特別研究報告

題目

新規開発者のためのプログラム依存グラフ可視化によるソフトウェア理解支援手法の提案

指導教員
楠本 真二 教授

報告者
藤川　達也

平成 22 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 21 年度 特別研究報告

新規開発者のためのプログラム依存グラフ可視化によるソフトウェア理解支援手法の提案

藤川　達也

内容梗概

近年，ソフトウェア開発，保守運用形態が多様化しており，ソフトウェアの保守担当者と，そのソフトウェアの開発者が異なる場合が増えている．保守担当者と開発者が異なる場合，保守担当者はバグの修正や機能追加をする上で，他人が実装したソースコードを理解しなければならない．ソースコードの理解は，一般的に, 関数の呼び出し関係を辿りながらその処理内容を読み進めていく．関数の呼び出し関係に注意を払いつつ，現在着目している関数内の処理を理解しなければならないため，ソースコードの理解自体が非常に困難な作業である．他人が実装したソースコードとなると, ソフトウェアに対する知識が不足しているため, より困難である．ソースコード理解支援に関する研究は，コールグラフなどを用いた関数間の呼び出し関係の理解支援と，プログラムスライスなどを用いた関数内の処理内容の理解支援に大別され, 両者ともツールを用いて自動的に関係を解析し, その結果を可視化する場合が多い．しかし，対象ソフトウェアの規模が大きく, 呼び出し関係が複雑な場合や，1 つの関数内の処理が多く複雑な場合は，可視化されたグラフも複雑になってしまい，十分に理解支援できているとはいえない．

そこで本研究では，他の開発者が実装したソースコードについて, 理解支援の第一歩として, 関数内で行われる処理の理解支援のための, プログラム依存グラフの可視化の手法を提案をした．さらに，提案手法を Eclipse のプラグインとして実装し適用を行った．その結果，従来に比べ理解しやすい可視化手法であることが確認できた．

主な用語

ソースコード理解支援
プログラム依存グラフ
Eclipse
プラグイン

目 次

1　まえがき

近年，ソフトウェア開発，保守運用形態の多様化により，ソフトウェアの管理や保守を行う保守担当者と，そのソフトウェアを開発した開発者が異なる場合が増えている．保守担当者と開発者が異なる場合，保守担当者は，他人が実装したソースコードを理解しなければならない．しかし，限られた時間の中で，大規模化，複雑化しているソフトウェアを十分に理解し，適切に管理・保守を行うことは難しい．例えば，この問題は下記のような場合で起こりうる．

(1) ソフトウェア開発を受注した企業は，その開発の一部もしくは大部分を，中国やインドなどのソフトウェアベンダに委託（オフショア開発）や，国内の下請けに発注を行うことが多い [1]．そして委託先から納入されたソフトウェアの品質を確かめるために，レビューやテストなどを行う．

(2) 長期間にわたって運用されているレガシーソフトウェアでは，当初の開発に携わった開発者が定年や異動などにより運用チームに残っていない場合がある．その場合，新規にそのプロジェクトに配置された保守担当者が，フィールドバグの修正や法改正による機能修正などを行う．

(3) 近年企業の統廃合が盛んに行われており，元来別システムであったソフトウェアを統合し，1 つのシステムとして運用する場合がある．そのため保守担当者は元来別システムであった部分に関しても保守・管理を行う必要性がある．

(4) 社内における既存のソフトウェア資産やオープンソースソフトウェアを利用して，新規開発に要するコスト削減を減らす動きが盛んである．既存の資産を再利用する場合，そのまま利用できる場合は少なく，新規開発で用いる場面に合わせて軽微な修正が行われる．社内のソフトウェア資産を利用する場合は，開発当初のドキュメントは付随しているが，設計，コーディング，テストの際に改訂された仕様変更がドキュメントに反映されていない場合が多く，再利用の際にドキュメントが役に立たない場合が多い．オープンソースソフトウェアの場合は，そもそもドキュメントが少ない，あるいは全くない場合が多く，ソースコード自体を良く理解したうえで再利用を行わなければらない．

このように，さまざまな場面において，他人の実装したソースコードを理解する必要がある．ソースコードの理解は，一般的に関数の呼び出し関係を辿りながらその処理内容を読み進めていく．呼び出し元の関数の機能を把握するためには，呼び出し先の関数で実現されている機能も把握しなければならない．関数の呼び出し関係に注意を払いつつ，現在着目して

いる関数内の処理を理解しなければならないため，ソースコードの理解は，非常に困難な作業である．他人の実装したソースコードの理解はより難しいといえる [2].

ソースコード理解支援に関する研究は，コールグラフなどを用いた関数間の呼び出し関係の理解支援と，プログラムスライスなどを用いた関数内の処理内容の理解支援に大別される．両者ともツールを用いて自動的に関係を解析しその結果を可視化する場合が多い．しかし，対象ソフトウェアの規模が大きく呼び出し関係が複雑な場合や，1 つの関数内の処理が多く複雑な場合は，可視化されたグラフも複雑になり，十分に理解支援できているとはいえない．

そこで，本研究では，他の開発者が実装したソースコードの理解支援手法を提案する．本研究は，ソースコード理解支援の第一歩として，関数内の処理の理解支援を行う．本研究は既存の研究と同じく，関数内の処理内容をプログラム依存グラフを用いて可視化するが，より容易に処理内容を理解できるよう，下記の 2 点について，工夫をしている．

制御依存辺の起点となる頂点の折りたたみ機能

関数内の処理内容が多く複雑な場合，一度に関数全体のプログラム依存グラフを表示したのでは，グラフ自体が大きくなりすぎてしまい，ユーザはどの部分に着目していいのか判断が難しい．そこで，制御依存辺の起点となる頂点に折りたたみ機能を持たせ，その頂点に依存している頂点を必要に応じて非表示にできる．複雑な処理内容を持つ関数の場合，関数内には複数の選択処理や繰り返し処理が存在する．それらの一部もしくは大部分を非表示することにより，描画される頂点数と有向辺数が減少し，理解しやすい規模のプログラム依存グラフとなる．

ソースコード上での位置を考慮した頂点の配置

可視化したプログラム依存グラフの頂点の配置がソースコード上での要素の位置と関係なくランダムに配置される場合，ソースコード上での要素と，プログラム依存グラフ上での頂点の対応を取ることが難しい．この問題を解決するため，本研究では，プログラム依存グラフの頂点は，ソースコード上での位置とできるだけ類似した形で配置する．これにより，ソースコード上での逐次処理の関係や if 文や while 文の条件文と，その内部に存在しているプログラム要素の関係をより簡単に把握することができる．

さらに，提案手法を Eclipse のプラグインとして実装し，実際のソフトウェア対して適用を行った．その結果，従来に比べ理解しやすい可視化手法であることが確認できた．

以降，2 章ではプログラム理解に関する関連研究を紹介し，3 章では提案手法を説明する．4 章では，提案手法を実装したツールの評価について触れ，5 章で考察を行う．最後に 6 章で本論文をまとめる．

2　関連研究

スライスとはプログラムの命令と変数の部分集合からなるスライシング基準をもとに導き出されたプログラム依存グラフの部分グラフのことである．スライシング基準の全ての変数に対し, スライシング基準の命令の直前における変数の値が, スライスとスライス元のプログラム依存グラフの実行において等しくなる必要がある．Krinke は, プログラム依存グラフの可視化手法と, プログラム依存グラフから求めたスライスをソースコード上で表示する手法を提案した [3]．プログラム依存グラフの頂点を, 垂直方向にはスパニングツリーグラフに基づいた配置を行い, 水平方向には辺の交差が少なくなるように配置することにより, プログラム依存グラフを可視化している．スライスの可視化はスライス基準の命令に対応した頂点から何回依存辺を辿れば到達できるかによって, スライスに含まれる頂点に対応したソースコードの部分の色を調節することで行っている．また, 関数の規模が大きい場合には, プログラム依存グラフを可視化しても複雑になるため, スライスの可視化においては, グラフィカルな可視化よりもテキスト上での可視化が優れていることについて述べている．

Wettel らは，ソフトウェアを都市に見立て，三次元コンピュータグラフィックスによる視覚化を行った [4] ．オブジェクト指向プログラムのクラスを 1 つの建物とし，メソッド数を高さ，フィールド数を底面積とした四角柱で表現する．この建物をパッケージごとに区分けされた地区に配置することにより，ソフトウェア全体の可視化を行っている．また，それを応用したソフトウェアの開発履歴の可視化手法を複数提案している．coarse-grained age map は，クラスやパッケージを作成時期によって色分けする手法であり，ソフトウェア全体の概要の理解に適している．coarse-grained time travel は，開発履歴をある期間ごとに区切って各段階での状態を可視化したものを順に並べる手法であり，クラスの生成や消滅の履歴を確認する事ができる．他にも 2 種類の手法を提案しており，それぞれの手法により得られる有益な情報を提示している．

Balmas は，プログラム依存グラフの理解を容易にするため，階層構造に基づいてノードをグループ化する手法を提案した [5] ．プログラムによる自動グループ化とユーザによる手動グループ化の双方が必要である事を述べた上で，自動グループ化として各関数のグループ化，各ループのグループ化，およびこれらを同時に用いる手法を提案した．また，これらの手法を実装したツールを作成し，実験によって各手法の長所と短所を分析し，最終的に関数とループのグループ化を用いる手法が最も有効であることを述べた．

リファクタリングとは、外部の振る舞いを変えずにソースコードの内部構造を整理することである．視覚的にリファクタリングを支援するツールが多く提案されている． Murphy-Hill らは，メソッド抽出リファクタリングを視覚的に補助するツールを提案した [6] ．提案したツールは，Selection Assist，Box View，Refactoring Annotations の 3 つである．Selection

Assist は，エディタで文を選択する際，完全な文の範囲をハイライトで表示する．複数行にまたがる文や if 文の範囲を容易に確認することが可能である．Box View は，コードのネスト構造をボックスで表示する．エディタで文を選択すると対応するボックスの色が変化し，ボックスを選択すると対応する文が選択状態となる．コードのネスト構造の範囲を理解しやすいが，ネストが深い場合，全てを表示するために広い表示領域が必要となる．Box View は完全にフォーマットを抽象化するため，馴染みのないコードフォーマットを理解する助けとなる．Refactoring Annotations は，ソースコード上にコントロールとデータのフローを矢印で表示する．実験により，視覚的に補助ツールを使用した方が，より早く正確にリファクタリングが行え，補助となるツールは特定の内容に対して最適化されている方がよいことがわかった．

Mens らは，リファクタリング間の衝突を自動で分析する手法を提案した [7] ．オブジェクト指向のメタモデルを定義し，それに基づき対象ソースコードのタイプモデルを作成する．リファクタリングをグラフの変形ルールとして定義し，複数のリファクタリング間の衝突可能性を自動的に分析する．分析の結果を表に示すことで，リファクタリング間の衝突関係を視覚的に示した．また，リファクタリング適用前後のタイプグラフを表示することで，効果を視覚的に確認することも可能である．

3　提案手法

この章では，本研究で提案するプログラム依存グラフの可視化手法および，提案手法を実装したツールについて述べる．

3.1　準備

本節では, 提案手法を読み進めるにあたり，必要な知識を記述する．

3.1.1　プログラム依存グラフ

プログラム依存グラフは, プログラム内の命令を頂点として, 頂点間における依存関係を有向辺で表す有向グラフである．頂点間の依存関係には, 次の 2 つがある [8].

1. データ依存関係 DD(s,t) (Data Dependence)
   命令 s から命令 t に対してデータ依存関係 DD(s,t) があるとは, ある変数 w が存在して, 命令 s における変数 w の定義が, 変数 w を使用している命令 t に到達する場合をいう．データ依存関係 DD(s,t) は, 命令 s で設定する変数 w の値が命令 t で参照される可能性があることを示している．

2. 制御依存関係 CD(s,t) (Control Dependence)
   命令 s から命令 t に対して制御依存関係 CD(s,t) があるとは, 命令 s が分岐命令であり, 命令 t その分岐文内に直接含まれている場合, あるいは, 命令 s がループ命令であり，　命令 t がそのループ文内に直接含まれている場合をいう．命令 t が分岐文 S 内に直接含まれているというのは, 分岐文 S 内の他の分岐文 S 内のほかの分岐文やループ文に命令 t が含まれていないことを意味する．制御依存関係 CD(s,t) があると, 命令 t が実行されるかどうかは, 命令 s の実行結果 (命令 s における制御移行) に依存している．

図 1 はプログラム依存グラフの例である．ソースコード一行目“ a = 1 ”の定義を三行目の制御述語“ `a > 0` ” で使用しているため,“ a = 1 ”の頂点から“ `a > 0` ”の頂点にデータ依存辺が引かれている．また, 三行目の制御述語“ `a > 0` ”の条件判定の結果は四行目の“ b = a + b ”の実行の有無に関わるため,“ `a > 0` ”の頂点から“ b = a + b ”の頂点に制御依存辺が引かれている．同様に, 他の依存辺も引かれている．

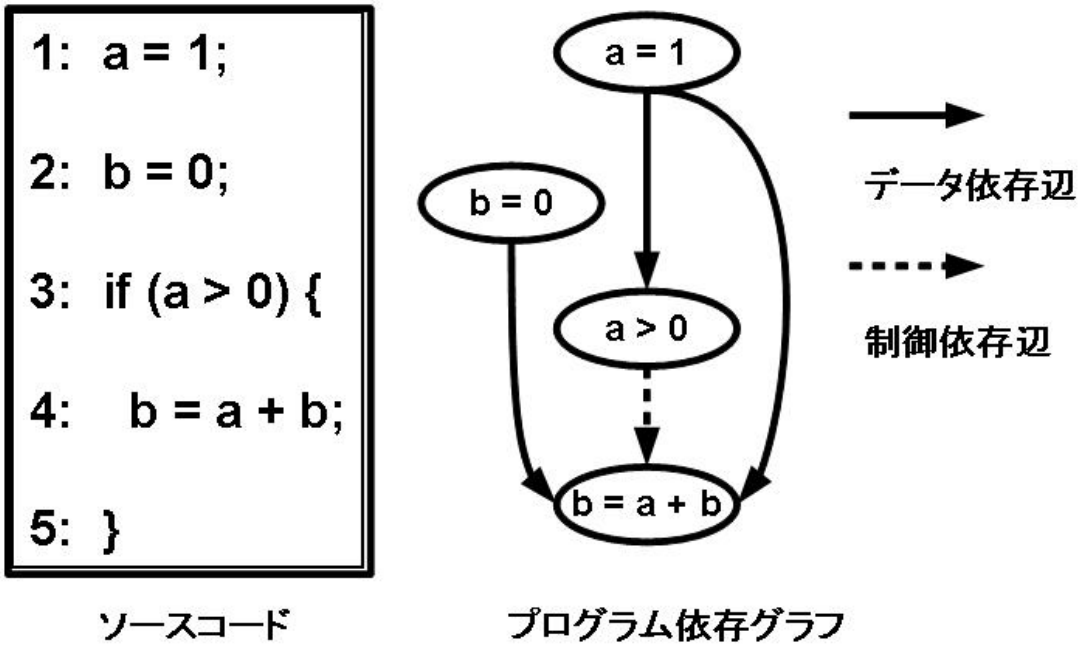


図 1: プログラム依存グラフ

3.1.2 SWT

SWT(Standard Widget Toolkit) は Java プラットフォーム用ウィジェットツールキットの一種で, Eclipse プラットフォームのために開発された GUI(グラフィカル・ユーザー・インタフェース) を作成するためのツールキットである [9][10]. SWT では Canvas というクラスを利用して描画を行うが, Canvas には基本的な API しか用意されていない. そこで, Canvas の機能を拡張したライブラリの Draw2D を用いる.

3.1.3 Draw2D

Draw2D は SWT Canvas の機能を拡張して, レイアウトマネージャーやフィギュアを追加したライブラリーである [9][11]. フィギュアには色の情報や形状の情報を持たせることができ, フィギュア同士が親子関係を持つことが可能で, フィギュアはそれぞれレイアウトマネージャーを設定することができる. Draw2D は Eclipse ワークベンチの提供するビューやエディタをといったリソースを使用しないので, GEF によって Draw2D と Eclipse ワークベンチとの統合を行う.

3.1.4 GEF

GEF はモデルをグラフィカルに編集するために使用するフレームワークであり, ユーザーとのインタラクションや開発効率の改善を行う [9]. GEF では MVC(Model-View-Controller) アーキテクチャを基にしてグラフィクスを描画する [12]. MVC アーキテクチャでは, 対話的なアプリケーションを実現するため, アプリケーションの構成要素を, データの保持や処理を受け持つ部分の Model, Model の状態を表示をする部分の View, マウスやキーボードからの入力を処理する Controller の 3 つに分割し, それぞれの関係は図 2 のようになる. 以下に Model, View, Controller の概要を説明する [13].

Model

アプリケーションで必要となる実際のデータを保持しており, 目的の業務に合わせた処理を実行する. GEF において, Model は自身がどのように描画されるかを知らず, 実際に描画するのはコントローラの役目をもつエディットパートが作成するフィギュアである.

View

モデルのデータを取り出して, ユーザーが見るのに適した形で表示する. GEF においてはフィギュアがその役目をもち, ここでいうフィギュアは Draw2D におけるフィギュアと同等である.

Controller

マウスやキーボードのクリックといった入力に関するイベントの中から, 自分に関係あるものを選択し, 適切な形に変換してモデルに伝達する. GEF においてはエディットパートがその役目をもつ.

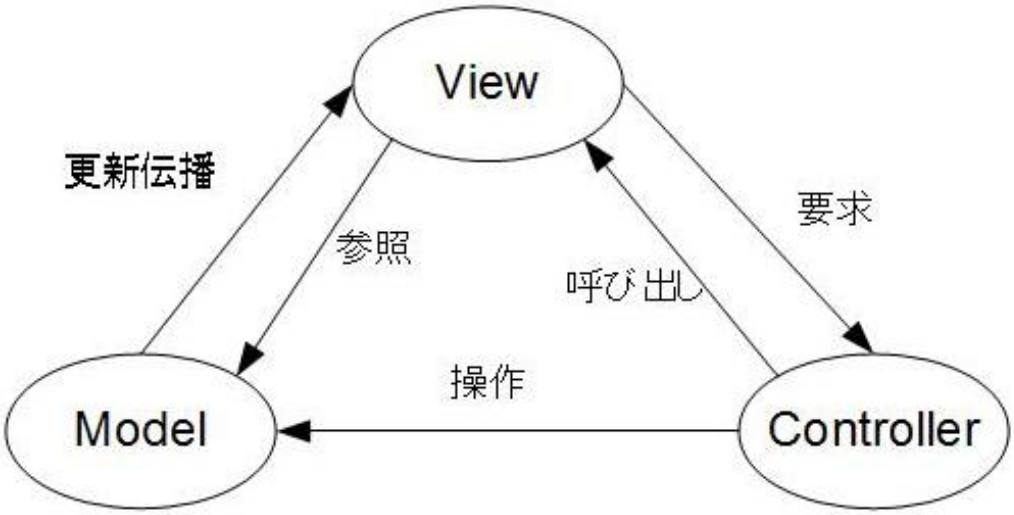


図 2: MVC

3.2 Loop Positioning Algorithm を用いた頂点の配置

PDG の頂点の配置には, Loop Positioning Algorithm を用いた [14]. このアルゴリズムは次のように頂点を配置する.

- 頂点は縦一列に配置する.
- 制御依存先の頂点は制御依存元の頂点に対して右下に配置し, エディタ上で if 文や while 文などのブロックがインデントされているのと対応させる.

さらに, ソースコードとともに表示することを考え, 頂点の並びを行の並びに準じさせるようにした.

図 3 は Loop Positioning Algorithm を用いたプログラム依存グラフ (辺は制御依存辺のみとした) の頂点の配置の様子を表している. ソースコード上において青色のブロックに直接含まれている文と, 緑色のブロックに含まれている文に, それぞれ対応した頂点は, ソースコード上でインデントされている文の位置に対応した配置となっている.

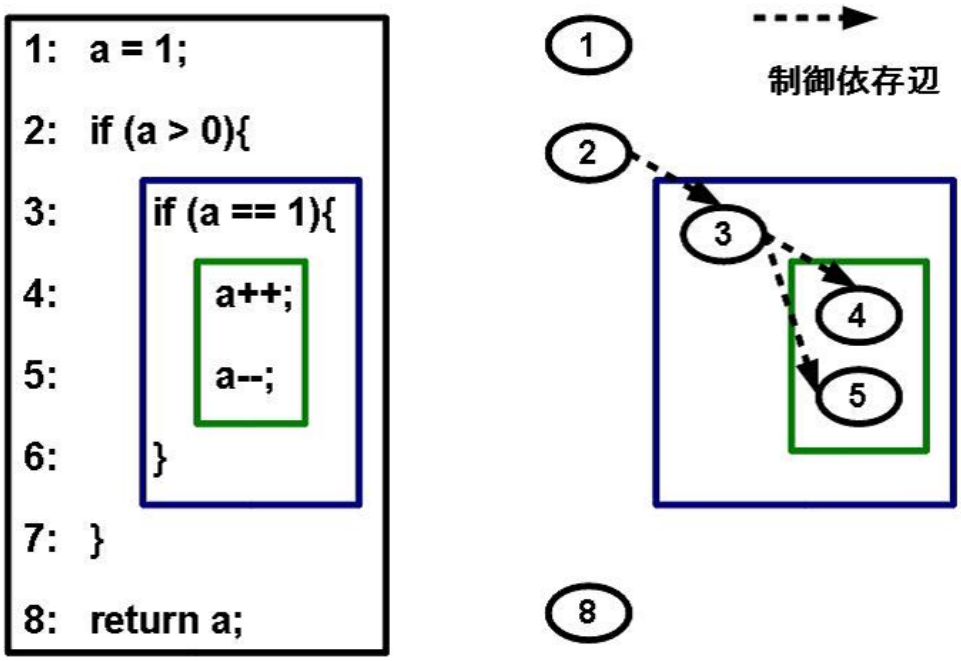


図 3: ソースコードのインデントに対応した頂点の配置

3.3 頂点の折りたたみによる視認性の改善

同時に表示する頂点の数を減らすことで, より視認性を向上させることができる. 本手法では if 文や while 文や for 文の条件式に対応する頂点とその頂点から制御依存辺が出ている頂点に対して折りたたみを行えるようにすることで, 表示する頂点数を減らすことができる.

図 4 は折りたたみの様子を表している. 例えば, バグの修正を行っていて, 図 4 の青いブロック内には何も問題がないことが分かっている場合を考える. プログラム依存グラフを用いてバグの箇所を見つける際には, そのブロック内についての情報は必要ないため, 青いブロックの折りたたみにより, 不必要な部分の頂点を非表示にすることで, 視認性を向上させている.

3.4 Eclipse プラグインとして実装

Eclipse は統合開発環境の 1 つで, コアとなるランタイム以外は全てプラグインとして追加されており, プラグイン開発環境自体もプラグインとなっている [9][15]. 本研究では, Java ソースコードの編集に適した Java パースペクティブ (図 5) において, Java エディタの右隣にプログラム依存グラフを出力するビューを追加するプラグインを実装した.

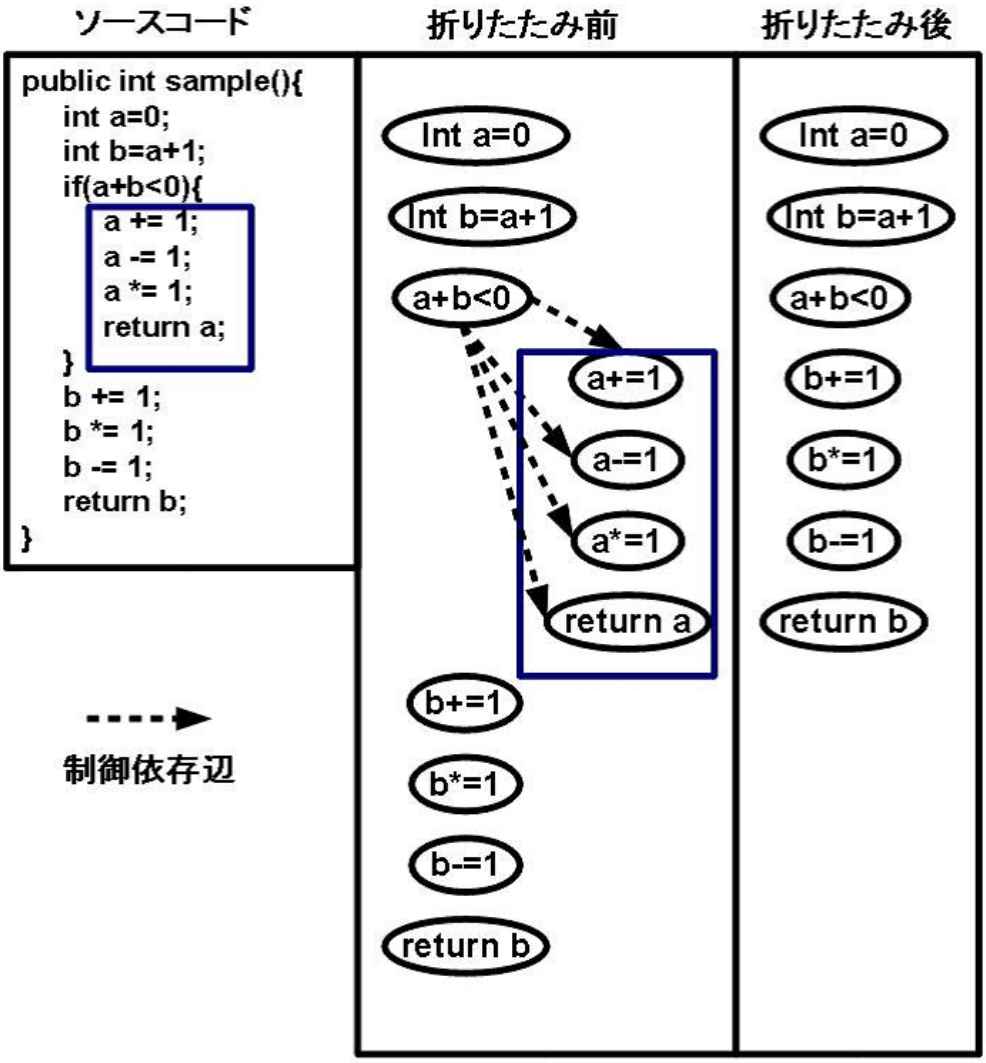


図 4: 折りたたみ機能の使用例

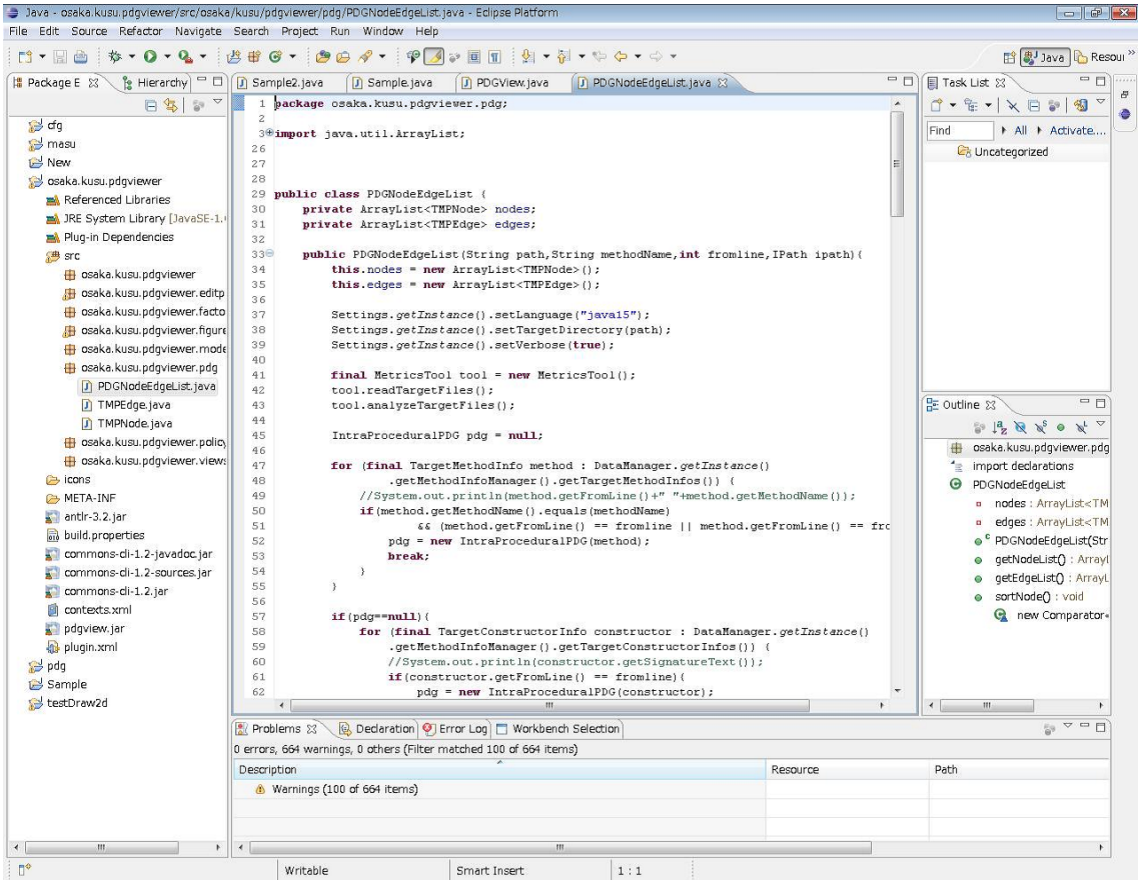


図 5: Java パースペクティブ

4　適用と評価

この章では, 提案手法をもとに実装したプログラム依存グラフ可視化を行うビュー PDG View の動作と, PDG View において提案手法がどのように動作しているかを説明する.

4.1　画面構成

図 6 は PDG View を表示した画面である. PDG View とそれに関連する箇所について説明する.

Package Explorer

Java パースペクティブでデフォルトで開かれているビューの 1 つで, ワークベンチ内の Java プロジェクトの階層構造を表示する. 本研究で作成したプラグインでは, Package Explorer からメソッドを選択し, PDG View にプログラム依存グラフを表示することができる.

Editor

Java ファイルのソースコードなどの編集を行うことができるテキストエディタである. 本研究で作成したプラグインでは, PDG View からクリックされた頂点の情報を受け取り, 対応した行にキャレットを移動させる.

PDG View

本研究で作成したプラグインにおいて新たに追加したビューで, Package Explorer もしくは Outline で選択されたメソッドに対するプログラム依存グラフを作成し, このビューに表示する.

Outline

現在 Editor で開かれているファイルの概略を表示すビューである. Editor が Java ソースファイルを編集している際は, クラスやフィールドやメソッドといった要素の構造を示す. 本研究のプラグインにおいては Outline からもメソッドを選択し, PDG View にプログラム依存グラフを表示することができる.

4.2　プログラム依存グラフを表示するメソッドの選択

図 7 はプロジェクト構造をツリー表示する Package Explorer ビューである. メソッドを表す (図 7 では緑の丸もしくは黄色いひし形) アイコンのついた項目に対してプログラム依

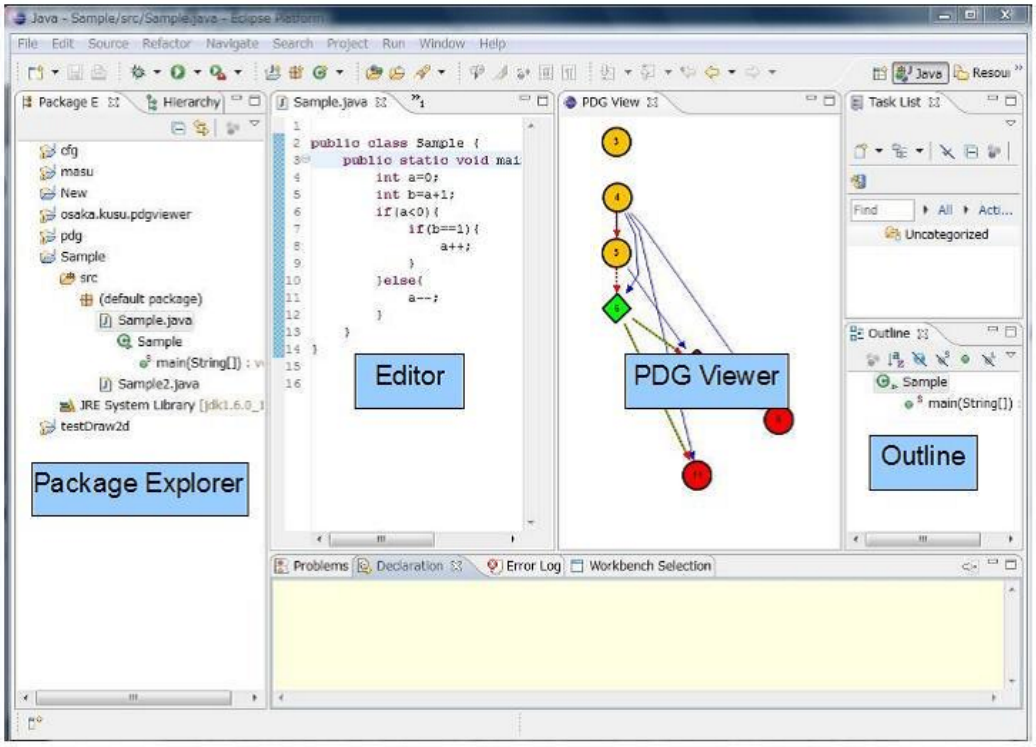


図 6: 画面構成

存グラフを表示させることができる．Outline ビューにおいても同様にメソッドを表すアイコンのついた項目についてプログラム依存グラフを表示させることができる．

4.3 プログラム依存グラフの表示方法

図 8 は Package Explorer ビューにおいてプログラム依存グラフを表示するメソッドを選択し, 右クリックした際の画面である．”Create PDG” というコマンドを右クリックにより表示するポップアップメニューの一番下の項目に追加しており, そのコマンドをクリックすると選択したメソッドに対するプログラム依存グラフを表示することができる．

4.4 頂点と有向辺の説明

図 9 は ”Create PDG” コマンドによって PDG View がプログラム依存グラフを表示している画面である．頂点の中心の数字は, 頂点がソースコードの何行目に対応しているかを表している．頂点と有向辺の色は頂点の種類を表している．頂点の赤色は, その頂点に対応するステートメントの実行でメソッドの処理が終了する可能性があることを示している．緑色の頂点は, その頂点が分岐頂点であり, 制御依存辺がその頂点から出ていることを示している．制御依存頂点は, 他の頂点が円形なのに対してひし形となっている．頂点の黄色は, それ以外の頂点であることを示している．有向辺の色は, 青色がデータ依存関係, 緑色が制御依存関係, 赤色が実行依存関係をそれぞれ表している．頂点の色に関しては優先順位があり, 赤色, 緑色, 黄色, の順で優先される．ソースコードの N 行目に対応する頂点を頂点 N と呼ぶこととすると, 図 9 では頂点 6 と頂点 7, 頂点 7 と頂点 8, 頂点 6 と頂点 11 の間は緑色の有向辺と赤色の有向辺が重なっている．

4.5 頂点の配置

図 10 はプログラム依存グラフの頂点の配置とソースコードのインデントの対応を表している．ソースコードにおいては, 6 行目の実行における条件判定「a<0」の結果次第で, 次に実行する行が 7 行目と 11 行目とに分岐しており, 7〜9 行目と 11 行目がインデントされている．PDG View では頂点 6 からの制御依存辺を持つ頂点 7 と頂点 11 が頂点 6 に対して右下に配置されており, ソースコードにてインデントが行われているのに対応している．ソースコードの 8 行目は, 6 行目の条件判定「a<0」が真であり, かつ 7 行目の条件判定「b==1」が真である場合に実行されるため, 2 段階インデントされている．PDG View では頂点 6 からの制御依存辺を持つ頂点 7 が頂点 6 の右下に配置されており, 頂点 7 からの制御依存辺を持

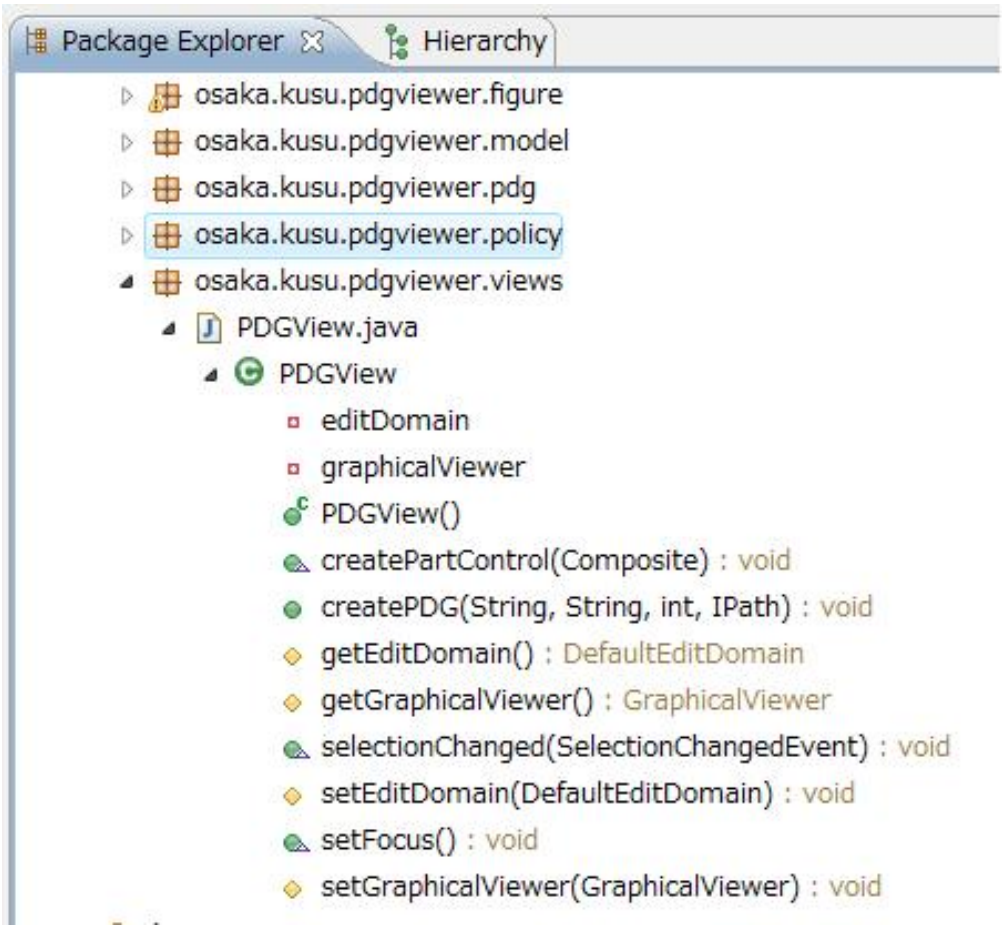


図 7: Package Explorer

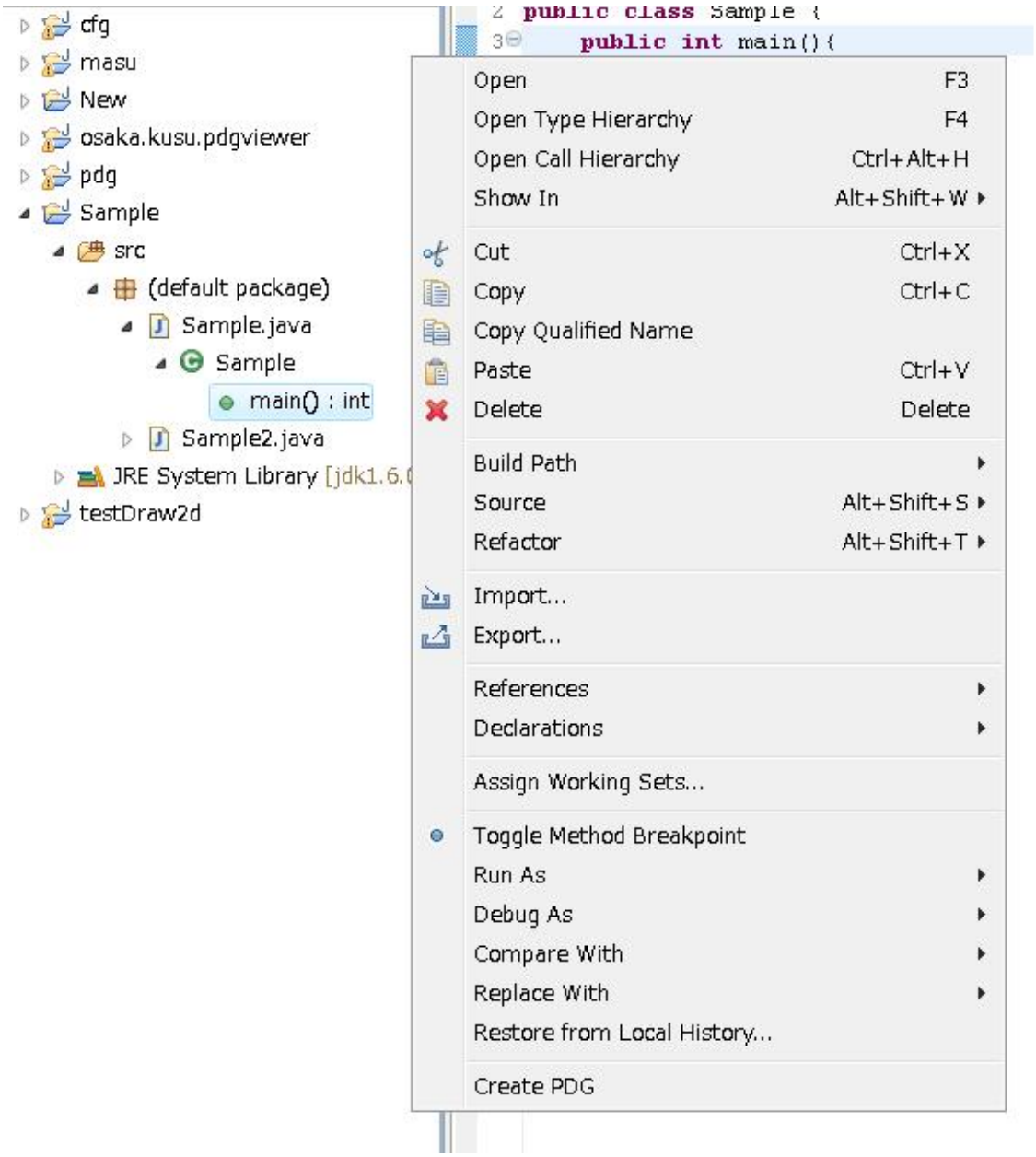


図 8: ポップアップメニュー

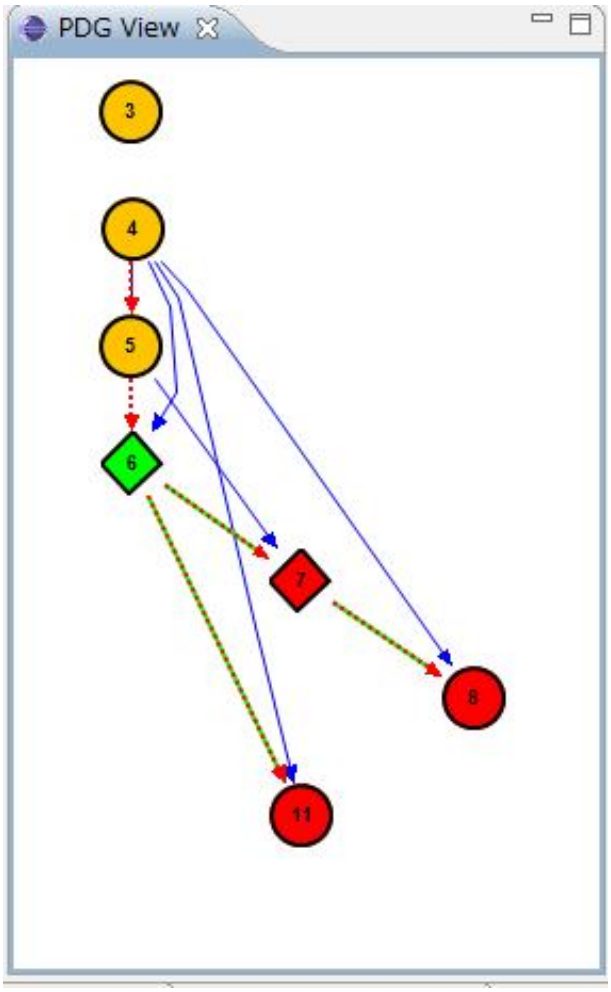


図 9: 頂点・有向辺の説明

つ頂点 8 が頂点 7 の右下に配置されており, 2 重以上の条件文のインデントにも頂点の配置は対応している.

4.6 ツールチップ

図 11 は頂点に対するツールチップを表示している画面であり, 頂点 5 に対するツールチップを表示している. 頂点にカーソルを合わせるとそれに対応したツールチップが表示される.

4.7 頂点の折りたたみ

図 12 は頂点の折りたたみの様子を表している. 図 12(a) の頂点 7 をダブルクリックすることで, 図 12(b) のように, 頂点 7 からの制御依存辺をもつ頂点 8 を折りたたむことができる. 図 12(a) の頂点 4 から頂点 8 に出ている制御依存辺が図 12(b) では頂点 7 へ向かっているが, これは, 折りたたみをされた頂点への有向辺を折りたたみを行った頂点に向けたものにまとめているためである.

4.8 キャレットの移動

図 13 は頂点のクリックによるキャレット (文字の入力位置を示す記号) の移動を行う様子を表している. 図 13(a) の状態ではエディタのキャレットは 4 行目にあるが, 頂点 5 をクリックすると, 図 13(b) のようにキャレットはソースコードの 5 行目に移動する. PDG View で表示しているプログラム依存グラフに対応したソースコードのファイルはエディタで開かれている必要はなく, 開かれていない場合は, プログラム依存グラフに対応したソースコードのファイルをエディタ上に開き, 頂点に対応した行にキャレットを移動させる.

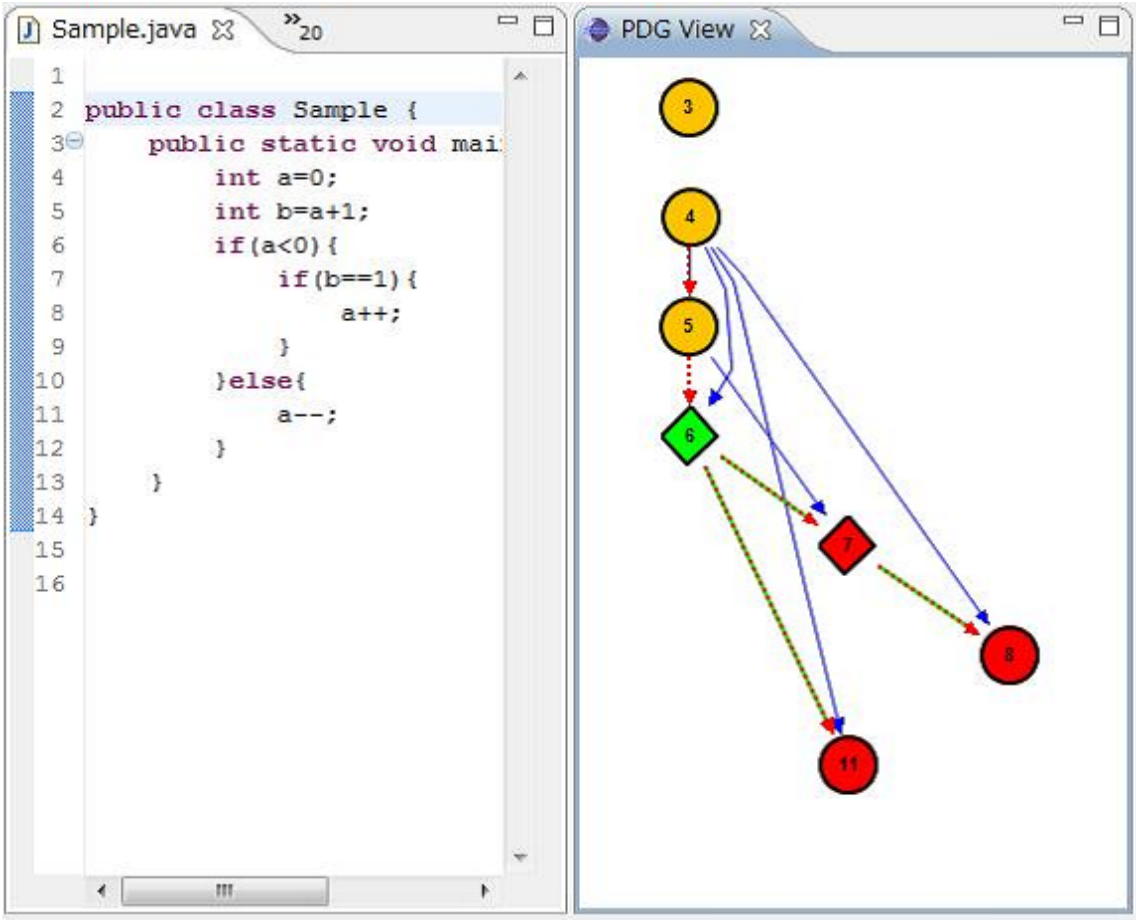


図 10: ソースコードと PDG View の対応

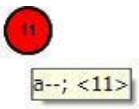
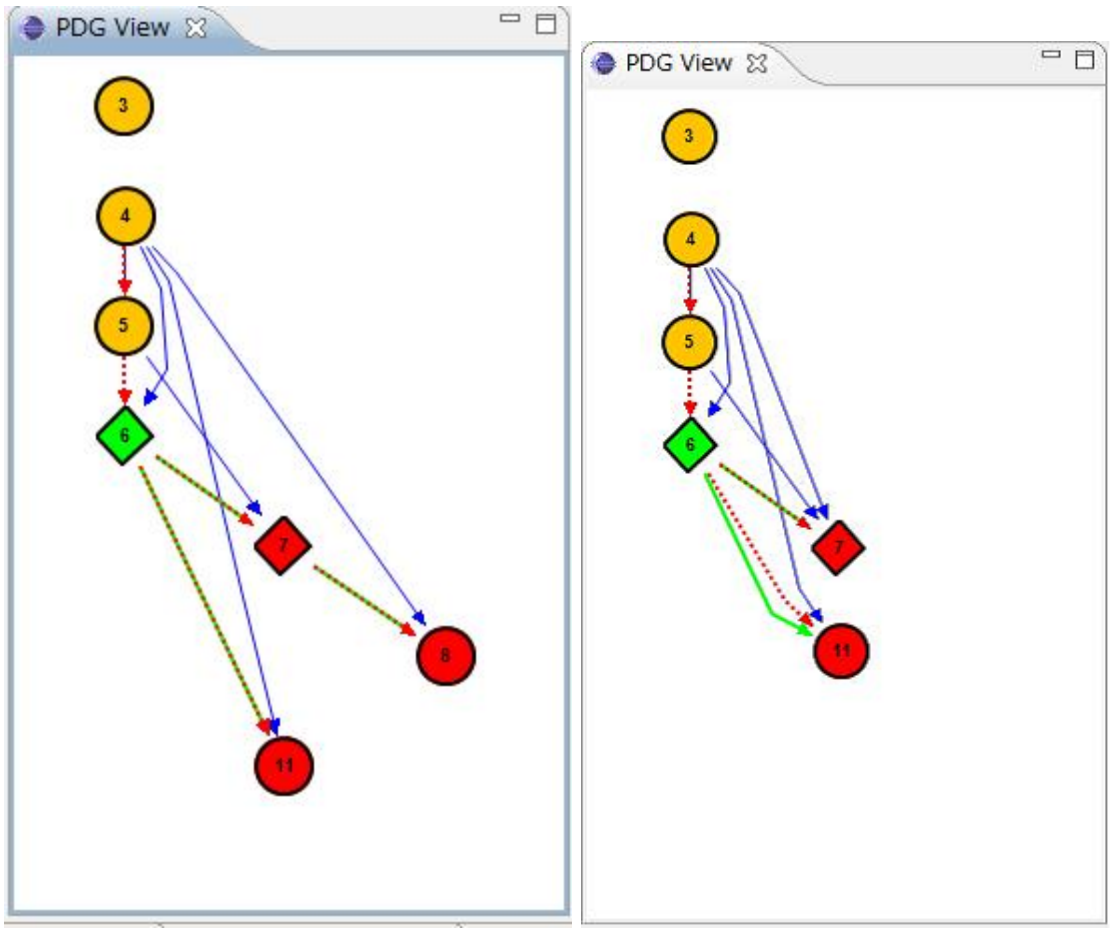
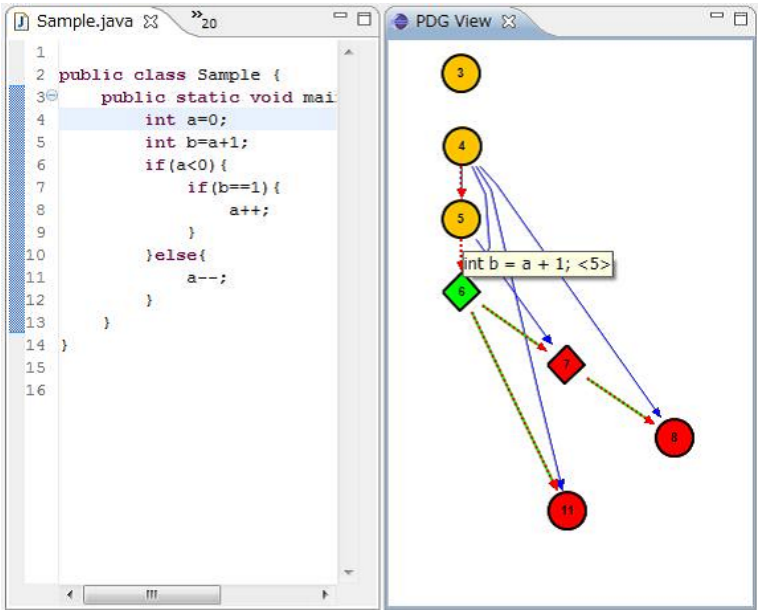


図 11: ツールチップ

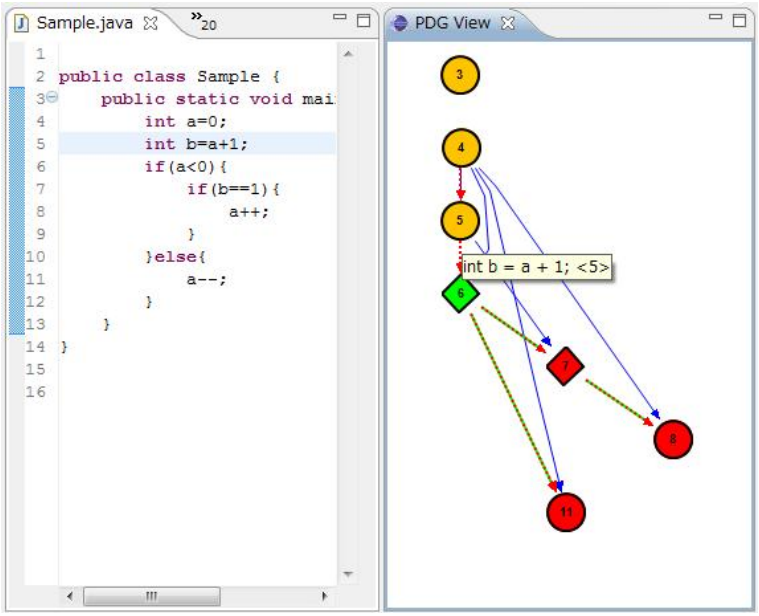


(a) 折りたたみ前　　(b) 折りたたみ後

図 12: 頂点の折りたたみ



(a) キャレット移動前



(b) キャレット移動後

図 13: キャレットの移動

5　考察

頂点配置アルゴリズムに Loop Positioning Algorithm を用いることで, 条件分岐, ループ文を分離させ, ソースコード上での位置関係に倣った頂点配置でプログラム依存グラフを表示することができている. さらに, ソースコードの並びに準じて頂点を配置しているため, ソースコードとプログラム依存グラフの対応が分かりやすくなっている. そして, 頂点の折りたたみを可能にすることで, 複雑なメソッドに対するプログラム依存グラフにおいて, 現在注目する必要のない箇所を折りたたむことができる. このことによって, PDG View における頂点数を減らし, 視認性を向上させている.

しかし, 本研究の提案手法におけるプログラム依存グラフの頂点配置アルゴリズムでは, 頂点が横に並ぶことがないために, プログラム依存グラフの表示領域が, 他の配置アルゴリズムに比べて縦に長くなり, 頂点数が多いプログラム依存グラフを表示する場合に, ユーザが頂点を探すのにスクロールバーを移動させる頻度が増える. これは, 本研究において作成した Eclipse プラグインにおいて PDG View の表示領域に使用できる領域の横幅が狭いことに起因している. しかし, この問題を回避するために, 頂点を横に並べることで縦幅を短くするアルゴリズムを用いた場合には, 大きなメソッドに対するプログラム依存グラフを表示する際に, 縦幅だけでなく, 横幅でも, PDG View の表示領域を超過するために, スクロールバーを二方向に操作しなければならない.　そのため, 移動幅の増大を考慮しても, スクロールバーの移動方向を一方向のみに限定できる可能性が高い提案手法のアルゴリズムの方が, Eclipse プラグインとしての実装においては優れていると考える.

他の問題として, 頂点の折りたたみを行った際に, 1 つの頂点に有向辺が集中し, 視認性が下がる恐れがあるということが挙げられる. この問題は, 折りたたみを行う頂点数が増えた場合に, 折りたたまれる頂点から出ている, もしくは折りたたまれる頂点に向かっている有向辺を制御依存のある頂点から出ている辺, もしくは, その頂点に向かっている辺として, 集められることで発生する. 頂点の数を減らそうとして折りたたみをすれば, 有向辺が集中し, 折りたたみを行わなければ, 一度に表示する頂点の数が増えるために, スクロールバーの移動が増える, といったようにトレードオフの関係にあると考えられる.

6　あとがき

本研究では, 新規開発者のソースコード理解支援を目的とした, プログラム依存グラフの可視化手法を提案した. そして, 提案手法を用いたツールを統合開発環境の 1 つである Eclipse のプラグインとして作成した. その結果, Eclipse において, エディタの隣にプログラム依存グラフを表示させることが可能になった. 提案手法では, 頂点の配置は Loop Positioning Algorithm を用いており, エディタ上で if 文や while 文, for 文のブロックがインデントされているのと対応させている. さらに, if 文や while 文, for 文のブロックに含まれる頂点に対して折りたたみを行うことを可能にすることで, 一度に表示する頂点の数を減らし, 視認性を向上させている.

プログラム依存グラフ可視化ツールについては視認性の向上の面で改善の余地がある. 頂点を選択し, その頂点に対して依存辺をたどって到達できる頂点や有向辺のみを表示することで, より必要な情報のみをユーザに与えられるようになると考えられる. 他にも, プログラム依存グラフをビューの大きさに合わせて表示できるようにすることで, 全体像を理解しやすくできると考えられる.

謝辞

本研究の全過程を通して, 常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 楠本真二　教授に心から深く感謝いたします.

本研究を通して, 御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 岡野浩三　准教授に深く感謝いたします.

本研究を通して, 多大なる御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後芳樹　助教に深く感謝いたします.

本研究を通して, 御指導頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 柿本健　特任助教に深く感謝いたします.

その他楠本研究室の皆様のご協力に, 心より感謝いたします.

参考文献

[1] 白井晴男. ベトナムにおけるオフショア開発と人材育成. 上武大学経営情報学部紀要, Vol. 33, pp. 63–80, 2009.

[2] 檜皮祐希. 同時更新情報と呼出情報を用いて機能実装クラス群を抽出する手法の提案. Master's thesis, 大阪大学, 2007.

[3] J. Krinke. Visualization of program dependence and slices. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, pp. 168–177, 2004.

[4] R. Wettel and M. Lanza. Visual exploration of large-scale system evolution. In *Proceedings of the 15th IEEE International Working Conference on Reverse Engineering*, pp. 219–228, 2008.

[5] F. Balmas. Displaying dependence graphs: a hierarchical approach. *Journal of Software Maintenance and Evolution*, Vol. 16, No. 3, pp. 151–186, 2004.

[6] E. Murphy-Hill and A.P. Black. Breaking the barriers to successful refactoring: Observations and tools for extract method. In *Proceedings of the 30th international conference on Software engineering*, pp. 421–430. ACM New York, NY, USA, 2008.

[7] T. Mens, G. Taentzer, and O. Runge. Analysing refactoring dependencies using graph transformation. *Software and Systems Modeling*, Vol. 6, No. 3, pp. 269–285, 2007.

[8] 下村隆夫. プログラムスライシング技術と応用. 共立出版, 1995.

[9] 竹添直樹, 志田隆弘, 奥畑裕樹, 里見知宏, 野沢智也. Eclipse 3.4 プラグイン開発 徹底攻略. 毎日コミュニケーションズ, 2009.

[10] Swt tips and samples - java swt でスタンドアローンアプリケーション開発. http://cjasmin.fc2web.com/.

[11] Draw2d 入門. http://www13.plala.or.jp/observe/draw2d/draw2d_overview.html.

[12] Gef 入門. http://www13.plala.or.jp/observe/GEF/GEF_Overview.html.

[13] J2ee tips:mvc アーキテクチャ. http://www.stackasterisk.jp/tech/java/mvc01_01.jsp.

[14] T. Würthinger. Visualization of java control flow graphs. Master's thesis, Johannes Kepler University Linz, 2006.

[15] Eclipse.org home. http://www.eclipse.org/.