プログラミング－言語・基礎・実践－ 15－4
（1994. 1. 20）

オブジェクト指向ソフトウエア開発におけるプログラム理解支援

中村 宏明　安田 和　大平 剛　三ッ井 欽一

日本アイ・ビー・エム (株) 東京基礎研究所

オブジェクト指向ソフトウエア開発は反復的・探索的な方法で行なわれるので、分析・設計に関する情報を得るためにプログラムを理解するプロセスが重要である。しかし継承・多相性などの性質のため、オブジェクト指向プログラムを理解することは難しい。我々はプログラム理解を支援するための要件をプログラム知識・抽象化・視覚化・理解方法の知識・対話性であるとして、これらをすべて満足するように理解支援システムを構築することによって、プログラム理解の難しさを軽減することを試みた。我々の支援システムの特徴は、問い合わせ処理とユーザインタフェースの変更・拡張が容易な構造になっていることである。また、手法の有効性を実際の使用例によって示した。

Program Understanding Support for Object Oriented Software Development

Hiroaki Nakamura, Kazu Yasuda, Tsuyoshi Ohira, and Kin'ichi Misui

IBM Research, Tokyo Research Laboratory

In the iterative and exploratory style of object-oriented software deveopment, it is necessary for a programmer to understand programs in order to build a working model of the system's design. However, it is not an easy task to understand object-oriented programs because of their natures including inheritance and polymorphism. We developed a tool to help programmers understand object-oriented programs. The tool was designed to satisfy the following requirements: knowledge of programs, abstraction, visualization, knowledge of the way to understand programs, and interactiveness. The main feature of our system is that its query processor and user interface can be customized easily.

1 はじめに

オブジェクト指向によるソフトウエア開発では共通のモデルを用いて分析・設計・実装を行なうことができる。各フェーズが継目なくつながることで、ソフトウエアの変更・修正・追加・再利用などの容易化が期待できる。この利点を生かすためにはフェーズ間を逆方向へ戻れることも要求され [7]、特にプログラムを理解することによって分析・設計に関する情報を得ることが、反復的・探索的なソフトウエアの開発の鍵になる [6]。具体的には、

- クラスに基づいたソフトウエアの管理 [5]
- クラス・ライブラリの選択と利用
- フレームワーク [12]・デザインパターン [4] の抽出と適用

などのために、プログラムを理解するプロセスが従来のソフトウエア開発にまして重要になる。従来より、メンテナンスはソフトウエア開発業務の中で大きな割合を占めることが知られているが、オブジェクト指向ソフトウエア開発ではメンテナンス的な作業の割合が増え、それにしたがってプログラム理解の必要性が高くなるということができる。

ところが次のようなオブジェクト指向本来の性質のために、プログラムを理解することは簡単ではない。

- 継承は差分プログラミングを促進し、このため記述するコードを少なくするのに役立っているが、差分プログランミングは本来ひとまとまりの機能・構造を断片化・分散化することになる。
- 多相性と動的束縛は、概念の分類と統一やプログラムに拡張性をもたらすために重要であるが、プログラムの構成要素間の関係をより複雑なものにする。

また、よく使われているオブジェクト指向言語がもたらす機能やそれによるプログラムのスタイルも、プログラムを理解することを難しくしている。

- プログラム言語 C++ では、クラス・メンバー・インヘリタンスの他に、テンプレート・例外処理・自動型変換・多義演算子・参照型などが導入されている。これらはプログラムの記述を簡潔にするのに役立つものの、プログラムに現れる依存関係の種類を増やし、プログラムを複雑化している。
- Smalltalk では C++ に比較すると言語要素は少ないが、変数に型がないために、動的束縛がもたらす依存関係の複雑化は C++ を上回っている。
- オブジェクト指向プログラムは一つ一つの手続きの大きさが非常に小さくなる、したがって手続きの数が膨大になる傾向がある [20]。再利用の観点からもこのようなスタイルが推奨されている [11] が、このためプログラムが断片化・分散化してしまう。

プログラムを理解する操作の例として、次のように呼び出されている C++ の関数が定義されている場所を求めることを考える。

```
table.put(token);
```

1. table の型を調べる。まず大域変数やローカル変数の中に table を探し、その型を知る。table がデータ・メンバーである場合、この関数呼び出しを含むメンバー関数が宣言されているクラスの継承階層の中で table が定義されているクラスを探し、table の型を知る。
2. 関数の多義性を解決するために、同様にして token の型を調べる。
3. table のクラスの継承階層の中で、メンバー関数 put を定義しているクラスを探す。

C などの言語では関数の呼び出し関係は簡単なパターン・マッチで求まるが、C++ では関数呼び出しの追跡に繁雑な操作が必要であることが分かる。

オブジェクト指向の利点を生かしてソフトウエア開発を行なうためにはプログラム理解が重要であるが、一方で、プログラム理解は非常に困難なものになっている。オブジェクト指向ソフトウエアの開発経験・成果物が蓄積されるにしたがって、プログラムの理解の困難さはより大きな障害となる。そこで我々はこの問題を解決するために、オブジェクト指向ソフトウエア開発においてプログラムの

理解を支援するシステムを構成した。本稿では、まずプログラムの理解支援に対する要件を検討し、次に C++ を対象としたソース・コードの理解支援システムの構成を示し、その使用例によって手法の有効性を示す。

2 プログラム理解支援に対する要件

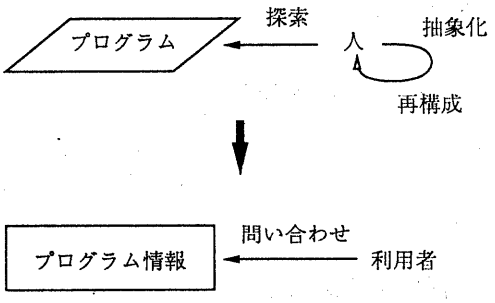

図 1: プログラム理解過程の支援

プログラムの理解は従来から、テキスト表示やパターン・マッチなどを行なうツールを用いてプログラム中の目的の場所の検索を行ない、そこから有用な部分を読みだして抽象化し、設計モデルを再構成する過程として行なわれていた。これらの作業の負担を軽減するためには、プログラム・テキストの検索・抽象化を自動的に行ない、その結果であるプログラム情報に対して問い合わせを行なえるようにシステムを構成すればよい（図 1）。

ここで重要なことは、プログラム情報を分かりやすく利用者に提示すること、プログラムの理解の方法は対象とするプログラム・理解の目的・利用者の嗜好などによって異なるのでこれに対処すること、理解過程にユーザが介在できることなどである。そこで我々は、次のような要素がプログラムの理解支援に不可欠であると考えた。

プログラム知識 プログラムを理解するためにもっとも重要なことは、与えられた個々のプログラムについての知識がなるべく簡単に、また柔軟に検索できる状態で保持されていることである。保持しておくべきプログラムの知識には次のような情報が含まれる。

- クラス・関数・変数・テンプレート・マクロなどの構文要素
- 名前・更新時刻、記憶クラスなどの属性
- 定義・参照・スコープ・継承・メンバー・型など要素間の関係

抽象化 プログラムの理解のために必要な情報は、プログラムに直接現れないものも多いが、何らかの処理をほどこすことによって有用な情報を得ることができる場合がある。これは、人間がプログラムを読んで意図を解釈する過程に相当し、プログラムをより理解しやすい形式に変換する抽象化プロセスとしてとらえることができる。

視覚化 プログラムを抽象化した結果を人間が理解するためには、それを視覚的表現に具体化することが必要である。このとき理解の対象や目的に応じて表示形式を変化させられることが重要であろう。

分離された理解方法の知識 目的等に応じて多岐に渡るプログラムの理解の方法、つまり抽象化と視覚化の方法は、全ての要求に応えられるようにあらかじめ用意しておくことはできない。したがって、理解の方法に関する知識をそれらのプロセスから分離して定式化・保存しておき、これを必要に応じて理解のガイドにとして利用できることが望ましい。

対話性 プログラムの理解過程には人間の介在が不可欠である。これは次のような理由による。

- 理解したい対象をあらかじめ明示的に定めることはできない。
- 大まかな理解と詳しい理解を場面に応じて選択することが必要である。
- 分析・設計に関する情報は実装の際に部分的に欠落するので、プログラムだけからは設計・設計情報を復元できない。

従って、対話的な環境が望まれる。

以上のような検討から、我々は効果的なプログラム理解支援のためのモデルは図 2のようになると考えた。

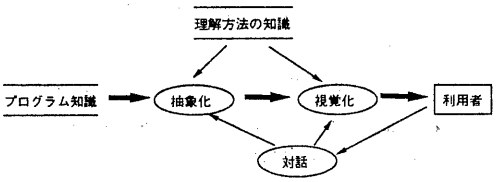

図 2: プログラム理解支援システムのためのモデル

3 理解支援システムの構成

我々は前節で述べた要件を考慮して、図 3のような理解支援システムを作成した。

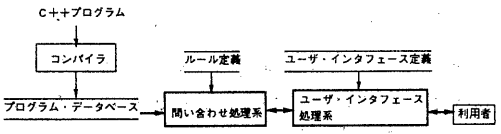

図 3: プログラム理解支援システム

プログラム・データベースには、コンパイラによるプログラムの解析結果を検索しやすい形式に変換したものを格納しておく。当初は関係モデルによってプログラム・データベースを構築したが [21]、現在は他のツールとの統合 [16] やメモリ効率などの観点からオブジェクト指向モデルを採用している [23]。

問い合わせ処理系はデータベース中のプログラムに関する情報を抽象化するのに用いられる。問い合わせの処理には Prolog インタプリタ [17] が組み込まれている。Prolog のプログラムとしてルール定義を与えることによって問い合わせ処理系の機能が決定される。Prolog インタプリタを採用した理由は次の通りである。

- プログラムの追加・変更が容易にでき、拡張性に優れている。
- データベースとの親和性が高い。
- プログラム情報の処理に不可欠な、再帰型の問い合わせを記述できる。

現在のデータベースはオブジェクト指向モデルで構築されているが、データベースとのインタフェースを Prolog インタプリタの組み込み述語として追加してあり、データベースに対する問い合わせはすべて Prolog から発行できるようになっている。

ユーザ・インタフェース処理系は、利用者からの要求を受け付け、問い合わせ処理系に質問を発行し、返答を視覚化する役割を持つ。問い合わせ処理系と同様に拡張性を得るために、ユーザインタフェース記述言語のインタプリタ [22] が組み込まれている。問い合わせ処理系とのやり取りや、コールバック関数などがこの言語で記述できるため、機能の変更・拡張が容易になっている。

4 システムの使用例

前節で構成したシステムを用いて我々が実現したプログラムの理解を助けるための機能をいくつか示し、手法の有効性を示す。

4.1 関数呼出しの追跡

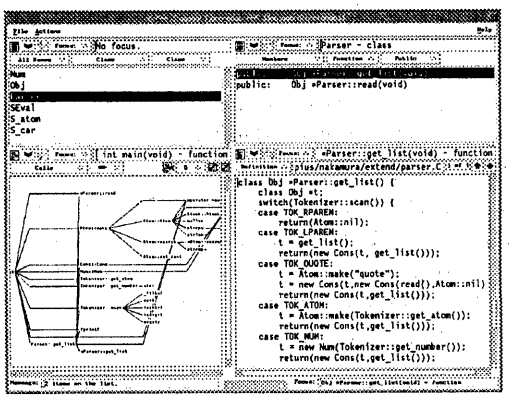

図 4: 関数呼び出しの追跡

プログラムの機能的な側面を理解するためには関数の呼出し関係を知ることが重要である。ところが第 1節で述べたように、オブジェクト指向プログラムでは一回の関数呼び出しの追跡が複雑な上に、関数呼び出しの回数が非常に多くなる。このため、我々は次のような複数の形式で関数呼び出しを表示することによって、利用者の理解をサポートしている。

- 呼出しの深さなどの条件にマッチする関数の一覧

- 関数呼出し連鎖のグラフ表示
- プログラム・テキスト上での関数の宣言・定義・呼出し場所

さらに、これらがハイパーテキスト状に結合されていて、関数名をマウス・クリックすることによって他の表現に移動できる。

図 4で関数呼び出しの追跡例を示す。ここではLisp の処理系を対象プログラムとして、構文解析がどのように行なわれているかを調べている。左上のサブウインドウにはクラスの一覧が表示されているが、この中に `Parser` という名前のクラスが見つかるので、これを選択する。次にこのクラスの外部インタフェースとして、メンバーの中からpublic な関数の一覧を右上のサブウインドウに得る。次に、得られた関数についてその関数呼び出し連鎖を左下のサブウインドウに表示する。また右下のサブウインドウに関数を定義しているテキストを表示する。ここで関数呼び出し連鎖グラフやテキスト中で新たな関数をマウスクリックすることによって、別の関数についての問い合わせを発行することができる。このようにして種々の表現形式を通して関数呼び出しの追跡を行なう。

4.2 クラスの複数の見方への対応

プログラムの構造的な側面を理解するためにはクラスを知ることが重要である。あるクラスについて知ろうとすると、メンバーなどクラス内で宣言されているものを調べる必要がある。しかしクラスが継承を用いて定義された場合、そのクラスを単独で見ても理解できないことが多い。つまり基本クラスをさかのぼらなければクラスについて知ることはできない。またアクセス指定のために、同一のクラスであっても、クラスをどのように使うかという視点によって見え方が異なる。このため、関数呼び出しのときと同様に、クラス定義の追跡を支援する仕組みが必要になる。そこで、我々は次のような機能によってクラスの理解をサポートするようにしている。

- 継承階層のフラット化
- アクセス指定・フレンド関係などによるフィルタリング

さらにクラスの理解を容易にするために有用な組合せをあらかじめ組み込んで、以下のようなメンバーが選択されるようにしている。

- **クラスの利用者の視点** 対象クラスの public メンバーと、public 継承された基本クラスの public メンバー
- **クラスの実装者の視点** 対象クラスのメンバーと、継承された基本クラスの public メンバーと protected メンバー
- **導出クラスの実装者の視点** 対象クラスの public メンバーと、public 継承されたベースクラスの public メンバー

4.3 クラスの全体一部分関係図の簡略化

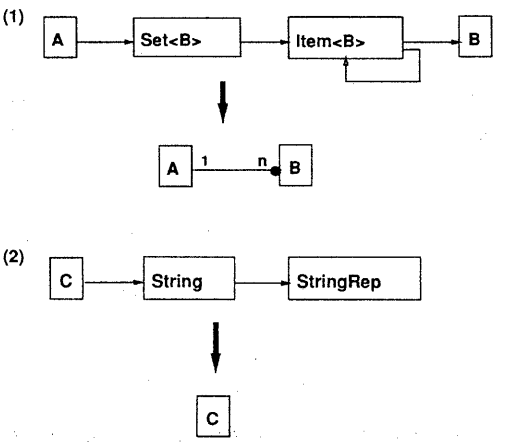

図 5: (1) テンプレート Set の簡略化と (2) クラス String の除去

クラスの全体一部分関係はプログラムの構造を理解するのに重要であるが、これを「あるクラスと、そのデータ・メンバーの型として使われているクラス」というプログラム・テキスト上の関係として定義すると、繁雑で理解しにくい結果しか得られない。これは、1 対 1 でない関連の実装や効率化のために導入される副次的なクラス [19] などを、本質的なクラスと区別できないためである。

あるクラス・ライブラリがプログラムで使用されていることを仮定すると、クラスの使い方のパ

ターンに関する知識を用いて、より高いレベルでの理解が可能になる。たとえば図5のように、(1)集合を実装する目的で用いられるクラス・テンプレート Set はクラス間の線分で表現する、(2)基本的なデータ型を提供するクラス String は良く理解されているので表示しない、という規則を付加してグラフを構成するとクラス間の全体ー部分関係が分かりやすくなる。図6の上のサブウインドウではクラスとデータ・メンバーの型をそのまま用いて全体ー部分関係を表示しているのでグラフが繁雑であるが、図5の規則を適応してグラフを表示すると下のサブウインドウのように簡略化され、全体ー部分関係が分かりやすくなる。

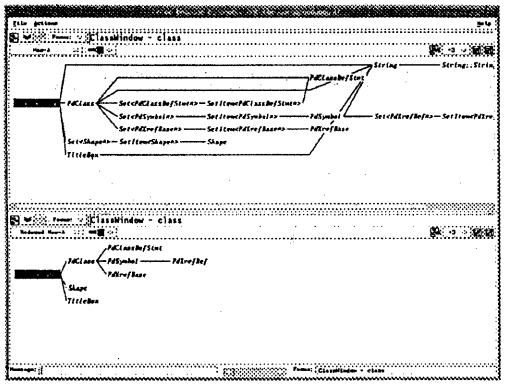

図 6: クラスの全体ー部分関係図の簡略化

4.4 スタイル検査

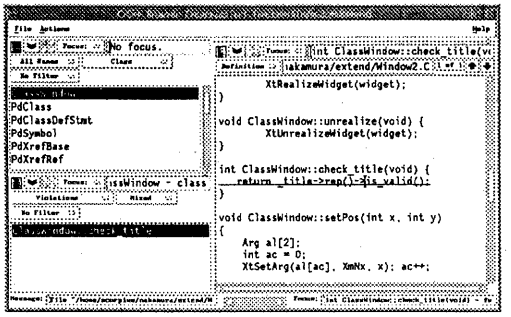

図 7: Demeter の規則の検査

より抽象的なプログラムの理解の方法として、プログラムがあるコンベンションに従って書かれているかを調べることなど、プログラムのスタイルを検査することがあげられる。オブジェクト指向の利点を生かしたソフトウエア開発を行なうためには、プログラム言語だけではサポートできない様々な枠組に則って開発を行なう必要があるので [1][15]、プログラムのスタイル検査は重要な工程の一つである。このような検査の一例として、Demeter の規則 [14] を自動的に検査できるようにした。これは、あるクラスに対してそのメンバー関数が呼び出すことができるメンバー関数が属するクラスの種類に制限をあたえることによって、クラス間の依存性を低く押え、プログラムの再利用・進化の容易性を保証する規則である。この規則に違反する関数がプログラムにある場合、その関数か、それが依存しているクラスを書きかえることが望まれる。ただし書き変え方は一意に定まらないし、プログラマが意図した違反もあり得るので、システムが違反している場所を対話的に指摘してくれることが望ましい。

図 7に Demeter の規則の検査の例を示す。左上のサブウインドウにはプログラム中のクラスの一覧が表示されている。ここで興味のあるクラスを選択すると、左下のサブウインドウには規則に違反しているメンバー関数が表示される。さらにここでメンバー関数を選択すると、違反している場所が右のサブウインドウに表示される。この例では `is_valid()` を定義しているクラスが、選択したクラス `ClassWindow` に許されたクラスでないので違反として報告される。このサブウインドウはテキスト・エディタになっているので、必要に応じてすぐにプログラムの修正ができる。

Demeter の規則の検査を実装するために必要な作業は、規則を 60 行 の Prolog プログラムとして記述することと、メニュー項目を追加するためにユーザインタフェース記述言語で 8 行記述することであった。

5 議論

5.1 関連研究

オブジェクト指向ソフトウエア開発におけるプログラム理解の難しさを問題点として取り上げ、これを克服することを目的としたシステムとして CIA++[8]、XREF/XREFDB[13]、CCEL[3] など

があげられる。

CIA++ はプログラムの構造に関する情報をデータベースに貯めておくシステムで、データベースに問い合わせを行なういくつかのツールと、問い合わせの結果をグラフ表示するツールを組み合わせて用いる。データベースを用いることと、グラフによって情報を表示する点が我々のシステムと共通である。問い合わせツールには、異なったバージョンのプログラムの構文的な差分を求める [9] など、高度な機能を備えているものが含まれる。しかし、問い合わせツールを組み合わせて使う仕組みは提供していないので、新しい機能が必要になるたびにツールをデータベースのクライアント・プログラムとして開発しなければならない。また、ユーザとシステムのインタラクションの仕組みは提供していない。

XREF/XREFDB はプログラムに関する情報を保持しておくデータベースと、テキスト・エディタに統合されたデータベース操作ルーチンからなるシステムである。C++ プログラムに現れる種々の関係を追跡することを編集中に簡単に行なえるようにすることによって、プログラム開発作業をサポートする。我々のシステムと同様に、データベースを用いてプログラム情報を管理し、また対話的な使用に重点をおいている。しかし XREF/XREFDB はテキスト・エディタのみをユーザ・インタフェースとするため、グラフなどの様々な表現形式を採り入れて統合することが難しい。また問い合わせの結果はテキストとの関連で示されるため、設計情報などプログラムを抽象化した情報を扱いにくい。

CCEL は、プログラムの設計・実装・スタイルなどに関する制約を記述し、それをデータベースに保存されている C++ プログラムの情報とつき合わせて、プログラムが制約を充足しているかどうかを検査するための言語である。CCEL とその処理系は、プログラム理解を直接の目的として設計されたものではないが、オブジェクト指向プログラムの読解の難しさを出発点としており、プログラムに直接記述されていない設計情報などを調べることを目指したものなので、プログラム理解支援システムとして解釈することができる。プログラム理解支援システムとして見たときの問題点は、対話的に使用することができないことで、このため、あらかじめ記述した制約の範囲でしかプログラムに関する情報をシステムにあたえることができない。また結果を視覚的に表現する手段を提供していない。

プログラム理解を支援するためには第 2節で述べた要件、すなわちプログラム知識・抽象化・視覚化・理解方法の知識・対話性を欠かすことができない。したがって、プログラム理解支援システムはこれらの要件をすべて満たすように構成されるべきであると我々は考える。

5.2 課題

我々が構成したプログラム理解支援システムの大きな特徴の一つは、理解方法の知識をインタプリタに対するプログラムとして与えることによって機能を変更・拡張できることである。ところが、問い合わせ処理とユーザインタフェースで二種類の言語を用いる必要があり、しかも 2 つの処理が協調するように記述しなければならないので、実際に利用者が機能を変化させるのは簡単でないことが分かった。これを克服するためには、より直観的な方法で問い合わせとユーザインタフェースの記述を行なう仕組みが必要である。Consens ら [2] は、GraphLog という図形を用いて表現する問い合わせ言語を用いて、テキスト表現による処理の記述を不要にするとともに、問い合わせとユーザインタフェースの統一に成功している。この方法を発展させると、システム使用の履歴をそのプログラムの理解を助けるための知識として抽出し、それを直接システムの挙動に反映することによって、あるプログラムの理解支援のために強化されたシステムを自動的に構成するなどということも可能になるだろう。

本研究では、プログラム・テキストだけから取り出される情報のみを対象としてプログラム理解を支援することを行なった。このため、プログラムの動的な側面に関して弱い。Grass[10] は CIA++ を用いて C++ プログラムから OMT の三つのモデルを復元することを試みているが、プログラム・テキストから得られる情報だけを用いた場合、オブジェクトモデルに比較して動的モデルや機能モデルは復元できる範囲が限られることを示している。これを解決するにはプログラムの実行時情報の収集を行なわなければならない [18]。ただしオブジェクト指向ソフトウエアを理解するためには、実行時情報に対してもなにかしらの抽象化を行なう仕組みが必要である。また動的な情報と静的な情報との

組合せも重要であろう。

6 おわりに

本稿では、オブジェクト指向ソフトウエア開発の普及・発展の障害となっているプログラム理解の難しさを解消することを目的として開発したツールについて報告した。プログラム理解を支援するためには、プログラム知識・抽象化・視覚化・理解方法の知識・対話性が重要であるとして、これらの要件を満たすようにシステムを構成した。特に問い合わせとユーザインタフェースをプログラム可能な点が特徴となっている。本研究の成果は製品化され、実際のソフトウエア開発に利用されて役に立っている。

参考文献

[1] T.Cargill: C++ Programming Style, Addison-Wesley, 1992.

[2] M.Consens, A.Mendelzon, and A.Ryman: Visualizing and Querying Software Structure, *Proceedings of ICSE*, 1992.

[3] C.K.Duby, S.Meyers, and S.P.Reiss: CCEL: A Metalanguage for C++, *Proceedings of USENIX C++ Conference*, 1992.

[4] E.Gamma, R.Helm, R.Johnson, and J.Vlissides: Design Patters: Abstraction and Reuse of Object-Oriented Design, *Proceegins of ECOOP'93*, 1993.

[5] S.Gibbs, D.Tsichritzis, E.Casais, O.Nierstrasz and X.Pintado: Class Management for Software Communities, *Communications of the ACM*, Vol.33, No.9, 1990.

[6] A.Goldberg: Programmer as Reader, *IEEE Software*, September, 1987.

[7] S.Gossain and B.Anderson: An Iterative-Design Model for Reusable Object-Oriented Software, *Proceedings of ECOOP/OOPSLA'90*, 1990.

[8] J.E.Grass and Y.Chen: The C++ Information Abstractor, *Proceedings of USENIX C++ Conference*, 1990.

[9] J.E.Grass: Cdiff: A Syntax Directed Differencer for C++ Programs, *Proceedings of USENIX C++ Conference*, 1992.

[10] J.E.Grass: Object-Oriented Design Archaeology with CIA++, *Computing Systems*, Vol.5, No.1, 1992.

[11] R.Johnson and B.Foote: Designing Reusable Classes, *Journal of Object Oriented Programming*, June/July, 1988.

[12] R.Johnson: Documenting Frameworks Using Patterns, *Proceedings of OOPSLA'92*, 1992.

[13] M.Lejter, S.Meyers, and S.P.Reiss: Support for Maintaing Object-Oriented Programs, *IEEE Trans. Software Eng.*, Vol.18, No.12, 1992.

[14] K.J.Lieberherr and I.Holland: Assuring Good Style for Object-Oriented Programs, *IEEE Software*, Septmber, 1989.

[15] S.Meyers: Effective C++, Addison-Wesley, 1992.

[16] K.Mitsui, H.Nakamura, T.C.Law, and S.Javey: Design of an Integrated and Extensible C++ Programming Environment, *Object Technology for Advanced Software*, LNCS 742, Springer-Verlag, 1993.

[17] H.Nakamura: Embeddable PROLOG Interpreter for Data-Intensive Applications, IBM Technical Report, TR-74.091, 1992.

[18] W.D.Pauw,R.Helm,D.Kimelman,and J.Vlissides: Visualizing the Behavior of Object-Oriented Systems, *Proceedings of OOPSLA'93*, 1993.

[19] J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, and W.Lorensen: Object-Oriented Modeling and Design, Prentice Hall, 1991.

[20] N.Wilde and R.Huitt: Maintenance Support for Object-Oriented Programs, *IEEE Trans. Software Eng.*, Vol.18, No.12, 1992.

[21] 中村、安田、三ッ井、S.Javey: 拡張可能な C++ ソースコード・ブラウザ - プログラム・データベース, 情報処理学会全国大会,7Q-06,1992.

[22] 大平, 三ッ井: 拡張可能な C++ソースコード・ブラウザ - グラフィカルユーザインタフェース, 情報処理学会研究報告,93-PRG-10,10-6,1993.

[23] 安田、三ッ井、中村、S.Javey: C++ プログラム・データベース構築, 情報処理学会研究会報告,94-SE-18,1994.