ソフトウェア工学 106－7
（1995．11．17）

ソフトウエア・マイクロスコープ

(大規模複雑プログラムの自動プログラム分析と可視化)

秋山義博

金沢工業大学 人間・情報・経営系、情報工学研究科

大規模複雑プログラムは，その設計文書を作成することが難しい，複雑なので理解するのが困難である等の問題をソフトウエア保守に於いて顕著に示している。ソフトウエアは、一般的に不特定複数ログラム設計方法論や複数のプログラミング言語レベルを利用して開発されている。これに加えて複数プログラマーが（恐らくは局所的な修正やテストなどを繰り返す手段を用いて）*個性的に*保守を繰り返している。従って、このようなプログラムの理解支援を考える場合、プログラマーの助け無しに、その*基本構造*や*基本設計*を浮き彫りにする分析，表現，可視化技術が必要になる。現存するこのような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為のツールの設計とそのプロトタイプ：ソフトウエアマイクロスコープ（Software Microscope）ついて述べる。このツールはこのような問題を解決することを目的とし、更には、プログラム情報検索のためのユーザオペレーション量を、通常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大幅に（10^{-3}以下に）減らすことを実現している。ＩＢＭ Ｓ／３７０アッセンブリープログラムについてこれらの課題を解決した。

Software Microscope

(Automatic Analysis and Visualization of Very Large and Complex Programs)

Yoshihiro Akiyama

C&N Core, 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 helping program understanding and the CASE tool called Software Microscope which does automatic analysis and visualization of difficult programs are proposed. The design and functions of the tool prototype are also presented.

It is noted that the design concept of Software Microscope does not depend on particular methodology or programming language, but is based on general one to which such software programs are analyzed and visualized automatically. It is also important to note that the tool automatically reveals and visualizes the fundamental structure of programs without expecting user's help. In addition, the new tool user interface is simple enough to help identifying functions which may be implicitly expressed over subroutines in difficult programs. The amount of user's key operations is decreased to 10^{-3} order of currently expected amount of operations if vanilla text editors are used.

はじめに

ソフトウエア保守の困難さは，ほとんどの場合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、保守対象のプログラムを自分で理解できる間は予定通り保守作業を終了し特別のツール等を必要としない。大規模複雑プログラムに遭遇するとこの事情は全く異なる。容易に理解出来ない、設計文書がない，修正の妥当性／正当性の検証が難しい，等など症状が発生した時に，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があることを悟り，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。この論文では，こうした問題の背景とプログラム理解支援ＣＡＳＥツール：ソフトウエアマイクロスコープについて述べる。

問題と課題

実際に現在数多く存在し重要な役割を現在果たしている大規模複雑プログラムの特徴的パラメータとしては次に挙げる変数と値と取ることが多い：

- ◆行数（10^4 のオーダ）、
- ◆データ変数数（10^3 のオーダ）、
- ◆サブルーチン数（1以上）、
- ◆外部参照変数数（0以上）、
- ◆ブランチ命令割合（２０～３０%）等

命令数及び条件分岐命令数が多いことからプログラム複雑度が極端に高くなっていることが解る。データ変数も正確には構造化（正規化）されてはおらす機能的干渉が大きい事を示す。

大規模複雑プログラムを自動処理出来ないツールは殆ど役に立たない可能性が高い。ツール使用のプラス効果と大規模複雑プログラムに対してツール使用不能のために要するツール利用準備手作業のマイナス効果が相殺し合う確率が高くなるからである。これは、プログラムの欠陥によるのではなくそのように部分的適用しか出来ないツール設計の不完全さによる。

長期間保守を受けているプログラムは、複数方法論やプログラミング言語、マクロ等を混合使用して開発保守されている。このような保守を助けるためには、ヘトロジニアス（異質複数）な方法論とそれらを用いて開発されてきたプログラムの理解支援を指向しなければならない。このヘトロジニアス（異質複数）な方法論を内臓したソフトウエア保守方法論とその支援ＣＡＳＥツールが無い以上は問題は解決しない。従って、現在のフォワード型ソフトウエア工学技術だけを利用してソフトウエア保守の支援環境を構築するのは難しく、それらを含んだいソフトウエア工学技術体系が必要である。歴史的には、ソフトウエア工学の研究は、下流工程における設計・プログラミング支援技術から上流工程の分析・設計支援を狙ったサポートへと範囲を拡大したが、プログラムの保守が当初の予想に反して重要な問題になってきている。

更にチャレンジ的なことは、大規模複雑プログラムの文書化支援である。文字通り、紙と鉛筆（最近はワープロやＤＴＰなどに変わってきたが）を用いて行う作業であった。記述量が多く複雑で気が遠くなることに加えて同様の作業を何度も行った経験を持つ人は多い。大規模複雑プログラムの場合にはその様な文書すらないことが多く、すべての作業がソースコード上で行われなければプログラマー間の連絡も取れない状況になると容易に想像がつく。この問題は現在殆どお手上げ状態であると言ってよい。

プログラムソースコード周辺のソフトウエア工学

大規模複雑プログラムの保守支援をソフトウエア工学の体系の中に入れる為には、構造化分析・設計・プログラミングやオブジェクト指向分析・設計・プログラミングなどの全体をひとつのソフトウエアの世界として支援する必要がある。この意味で、大規模複雑プログラムについて考える。ソースプログラムに関係した代表的プログラミング作業を以下にリストする。

- ◆プログラム構造分析：プログラムの構造・機能・インターフェスの現状分析・可視化を行う。実現情報を隠蔽した表現を突き止める。
- ◆プログラム修正：ソースコードの修正場所とその修正の他への影響を正確に予測し正しさを確認する事を支援する。
- ◆コードインスペクション：”意図した通りにソースコードが書かれている”

ことを確かめる手順&方法論。ソフトウエアシステム又はプログラム設計書に対してソースコードの整合性を検証する作業である。

◆テスト：コントロールフローに沿ってデータ変数とレジスター内容の検証等を行う。

◆プログラム文書化：ソースコード表現とは別に”標準化された”可読形式（図やテーブル形式が望ましい）で全体構造、修正場所、修正機能&インターフェス等の情報を提供する。

明らかにプログラム理解支援はこれらの共通（基本）支援になっていて、そのようにプログラム理解支援ツールのユーザインターフェスと機能を設計することがキーとなる。そして、プログラム理解の単位は、上に挙げた活動の単位でもなければならない。今回の場合、〔非構造化＋構造化〕アプローチに共通なこのような単位は一つのプログラムファイル（ソースファイル）が具合が良い。

従って、ここで考える問題は次のように言うことが出来る： 機械語、プログラミング言語、システムとユーザマクロ言語（定義）、実行環境（前提ソフトウエアやＯＳなどの機能&インターフェス）等の情報利用を仮定し、「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・理解することが難しく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？」であり、対象プログラムは〔非構造化＋構造化〕手法、ユーザマクロ、コーディングルールも新旧形式取混ぜて使用する様なプログラムとする。

ソフトウエアマイクロスコープのアプローチ概略

プログラム理解活動の概略を上記のプログラミング作業に関して考える。「プログラム理解活動とは、ソースプログラムに実現されている構造を把握し、その構造に対してアプリケーション的機能・意味（これはプログラマーが事前に知っている）をマッピングすることが出来れば良い」として考えることにする。この前部分はツールの自動処理で、後半は自動処理結果に基づくプログラマー作業支援で実現出来れば良いが、特に前半は重要である。

図ー１は**プログラム理解プロセスモデル**である。第１層ではプログラムの低レベル（物理）情報の同定、第２層ではその情報に基づいてプログラムの基本構造情報の導出がなされる。第３層ではプログラム機能構造の理解，第４層ではそれぞれの目的に応じて内臓されている機能の同定／把握を行う。大規模複雑プログラムの場合，この前半の”認識”作業の複雑さ／難しさをツール分析／表現機能により解決し，後半の理解と抽象化作業を，ユーザインターフェスと抽象化示唆機能を設けて解決する事を考える。

ここで考えるプログラム表現は，複数方法論，表現方法，プログラミング言語に対して適用可能でなければならない。ひとつの方法はそれらのスーパーセットを定義することが考えられるが、ここではそれとは異なるアプローチ：図ー２に示すような共通で普遍性の高い**プログラム枠組みモデル**（プログラム枠組み情報）を取ることにする。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名前）とそれらの呼出し関連情報、命令文、機械語、サブルーチンのコントロールフロー等はここで考えている方法論、プログラミング言語、命令セット、プログラミングルールに共通している。これに従って第１層と第２層の機能を実現する。

第３、４層では、プログラム基本構造のグラフィック表現・表示機能とプログラム抽象化機能ブロックの同定に関する対話型支援を考える。ユーザがプログラムの基本構造を簡単に認識することが出来る為には、大規模複雑プログラムの場合、特にプログラムの意味的構造をエンハンスするようなグラフィック表現でなければならないし、又、その全体イメージを表示出来るものでなければならない。第３層については、

◆全体像を１画面に表示する、

◆各要素が細かになり直接目で追えなくなることが発生する場合でも可能な限

りそのイメージ上でプログラム内部構造の認識が出来る
・要素間関連を表示する等

が必要になる。第4層では、更にデータアクセスに関するアグリゲーションを提供することによりプログラム内部に隠れた機能の所在を同定することを容易にする。サブプログラムは機能を陽に表しているがサブプログラム内或はサブプログラム間に渡って分散する機能を見つけることを支援する。これは、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，命令の集合を可視化することで対処する方法を考える。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出来る。

ソフトウエア・マイクロスコープの実現技術

ソフトウエアマイクロスコープは、IBM S／370 アセンブリー言語でかかれたプログラムの分析・表示を行いプログラム理解の支援を目的として研究開発したプロトタイプである。図ー3にその概略を示す。

入力はコンパイルリスト形式のソースコードと機械語命令、SVC、OSとユーザマクロ等の定義テーブルで、命令や文の構造と属性や処理オプションの指定を与える基本テーブルある。これらのテーブルは、入力ソースコード解析に対してアプリケーション処理仮想計算機を定義し各命令文の解釈評価する為のもので、SVC定義はOSのタイプによるので用意してある。

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しプログラムの枠組み情報を自動抽出（統合プログラム分析を利用）し、基本枠組み図を自動生成（統合プログラムブラウザーを利用）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。ここで基本出力情報としては、コントロールフローグラフ（CFG）、サブルーチン呼び出し関連図（WCWG）、標準化ソースリスト、プログラム構造情報（制御文、コール文、リターン文、マクロ文とその展開文、データブロック構造と使用変数等）である。図ー4にその概略イメージを示す。例えば，コントロールフロー図（右上）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を距離的に出来るだけ最も近い場所に配置する。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，ユーザが基本テーブルで指示すれば，基本表示から自動的に取り除く事が出来る。

ソースプログラムの上位機能の詳細情報は基本表示上に拡張表示する。コントロールブロックやレコード形式にまとめられている変数名やその集合名のそれぞれ又は幾つかの組み（データ単位）に対してひとつの色（色単位）を割り付ける（右下）。一方ステートメントの情報（オペレーションコードも含めて）と突き合わせて指定したデータ単位をアクセスしているノードに色単位として指定した色を付ける。例えばこの色付きコントロールフロー図は，

・プログラム実行順序，
・データ単位をアクセスするステートメントの所在場所

を同時に（可能な限り）1画面に表示する。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される。このようにして，上位の機能を見つける事を容易にし、記憶に頼るプログラム理解から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ことが出来ると期待する。

表示情報に対して可能操作（どのような追加情報が見られるか）が常に簡単に解るように設計してある（中央下）。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、画面上で必要なメモを書き込める（この機能は実現中）等は役立つ便利な機能である。更にプログラム情報を表示しているウインドウを最少化してもすぐにそれを見分けられるようにアイコンの個性化（同じアイコンイメージを使用する場合には別アイコンテキストを指定する）とひとつの小さいアイコンテーブルに定義する。

キー操作量を、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場合に比べて少なくとも千分の１以下に減げる事を目標にし[1]、テキスト、ノード検索について自動ナビゲーションを実現した。自明な操作はすべてツールが行い，ユーザは目的の情報を見つける為の高いレベルの（大きい）操作をすることが出来る。これとズーミング機能を実現すること、それらの応答時間を良好に保つことが工夫されている。例えば、共通基本情報はプログラム分析フェーズの処理で前もって完了しておいて、再処理を避けるようにしている。

文書化支援支援は，プログラム理解に必要な情報が１画面で表示出来ているので、その画面イメージ又はそれぞれのウインドーイメージ又はファイルの印刷をボタン・プッシュにより得られるようにしてある。数日かかる手作業を即時に簡単に完了出来る。プログラム設計情報としてもこの印刷出力も利用出来る。更に、ツールシステム状態保存機能を利用すると、この最終画面状態を保存することが出来て、次回セッションにこの画面から再開出来る。

ハードウエアは，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。

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問題と課題，それらを解決する為のアプローチ，ソフトウエアマイクロスコープの設計、機能と効果について述べた。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程度で評価される。ソフトウエアマイクロスコープは、人間能力限界をはるかに越える機械的作業を伴う大規模複雑プログラムの理解に役に立ち、特定プログラミング方法論やプログラム言語に依存しない新しい（自動化）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，この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程度で済むように効率を高めてあることが解る。

謝辞

日本アイービーエム東京基礎研究所とＳＥ研究所に於いてこのプロトタイプ実現に協力頂いた研究員諸氏と、プロジェクト実現についての強いご支援を提供された刈部英司元ＳＥ研究所所長、高柳繁雄金融営業企画ＩＰＣ部長に感謝します。

参考文献

S. Chen et. al., A Model of Assembly Program Maintenance, 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 of for Drawing Directed Graphs, IEEE Trans. on Software Engineering, Page 214, Vol. 19, No. 3, 1993

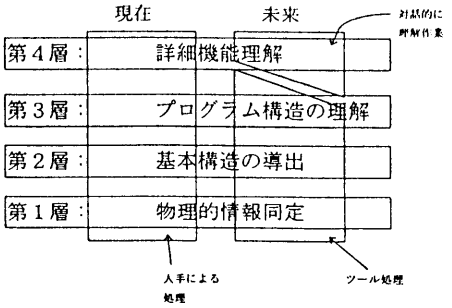
M. H. Williams et. al., Conversion of Unstructured Flow Diagrams to Structured Form, The Computer Journal, Page 161, Vol. 21, No. 2, 1979

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

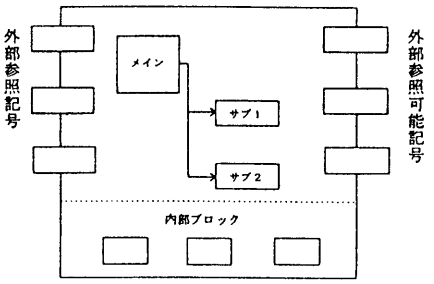
秋山義博、大規模複雑プログラム理解のためのプログラム分析と可視化技術、情報処理学会研究報告、95-SE-102, Jan. 1995

---

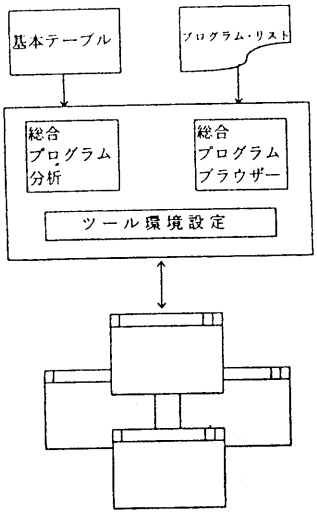
[1] 命令文間の関係は，大規模複雑プログラムの場合，大体，命令数[2]のオーダーに比例する。10^4行のプログラムを現在（10^2行プログラム理解）の操作量でこなすとすると，関係数の比は10^4となる。実際には，10^3～10^4行プログラムが殆どなのでこの比は，10^3ぐらいである。つまり，ツールを使用すると，プログラム内の関係情報検索は，約10^3倍効率よくなることが求められる。実際，10^2位であると，そのツールの使い勝手は，大規模複雑プログラムに対してはあまりよくない。



図－１　プログラム理解プロセスモデル

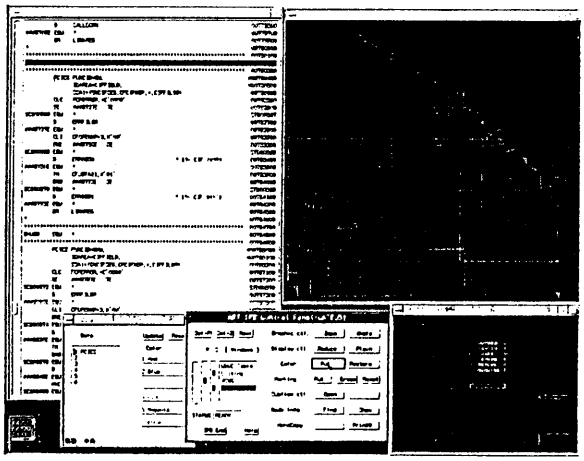


図－２　プログラム枠組み情報



多重ウインドウによる基本／拡張表示

図－３　ソフトウェア・マイクロスコープ



図－４　基本表示