ソフトウェア工学 102-6
（1995. 1.26）

大規模複雑プログラム理解のためのプログラム分析と可視化技術[1]

秋山義博
金沢工業大学情報工学科

長期に及ぶソフトウエア保守に於いて，かなり多くのソースステートメントを含む大規模複雑プログラムは，設計文書が無い，複雑なので理解するのが困難である等の問題を顕著に示している。現存プログラムの実体と特徴に立脚して，この論文では，プログラム理解支援のアプローチとツール設計，開発したプロトタイプツール：ソフトウエアマイクロスコープとそれを用いた実験について述べる。ソフトウエア保守目的のプログラム理解支援のアプローチ，ＣＡＳＥツール設計は，特定のプログラム設計方法論，プログラミング言語に独立であり，問題プログラムの基本設計や基本構造をプログラマーの助け無しに自動的に浮き彫りにするようなプログラム分析，表現，可視化技術等が必要になる。又，大規模複雑プログラムの理解の効率が低下しないように，隠れた機能や分散した機能の把握を助ける段階的抽象化支援とユーザオペレーションの量をエディターを用いてプログラム理解を得る場合に比べて１０^{-3}以下に減らすユーザインターフェスを実現することが必要である。ソフトウエアマイクロスコープは，これらの課題を解決した。

CASE Technology for Analyzing and Visualizing Large and Complex Programs to Help Program Understanding

Yoshihiro Akiyama

Information and Computer Engineering
Kanazawa Institute of Technology
7-1 Ohgiga-oka Nonoichi-machi Ishikawa 921 Japan
E-mail: akiyama@infor.kanazawa-it.ac.jp

Long term software maintenance is deceased by large and complex programs leading difficulties in understanding, modifying, documenting, and testing software including them. In this paper, A new approach for program understanding and CASE tool design are proposed and the prototype called Software Microscope is described with the experimental evaluation. It is noted that the approach and CASE tool design of Software Microscope do not depend on particular methodology or programming language, but is based on a general one to those used during the maintenance. It is important to note that the tool automatically reveal and visualize the fundamental program structure and design without having user's help. In addition, the new tool user interface is simple and powerful enough to help Stepwise Abstraction needed for understanding the difficult programs and to decrease the amount of user's operations to 10^{-3} of currently expected amount of operations if vanilla text editors are used.

[1] 著者が日本アイ・ビー・エム　ＳＥ研究所，及び，東京基礎研究所に於いて行なった研究内容を含む。

はじめに

ソフトウエア保守の困難さは，殆どすべて大規模複雑プログラムに原因する。プログラム設計者やプログラマーは，全てのプログラムを理解できる間は予定通り保守作業を終了する。大規模複雑プログラムに遭遇すると事情は全く異なる。問題プログラムは，往々にして重要な機能を担っていて，一方では，設計文書がない，複雑過ぎてすぐには理解出来ない，修正の妥当性／正当性の検証が難しい，修正を完了しても適切な文書化を行えない，などの悪症状を引き起こす。そして，このような状況が発生した時に，自分（本当は，人間の）の能力ではどうにもならないプログラム（大規模複雑プログラム）があることが解り，プログラム理解支援ツールに対する必要性／期待を高める。

ツールが大規模複雑プログラムを処理出来ない（例えば，ユーザ定義のシンタックスや省略時解釈を正確に取り扱うことが出来ない，表示された数多いノードの関連図の解釈と意味解釈が困難である等の）場合，その様なツールは，プログラム保守プロジェクトに於いて役に立たない。部分的なツール適用は，ツール使用により得られたプラス効果とそのツールを使用する為の準備作業等のマイナス効果が相殺し合う確率が高いからである。これは，プログラムの欠陥によるのではなく，ツールの設計の不完全さによる。

この論文では，こうした問題の背景，新しいプログラム理解プロセス，ＣＡＳＥツールによる理解支援アプローチ，ツール設計，ソフトウエアマイクロスコープ開発とその実験結果について述べる。

問題

ここで考える対象プログラムは，つぎの条件を満たすものとする：

・複数のプログラミング方法論，プログラミング言語，ユーザマクロで書れていてもよい。プログラムの大きさ，構造には制限がい。又，10^3～10^4行の大きいプログラムに関心がある。
・次のどれかひとつがヒットしても，（狭い意味での）プログラム理解活動が要求される[2]。
　　プログラムの一部又は全体の理解
　　プログラムの一部又は全体のコントロールフロー像の把握
　　プログラムの一部又は全体の機能構造把握
　　修正対象プログラムコードの範囲と修正の他の部分への影響範囲調査
　　テストデータ作成，等
・プログラムの実行環境として，現在の実行環境を含めて，他の実行環境も選択出来る。

ここで取り扱う問題は，「ひとりのプログラマーが，テキスト・エディターなどを用いて，プログラム・リスティングを読むことにより上記の活動を成し遂げられない場合，どのようなツールを提供し支援すればよいか？　そのようなツールの機能とユーザ・インターフェスはどのようなものか？」ということになる。

アプローチについて

プログラム理解の活動を図－１のようにモデル化する。第１スッテプは，プログラムの低レベル（物理）情報の把握とプログラムの基本構造情報の導出がなされる。この情報に基ずいて，次のステップでは，プログラム機能構造の同定／理解，第３ステップでは，それぞれの目的に応じて内臓機能の同定／把握へと進む。ここで，後続ステップは，必ずその前の全てのステップをその都度辿るのが普通である。大規模複雑プログラムの場合，この前半の作業の複雑さ／難しさと後半の抽象化作業のための（関連情報を探す等の）大きい情報検索ができないと言う問題を解く必要がある。最初の問題は，ツールの分析／表現機能により解決し，後の問題は，ユーザインターフェスと抽象化示唆機能に於いて解決する事を考える。

[2] ここでは，簡単の為に，狭い意味でのプログラム理解を考える。広い意味でのプログラム理解は，類似機能の同定，機能の内包関係，性能評価など大局的情報に関する理解を指す。

上位機能の詳しい情報は，基本表示上に追加情報として表示（拡張表示）する。レコード型の物理イメージと使用する変数だけにサブセッテングしたデータ構造イメージをユーザに提示する。この情報とステートメントの情報（オペレーションコード[5]）の組み合わせ（データ・アグリゲーション）を作り，それぞれの組み対してひとつの色を割り付ける。基本表示内ノードがどれかのある組みのデータを参照する場合，そのノードに，割付けた色を付ける。この色付きコントロールフロー図は，１）プログラム実行順序，２）データブロックに対応するステートメントの所在場所　を同時に表示する。この関心のある色が集中している領域が内部機能として認識してよい。このようにして，上位の機能を見つける事を容易にする。理解作業中に重要である思うノードにマーキング（印をつけること）を付けることは，後々の記録や文書化に役立つ。便利な機能である。

文書化の重要な第１の目的は，プログラム理解に必要と思われる情報を表示している画面のイメージをメモ付きで保存し，後の参考データとする事である。第２の目的は，プログラム設計情報としてツール出力を利用する事であり，また，これはプログラムの現状診断である。ツールユーザインターフェス上で，これらの作業が簡単に出来るように機能とマウス・ボタンが設計されている。

ハードウエアとしては，強力なＰＣまたはワークステーションを選ぶ必要がある。高解像度カラーグラフィックディスプレイとカラーグラフィックプリンターを用いて，複数ウインドウにテキスト，ダイアグラムの両方を同時表示する。十分大きなハードディスクも重要である。

プログラム理解支援ツール：ソフトウエアマイクロスコープと実験評価

一つの例を示す。このプログラムは，約10^4行の文からなる。複雑なアプリケーション　ルールに従って入力データの正当性検査を行い，正しければ所定の計算を実行して値を得るプログラムである。約１０数年間の保守を経過しており，結果としてプログラム自体もかなり複雑になっている。

前に述べたことを反映したプロトタイプツール：ソフトウエアマイクロスコープを，ＩＢＭ　Ｓ／３７０　アッセンブリー言語で書かれたプログラムの理解支援を目的に研究開発した[6]。ツールシステムの環境設定，プログラム分析，およびプログラムブラウザーの３つの機能コンポーネントから成り（図ー３），環境設定環境では，ユーザが，基本テーブルの設定，更新を簡単にできる。プログラム分析への入力はコンパイル・リストである。

マクロの定義を基本テーブルに対して行なうことは高々１０分ぐらいで完了する。プログラム分析を約数時間程度で完了する[7]と，そのプログラムの基本表示をプログラム・ブラウザーを起動して表示することができて（図ー４），速やかにプログラムの基本構造を知ることが出来る。大規模複雑プログラムのコントロールフロー図は，かなり大規模で複雑になると考えられるので，詳細レベルと荒いレベルの２種類のレベルの表示を用意した。詳細レベルのコントロールフロー図の中に現れるプロパーサブグラフを適当に縮退させて得られる荒いレベルのコントロールフロー図は，サブプログラムの開始／終了ノード，サブプログラム間の機能干渉の有無を簡単に示す事が出来る。

サブプログラム呼出し関係図からは，サブプログラムの呼出しレベル数が（外部サブプログラム[8]呼出しを除くと）２レベルになっていることが解る。全体コントロールフロー図からは，１）かなりのデッドコードが含まれている，２）通常のサブプログラム呼出し／戻りのインターフェスは使われておらず，直接に親サブプログラムに戻る場合が多い，３）正常処理は中央最右端に出ているフロー線で表されている処理で，そのほかのフローは，ルール適用不可能条件発生を示している，などが数十分程度で把握できると思われる[9]。図ー５は，これらのフロー図を拡大したものである。

5　コンパイラーがクロスリファレンスに表示しない変数等を含む。
6　このソフトウエアマイクロスコープの設計は，３ＧＬ相当の高級言語に対して，適用出来る。
7　中規模ワークステーションを使用した。
8　枠の太い箱内に示されている名前が外部サブプログラムを表す。
9　この時間は個人差があるので厳密には決められない。

プログラム分析／表現は，複数版の方法論，表現方法，プログラミング言語に透過的で，常にプログラムの論理構造を表現するものでなければならない。プログラムが依って立つところの情報：プログラム枠組み情報構造は，コンパイラー，アセンブラー，リンカー，及びＯＳによるプログラム実行において必要な情報をベースに，図－２に示す構造モデルを考えた。プログラム全体を表す名前，プログラム外部からインポートする要素名，プログラム内で使用されるデータブロック名とそのブロック内で定義されていて使用されている変数名リスト，サブルーチン名（サブルーチンの最初のノードに付けられた名前[3]）とそれらの呼出し関連情報，サブルーチンのコントロールフローから成る。

段階的抽象化支援は，データ・アグリゲーションを利用して実現する。サブプログラムは，機能を陽に表している。しかしながら，サブプログラム内に含まれる或はサブプログラム間に渡って分散する機能の同定は，プログラム理解とって重要である。構造化プログラミング，抽象データ型等の考え方に従って，意味的にまとまったデータ変数又はブロック，オペレーションを参照するステートメント，命令が，そのレベルの機能を実現している構成要素であると考えて，その集合を可視化する方法を与える。裸のデータ変数や命令だけでなく，それらをまとめているマクロ，或は，もっと詳しい機械語命令，ビットデータ，オペランドなどもこの対象とする。

ツール設計について

大規模複雑プログラムの場合，記憶に頼るプログラム理解ではなく，表示されている情報に基ずくプログラム理解へと変える必要がある。分析結果を表示するユーザインターフェスは，一画面に表示されてる情報に対してどんな操作を行なえるか（どのような追加情報が見られるか）が簡単に解るように設計する。このためには，アイコンの一般化と汎化を最大限に行ない一枚の小さいアイコンテーブルに定義できるようにする。通常のテキストエディターを用いてプログラムを理解する時の情報検索オペレーション量に比べて，少なくとも３桁以上減げる必要がある[4]。この為に，自動ナビゲーションを考える。細かく低レベルな自明な操作はすべてツールが行ない，ユーザは，目的の情報を見つける為には，表示された情報に関して高いレベルの（大きい）操作が出来る。これとズーミング機能を実現することが重要である。

プログラム理解を進める間のツール操作の応答時間を良好に保つことである。共通基本情報は，プログラム分析フェーズの処理として前もって完了しておく。例えば，コントロールフロー図，データ構造や参照情報等である。これらは，ユーザが紹介要求を出してから処理を開始したのでは到底正常な応答時間は望めない。

ステートメント，マクロ，アセンブラー命令，機械語命令等の属性や処理オプションの指定を与える基本テーブルを設け，この基本テーブルに照らし合わせて，大規模複雑プログラムのステートメントを自動解析する。その結果，プログラムの枠組み情報を自動抽出し，その基本枠組み図を自動生成する。この図の組みとプログラムテキストを合わせて基本表示と言い，プログラムの論理設計／論理構造を明示するものである。例えば，コントロールフロー図では，各ノードは分岐のない処理ブロックか条件分岐文を表し，論理的に次に実行されるブロックノード又は条件分岐ノードは，この図の上でも距離的に最も近い場所にレイアウトされる。その結果，サブプログラムの開始／終了ノード，ループ構造，判断の範囲等が簡単に解るような図になり，デッドコードやデッドパスなども，基本表示上で簡単に判別つく。そして，必要ならば，これらの不要なコントロールフローは，ユーザの指示によって，基本表示から取り除く事が出来る。

---

[3] 内部プロシージャ名である。アセンブラリープログラムの場合，サブルーチン・コール命令のオペランドに指定されるＩＤである。

[4] 命令文間の関係は，大規模複雑プログラムの場合，大体，命令数^{2}のオーダーに比例する。10^4行のプログラムを現在（10^2行プログラム理解）の操作量でこなすとすると，関係数の比は10^4となる。実際には，10^3～10^4行プログラムが殆どなのでこの比は，10^3ぐらいである。つまり，ツールを使用すると，プログラム内の関係情報検索は，約10^3倍効率よくなることが求められる。実際，10^2位であると，そのツールの使い勝手は，大規模複雑プログラムに対してはあまりよくない。

図ー6は，"PCICS"（かなり大きデータブロックである）を参照するノードを，サブプログラム呼出し関連図と全体コントロールフロー図に赤色で示しているが，サブプログラム呼出関係図ではよく見えるが，全体コントロールフロー図では細かすぎて見えない。そこで，サブプログラム呼出し関連図から全体コントロールフロー図への連動表示[10]とこれらのウインドウに対するズーミング機能を用いると，図ー7を得る事が出来る。これらの図では，PCICSデータブロックを参照している場所とプログラム実行に関してどのように影響しあうかを表示している。

まとめ

この論文では，大規模複雑プログラム理解における困難な問題，それらを解決する為のアプローチ，ツール設計技術及び実験結果について述べた。一般プログラム保守支援を目的としたプログラム理解支援ツール設計は，大規模複雑プログラム理解支援の可能性の程度で評価される。人間能力限界をはるかに越えているような問題プログラムの理解に役に立つような理解支援ツールは，特定方法論やプログラム言語に依存しない新しいプログラム分析，表現とユーザインターフェスを持ち，更には，テストデータ作成の助けにもなることが解る。一方，ツールのユーザインターフェスに於いても，テキスト・エディターを用いて理解するための操作量に対して，10^3〜10^4分の１程度の操作量で済むように効率を高める工夫が必要になる。ここで紹介したソフトウエアマイクロスコープでは，これらの課題を解決している。これは，また，複数言語で書かれたソフトウエア理解支援ツールに拡張出来る。

参考文献

S.Chen et.al., A Model of Assembly Program Maintenence, Journal of Software Maintenance, Page 3, Vol.2, No.1, 1990

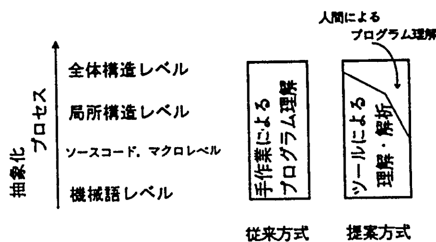
Tomihisa Kamada et.al., A General Framework for Visualizing Abstract Objects and Relations, ACM Trans. on Graphics, Page 1, Vol.10, No.1, 1991

Emden R. Gansner et.al., A Technique for Drawing Directed Graphs, IEEE Trans. on Software Engineering, Page 214, Vol.19, No.3, 1993

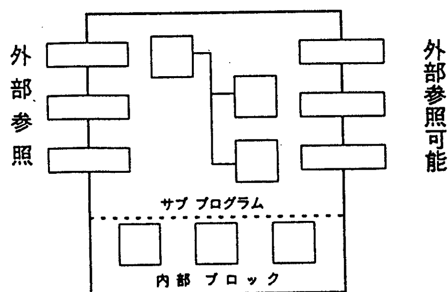
M.H.Williams et.al., Conversion of unstructured flow diagrams to structured form, The Computer Journal, Page 161, Vol.21, No.2, 1976

Robert Boydston, The Effect of Program Complexity on Programming Productivity and Program Quality, IBM Santa Teresa Laboratory, TR 03.071, 1979

図ー1 プログラム理解プロセス

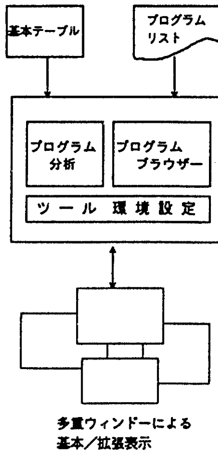


図ー2 プログラム枠組み情報

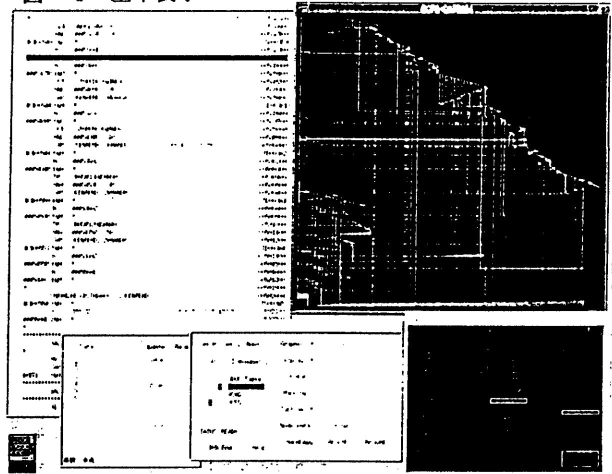


[10] 全体コントロールフロー図に大きい十字印のついているるノードが対応サブルーチンの開始点である。

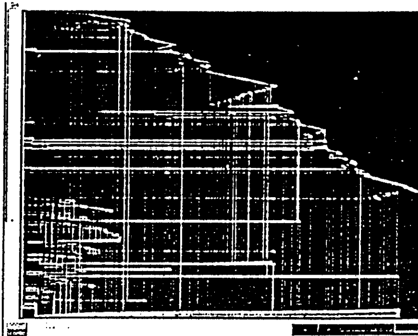
図ー3 ソフトウェアマイクロスコープ



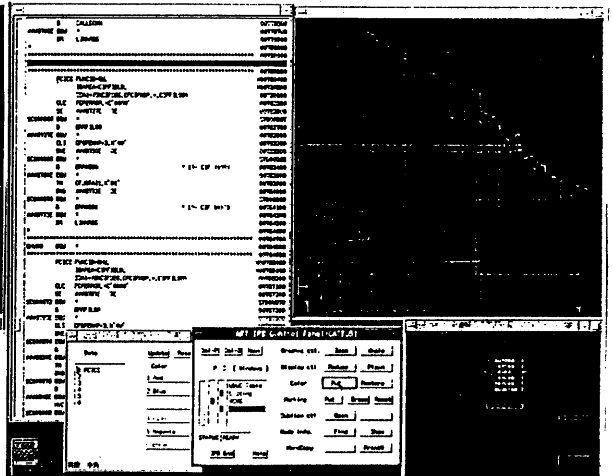
図ー4　基本表示



図ー5　全体コントロールフロー



図ー6　ＰＣＩＣＳ参照



図ー4は基本表示を示す。左側ウインドウがプログラムリスト，右上ウインドウがプログラム全体のコンとローフロー図（図ー5に拡大図を示す），右下ウインドウにサブプログラム呼出し関連図を示す。サブプログラム呼出関連図ウインドウに於いて，左上に主ルーチンの名前の箱があり，中央部にある多数の箱（の中に呼び出されるサブルーチン名が示されている）に有向線で結ばれている。図ー6に於いては，先ず，右上下のウインドウに，ＰＣＩＣＳを参照しているノードに赤色を付けているが，上のウインドウではそれらのノードを確認できないので，ひとつのルーチン名を右下のウインドウで選択し（十字印），ナビゲーションを利用してその先頭ノードを上のウインドウで十字印の所に見つけている。この部分をズーミングすると図ー7が得られる。

図ー7　コントロールフロー図のズーミング