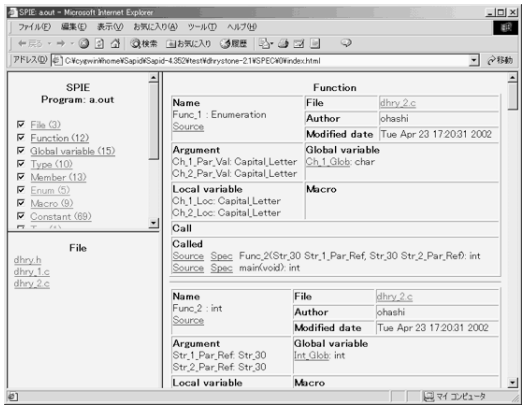

図 1　ソースコードブラウザ SPIE
Fig. 1　Source code browser SPIE

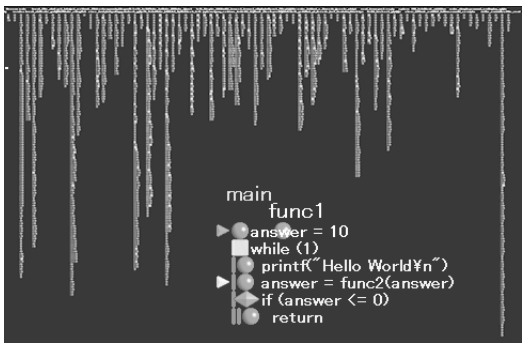

図 2　実行文のマッピング
Fig. 2　Mapping of a statement

2.1 基本的可視化

テキストエディタ上のソースコードは，“文字” という視覚的表現である．つまり，テキストエディタはソフトウェアを基本的レベルで可視化するものである．エディタは，可視化の最小単位を “文字” としているが，最小単位を “式”，“文”，“手続き”，“クラス” とすることにより，様々な粒度で基本的可視化を行うことができる。テキストベースのソフトウェア理解支援ツールとしては，GRASP[2]，SPIE[3] などが挙げられる．GRASP は繰り返しや分岐の制御構造を，矢印などの図形情報を用いてソースコード上に表示する．SPIE は HTML ブラウザを利用し，ソースコードに索引のタグ情報を付加し，リンクを張ることで理解支援を行う（**図 1**）．テキストベース以外の可視化手法としては，実行順に実行文を列挙し，平面上にマッピングする方法や，クラスを列挙する方法が挙げられる．これらは最小単位をそれぞれ”文”，”クラス”としており，任意の箇所について最小単位の粒度や参照箇所を変更できるような機能を付加することができる（**図 2**）．また，実世界のメタファーを用いて可視化を行う，Software

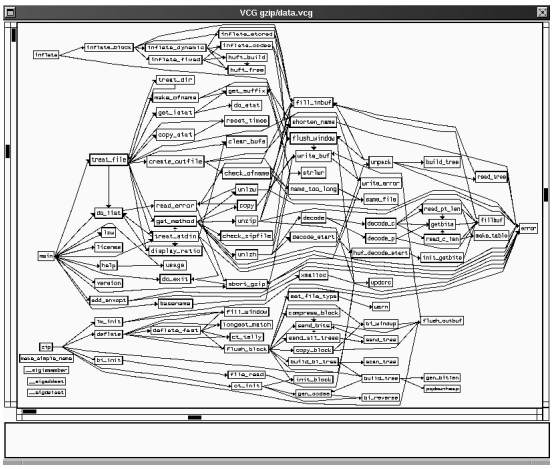

図 3　手続きの呼び出しグラフ
Fig. 3　procedure call graph

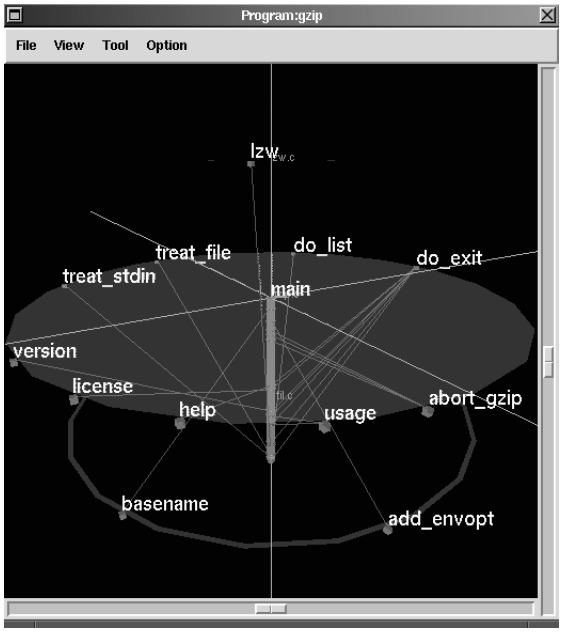

図 4　3 次元呼び出しグラフ VCRGL
Fig. 4　3-dimensional call graph VCRGL

World[4] が提案されている．Software World は，都市のメタファーを用いて，3 次元空間のバーチャルリアリティー環境で JAVA ソースコードの可視化を行う手法である．ソフトウェア全体を一つの都市と見立て，クラスを地区，メソッドをビルディングとして表現している．

これらの基本的可視化手法は，データを “順に並べた” だけで，各データ間の関係や大局的情報は視覚的に表されていない．基本的可視化は，可視化の最小単位の粒度によってソースコード全体を見渡すことはできても，具体的にどこを見ればよいのかは分からない．

2.2　局所的可視化

既存のソースコードの可視化手法の多くは，局所的可視化に分類できる．ソースコードの手続きなどの単位をノードとし，ノード間の様々な関係（呼び出し関係・継承関係・データ依存関係など）を線分で示した可視化手法は全て，局所的可視化に相当する．ノードと線分による局所的可視化は，線分を目で辿っていくことで，細かい関係まで知ることができるが，位置情報を利用して関係を表している手法は少なく，一目見て理解できる表現とは言えない．

図 3 のように，単純に手続き間の呼び出し関係を線分で結び，平面上に配置すると，非常に複雑なグラフになってしまう．そこで，2 次元での情報表現能力の限界を考え，3 次元空間内にグラフを配置した手法としては，VCRGL[5] が挙げられる．奥行きを利用した 3 次元配置によるグラフは，視点を変えることが可能であるため，2 次元配置よりも見やすく表示することができる（**図 4**）．3 次元コンピュータグラフィックスを用いることで，半透明などの視覚効果を与えることもでき，2 次元配置と比べてより多くの情報が表現できる．しかし，3 次元空間を自由に移動できるインタフェースが必要になることや，グラフィック処理速度の低下などの問題点もある．

呼び出し関係以外を局所的に可視化した手法としては，メッセージ通信の様子やクラスの継承関係などを可視化する GROOVE[6] が挙げられる．対象データ間の関係に注目した局所的可視化手法は，呼び出し関係，継承関係といった単一の関係のみを知りたい場合に有効である．一般に，ソースコードを局所的に可視化した場合，非常に複雑な図形になってしまい，別の関係を表すそれ以上の情報を付加することができない．それは，規模が大きく複雑なソースコードは，可視化の際に図形オブジェクトを配置することさえ困難であるからである．そこで，可視化手法ではなく，オブジェクトの配置方法に注目した研究も行われている[7][8]．しかし，それらの研究もやはり，データを単に並べた基本的可視化か，グラフを描く局所的可視化のための配置方法である．局所的可視化はノードと線分によるグラフであり，全体像を捉える目的には向いていない．

2.3　大局的可視化

既存のソフトウェアの可視化手法に，大局的可視化に分類できる手法は存在しない．大局的可視化を行うためには，対象データ全体をスキャンして初めて得ら

れる情報を明示するように可視化しなければならない．本研究では，既存の手法にはない，ソースコードの大局的可視化の実現について考えていく．

3. 可視化手法

3.1 大局的メトリクス

ソースコードの手続きを可視化する際，手続きの性質を表すためにソフトウェアメトリクスが利用できる．ソフトウェアメトリクスとは，複雑度，信頼性，効率などソフトウェアプロダクトの様々な特性を判別する客観的な数学的尺度である．メトリクスを用いて開発作業の生産性やプロダクトの状態を評価することで，問題のある作業に対する改善を行うことができたり，開発プロジェクト全体を円滑に進めることができる．ソフトウェアメトリクスには主に，モジュール結合度やモジュール凝集度など，ソフトウェアの複雑度を表す複雑度メトリクスと，行数や式の数など，ソフトウェアの規模を表す規模メトリクスがある．複雑度メトリクスを可視化したツールは，デバッグやリファクタリングの候補箇所を示す目的で利用されているが，ソフトウェアの理解を目的としたメトリクスではない．そこで，可視化には規模メトリクスを利用する．これまで可視化に利用されてきた規模メトリクスは，手続きに対する行数や変数の数など，単純な規模メトリクスであり，一つの手続きのみを見て得られる局所的な情報であった．しかし，本研究で目的とする大局的可視化はその定義上，対象の全体を解析してはじめて得られる大局的情報を可視化しなければならない．そこで，局所的なメトリクスではなく，大局的なメトリクスを用いて可視化を行うことにする．

3.2 組織化規模メトリクス

ソフトウェアの理解過程をモデル化する場合に，認知心理学における組織化の概念を用いることができる．組織化とは，人間が受け取った幾つかの情報を，それらの意味によって一つにまとめ上げることで，より高水準な情報を構成するることである．これまでにも，認知心理学における組織化の概念を利用し，ソフトウェアの理解コストを計量する研究が行われている[9)10)]．ソフトウェアを理解する際は，仕様やコメントなどを手がかりにソースコードを読み進め，多くの手続きを機能や呼び出し関係において高水準の概念にまとめ上げていく．この手続きをまとめ上げる行為は，組織化の概念に非常に近い．そこで，ソフトウェアの理解を，組織化を繰り返し行うことによって，ソフトウェア全体に対応する単一の高水準情報を構成することと定義する．組織化の対象が手続きなら，呼び出す側の手続きが組織化においてより上位，呼び出される側の手続きが下位となる（5）．言い換えれば，全ての手続きをそれらの関係を基にまとめ上げることが，ソフトウェアの理解の定義である．

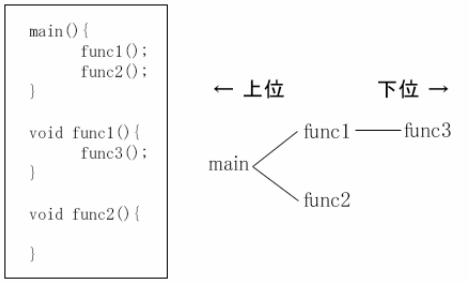

図 5 手続きの組織化
Fig. 5 Organization of procedure

この理解の定義に基づき，手続き f に対して局所的な規模メトリクスを呼び出し関係において組織化した，組織化規模メトリクス $S(f)$ を次のように定義する．

$$S(f) = n + \sum_k t_k * S(f_k)$$

n は f の局所的規模メトリクス
f が手続き f_k を t_k 回参照しているとする

今回，可視化するソースコードは C 言語を対象としたため，n は f のトップレベル式数とした．トップレベル式とは，セミコロンまでの式の集合である．分岐や繰り返しの条件判定式もトップレベル式とする．局所的規模メトリクスとしてトップレベル式数を利用した理由は，プログラマが部分式の集まりであるトップレベル式を，一つの処理のまとまりと理解していると考えたためである．$S(f)$ は，大局的な手続きの規模を表しているのと同時に，手続きを呼び出し関係において組織化した際に，どの手続きがどの手続きの上位・下位に位置するのかを表している．

3.3 提案手法

先に述べた従来のソースコードの可視化手法の多くは局所的可視化手法であり，手続きなどのオブジェクトを表すノードとその関係を表す線分とで作られているグラフが大半を占めた．グラフの全体を理解するには，関係を目で何度もたどっていく必要があるため，複雑なグラフは全体像を捉える目的には向いていない．ソースコードの全体像を瞬時に捉え，直感的な理解を支援するには，オブジェクト間の関係を表すのに線分を用いるより，位置・形・色・向きなどの類同性を利用したゲシュタルト心理学における群化の要因を用いた方が適している．位置における群化の要因を利用するためには，オブジェクト間の距離に意味を持たせた配

表 1 可視化情報
Table 1 Information of visualization

縦軸:	組織化規模メトリクス
縦以外の軸:	呼び出し階層
軸の周回:	制御
色:	ファイルの種類
マーク:	その他の情報

置を行わなければならない．既存の可視化手法には，距離に意味を持たせた配置を行うため，図に複雑な制約を持たせた手法があるが，ユーザの心的なモデルとかけ離れた可視化表現は理解の妨げとなる．本研究では，ユーザが直感的に理解できる表現を用いてソースコードを可視化することを目標とした．これまでのように，手続きを表すオブジェクトの長さをステートメントや行数で表現した場合，ソースコードの複雑さ故に，距離に意味を持たせた配置を行うことは困難であった．しかし，大局的情報である組織化規模メトリクスを用いてオブジェクトを表現することにより，呼び出し関係における距離の意味を考えた配置を実現することができる．位置情報がオブジェクト間の呼び出し関係を示しており，線分を用いない簡潔な表現でソフトウェアを可視化することが可能となる．

手続きの配置は 3 次元で行い，組織化規模メトリクスが最大となる手続きを 3 次元座標の中心に配置する．手続きの縦方向の長さは組織化規模メトリクスを表す．組織化規模メトリクスの最小単位はトップレベル式であり，トップレベル式のソースコード上の出現順に縦軸が上から下へ対応している．手続き呼び出しがあった場合，呼び出し側の手続きの外側に呼び出される手続きを配置する．縦軸に注目することで，手続きの規模や，呼び出し順序を知ることができる．ある手続きが中心からどれくらい離れているのかに注目すれば，呼び出し階層の深さを知ることができる．中心の手続きに限り，制御情報も表している．分岐によって 2 つのうちどちらかしか呼び出されることのない手続きは，縦軸の位置を揃え，中心の手続きを取り巻くように配置する．ただし，本手法のソースコード解析は静的である．分岐の行く先を実際の動作に合わせて判断しているのではなく，ソースコード上から得られる情報のみを用いて判断している．つまり可視化された制御情報は，実際の振る舞いとは必ずしも合致していない．しかし，プログラマが常に動的な振る舞いを考えてソースコードを理解しているとは言えず，特に構造化プログラミングでは，分岐や繰り返しなどは一連のブロックとして理解している場合が多い．ソースコードを読んでその内容を理解する場合も同様で，ソフトウェアの動作の振る舞いを表示したアニメーションを見るより，実際にコードを読んだ方が理解できるという場合も少なくない．本手法は，繰り返しや分岐が動的な振る舞いとは異なっているが，ソースコードの構造に合致した表現である．本手法は，ソフトウェアの大局的情報を明示し，直感的に全体像を捉える目的で可視化を行う．グラフとは違い，呼び出し関係をわざわざ目で辿る必要がなく，一目で主要な手続き同士をまとめ上げて理解することができる．

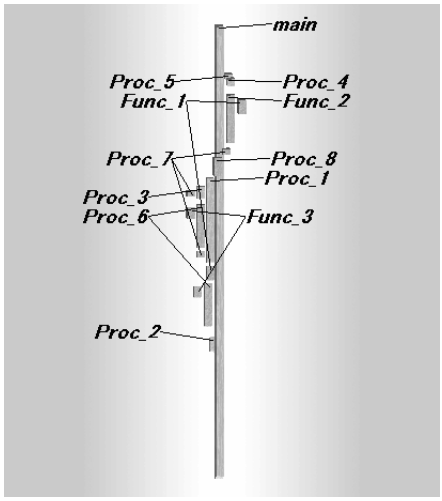

図 6 Dhrystone の大局的可視化
Fig. 6 Perspective vizualization of Dhrystone

図 6 は，ベンチマークテスト用に用意された Dhrystone を提案手法で可視化した結果である．処理の中心である main 関数は図の中心に位置しており，上から下へ処理の実行が表され，手続きを呼び出した時点で呼び出された手続きが配置されている．手続き間の境が分かりやすいように，手続きの配置は中心に対して螺旋状に行った．main 関数を上から下へ見て行くと，最初に Proc_5, 次に Proc_4 を呼び出していることが分かる．Proc_5,Proc_4 は，縦の長さが短いため，実行命令数の少ない小規模な手続きであると判断できる．その次に呼び出されている Func_2 は，縦の長さも比較的長く，規模の大きい手続きである．Func_2 は途中で Func_1 を呼び出しており，Func_1 は全体で 2 回呼び出されている．以上のような情報から，手続き同士のまとまり，どの手続きがどこで何回呼ばれているのか，呼び出し関係で末端の手続きはどれか，といった事柄を瞬時に知ることができる．

3.4 ソフトウェアナビゲーション

プログラマが，大きなソフトウェアの保守に新しく着手する場合，そのプログラマは新しい町に引っ越し

てきたばかりの新参者と同じである．膨大な規模のソースコードの中で，右も左も分からず，始めはソース内に残されたコメントや，部分的にでも残されているドキュメントを手がかりに，手探りで理解を進めていくしかない．この時，プログラマの進むべき道が示されていれば，大きな理解の手助けとなる．ソフトウェアナビゲーションとは，ソースコードの参照すべき箇所を明示し，ソフトウェア内を案内してくれる機能のことを指す．

ソフトウェアナビゲーションには様々な手段が考えられる．手続きに対し，呼び出し関係やデータ依存関係をたどる機能や，規模やバグの潜在可能性など手続きの性質を表示する機能がそれに当たる．ナビゲーション機能を持つ既存の開発支援ツールには，SPIE[3]や，商用の開発環境で利用されている Tree View がある．これらのツールはテキストブラウザやツリー上でナビゲーションを行っているが，どちらも一長一短と言える．

ブラウザ　細かい情報までリンクをたどってジャンプしていけるため，参照の自由度が高い．しかし，ジャンプの履歴が分かりにくいため，ソフトウェア全体における現在位置の把握ができない．数回のジャンプですぐに現在の参照地点に至った経路が分からなくなってしまう．

ツリー　ソフトウェアの階層的構造に合致しているため理解しやすいが，ナビゲーションの自由度はツリーの項目の最小単位に制限される．あまりにツリー項目を小さくしすぎると，展開した際にテキストを見る場合と何ら変わりがなくなってしまうため，大局的なナビゲーションには向いていない．

両者に共通の問題点は，どちらもソフトウェアの実行経路が表現できない点である．数ある目的候補地の中から，どのような実行経路を通って目的地に到着するのか，目的地までの距離がどのくらいなのか，目的地に到達できるのかという点は，目的地の決定に重要な役割を果たしている．ツリーの場合，関係のあるツリーを全て展開すれば，経路を表現することも可能だが，その場合，非常に複雑なツリーになってしまうことが予想される．そこで，経路を考慮に入れた自由度の高いナビゲーションツールが必要とされる．

実行経路を理解するには，対象の全体像を表示する必要がある．ここでの全体像とは，閉じたツリーのように，小さくひとまとまりにされているのではなく，常に全てのツリーが展開された状態で，その縮図が表示されていることを指す．つまり，必要なソースコードのどの一点も，必ず可視化されたソフトウェアの図の一点に対応していなければならない．提案した可視化手法は，ソフトウェアの縮図の条件が満たされている．縦軸のみが実行経路を示しており，経路が理解しやすい表現である．提案した可視化手法を利用すれば，ブラウザやツリーよりも自由度の高いナビゲーションを行うことができる．

図 7　zip 関数の可視化
Fig. 7　Visualization of zip function

4. ツールの利用例

本研究で実装したツールの利用例を，gzip を解析し理解していく例を挙げつつ説明していく．gzip は，Lempel-Ziv アルゴリズムを利用して，ファイルの圧縮と伸長を行うプログラムであり，ファイル数 14，関数の数 92 の比較的大きなプログラムである．

4.1 Step1: 呼び出し関係において，閉じた手続きの集合を作成

始めに，対象プログラムの全手続きに対し，呼び出し関係において閉じた集合を作成する．各集合内で，組織化規模メトリクスが最大の手続き名を，ラベル情報として表示する．表示された手続きは，ソースコード全体で他から一度も呼び出されない手続きである．ここで表示された手続きは，zip，make_simple_name，inflate，main の 4 種類であった．

4.2 Step2: 注目したい手続きを中心とした可視化

表示された手続きのラベルをクリックすることにより，その集合の手続きが呼び出し関係によってまとまった図として可視化される．**図 7** では，zip をマウスでクリックしたことで，zip 関数を中心とした図が描かれている．手続きのオブジェクトは 3D で配置されており，視点を自分の好きな角度，距離に変更することができる．図を見ることで，どの手続きがどの手続きの上位にあるのか，どの手続きか呼び出し元であるの

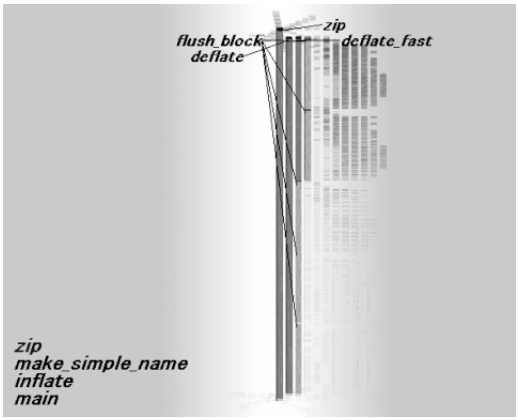

図 8 zip 関数の可視化（手続きラベル数 4）
Fig. 8 Visualization of zip function (label 4)

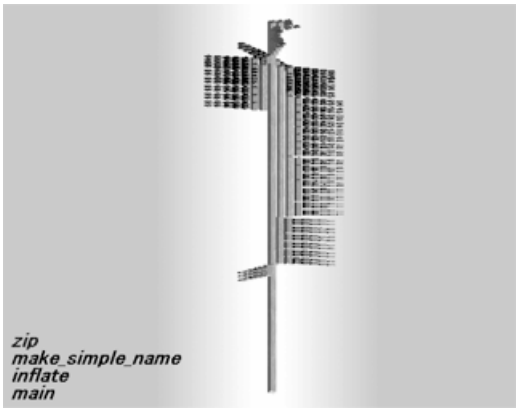

図 9 flush_block 関数の可視化
Fig. 9 Visualization of flush_block function

かを知ることができる．また，中心の手続きに限り，分岐がある場合は同じ y 軸上に表示されるため，大まかな処理の流れを理解することができる．

4.3 Step3: 手続きラベルの表示

関数名をラベル情報として表示することができる．しかし，ラベルを表示する手続き数が多いと複雑な図になってしまうため，組織化規模メトリクスの大きい手続きを選び，ラベル情報を付加する手続きを 4 に限定した結果が **図 8** である．

中心の zip 関数は，最初に小さな手続きを幾つか呼び出しているが，その手続きの組織化規模メトリクスは小さく，ラベルが非表示となっている．図 8 では半透明で表示されている．これらの手続きは前処理の手続きであり，zip 関数を理解する上では無視してもそれほど支障のない手続きである．処理の中心的役割を担っている手続きは，重要度メトリクスが zip 関数に次いで大きい deflate 関数 である．実際 deflate 関数は多くの処理を任されている手続きであり，zip 関数を理解する上で重要な手続きである．deflate 関数は，deflate_fast 関数を 1 度，flush_block 関数を 3 度呼び出している．deflate_fast 関数は，flush_block 関数を 2 度呼び出しており，全体として flush_block は 5 回呼び出されている．手続きのオブジェクトの縦の長さは，組織化規模メトリクスであり，これは実行命令数に比例する．つまり，zip 関数の処理のほとんどは，flush_block 関数である．以上のことは，図 8 を一目見ることで直感的に理解可能である．

4.4 Step4: 注目手続きの変更

任意の手続きを中心にして，図を変更することができる．先の説明で，zip 関数の処理の中心的な役割を果たしているのが flush_block 関数であることが分かった．そこで，今度は flush_block 関数を中心として表示する（**図 9**）．flush_block 関数は分岐処理を行っているため，手続きの配置が途中で 2 分されていることが分かる．以降，zip 関数を見たときと同じように調べていくことで理解を進めることができる．このようにして，全く分からないソフトウェアに対しても，大局的な情報を基に理解することができる．

4.5 ソースコードの表示

可視化した図は，対応するソースコードの箇所を参照するためのインタフェースとなる．図をマウスでクリックすることにより，対応する部分のソースコードを見ることができる．ソースコードの任意の場所にジャンプすることができ，図とソースコードの相互参照が可能になると共に，情報空間を自由に移動することができる．

5. 可視化手法の評価

可視化手法は，どのくらい多くの情報を，どのくらい分かりやすく可視化できているかを比較することで評価できる．

5.1 情報量に関する評価

どれだけ多くの情報量が一つの図として表現できているかを既存の可視化手法と比較する．比較対象として，現在多くの開発環境で実用化されている Tree View，3 次元配置のグラフで手続きの呼び出し関係を表示した VCRGL を選択した．表 2 から分かるように，本手法では，既存の手法以上の情報量が可視化できている．

5.2 表示オブジェクト数に関する評価

どれだけ図が簡潔に表現できるかを定量的に評価するため，既存の可視化手法の評価として，表示オブジェ

表 2 情報量の比較
Table 2 Comparison of the amount of information

	ファイルの種類	呼び出し関係	規模メトリクス	制御情報
Tree View	○	×	×	×
VCRGL	○	○	○	×
本手法	○	○	○	○

クト数の個数が数えられている[5]．しかし，本手法では組織化規模メトリクスの大きさを利用し，目的に応じて表示オブジェクトの数を自由に変更することができるため，表示オブジェクト数は全てのソフトウェアに対して任意である．しかも，表示オブジェクト数の数をどんなに少なくしても，非表示のオブジェクトは邪魔にならないように半透明で表示されており (図 8)，呼び出し関係や大局的な情報が失われることはない．表示オブジェクト数に関する評価としては，既存のどの手法よりも優れている．

6. まとめと今後の課題

本論文では，ソフトウェアの保守や再利用の効率を高めるため，ソフトウェアの理解を支援するソースコードの新たな可視化手法を提案した．既存の可視化手法は，大局的可視化に重点を置いた手法ではなく，基本的，または局所的な可視化手法であった．本研究では，組織化した規模メトリクスを用いることで，これまでにない大局的なソースコードの可視化を実現することができた．組織化規模メトリクスは，表示オブジェクトの選択にも役立ち，多くの情報を分かりやすく可視化することができる．可視化の自由度，情報量，利用例における理解の例から，本手法が優れた可視化手法であることを示すことができた．今後の課題としては，本可視化手法にさらなる情報を付加していくことにより，手続き型の言語のみでなく，オブジェクト指向言語に対応した可視化を行うことが考えられる．また，本可視化手法がソフトウェアの理解に対してどのくらい優れているのか，その有効性を定量的に示すことが必要である．

謝辞 本研究を進めるにあたり，多くの助言をいただいた阿草研究室の皆様に感謝致します．

参 考 文 献

1) 岸野文郎 大野義夫 藤代一成 北村喜文, “岩波講座 マルチメディア情報学 情報の可視化” 岩波書店 (2001)
2) James H. Cross II, Kai H. Chang, and T. Dean Hendrix, “Visualization with Control Structure Diagrams” GRASP Ada CrossTalk Articles (1996)
3) 大橋洋貴, “ハイパーテキストに基づいたソースプログラム・レビュー支援ツール”, 電子情報通信学会, ソフトウェアサイエンス研究会, 電子情報通信学会技術研究報告 SS98-28,pp15-22 (1998)
4) Claire Knight and Malcolm Munro, “Virtual but Visible Software” Published in the International Conference on Information Visualisation 2000 (2000)
5) 安原継二 山本 晋一郎 阿草 清滋, “オブジェクト属性を利用したソフトウェアの視覚化”, 日本ソフトウェア科学会 FOSE2000 pp.189–196 (2000)
6) Dean and Stasko, John, “The Information Mural: A Technique for Displaying and Navigating Large Information Spaces”, IEEE Transactions on Visualization and Computer Graphics, Vol.4, No.3 pp257–271 (1998)
7) 酒井恵光 山口和紀 川合慧, “図形オブジェクトの遠隔度に基づく階層集合の可視化モデル” 情報処理学会論文誌 Vol.40 No.9 pp.3455-3470 (1999)
8) 尾下真樹 牧之内顕文, “オブジェクト指向データベースの半自動可視化環境” 情報処理学会論文誌 Vol.42 No.SIG 1(TOD 8) pp.56-69 (2001)
9) 掛下哲朗 山崎直子, “静的なオブジェクト指向プログラムに対する理解コスト計量法” オブジェクト指向 2000 シンポジウム pp101–108, 2000
10) 山崎直子 松原義継 掛下哲朗, “認知心理学的アプローチに基づくソフトウェア理解度計量法” コンピュータソフトウェア Vol.16 No.6 pp55–67, 1999