ソフトウェア工学　80－16
（１９９１．７．１９）

設計プロセスを利用した修正支援法について

安達 久人　浜田 雅樹　竹中 豊文
ATR 通信システム研究所

ソフトウェアの設計書の変更に伴う影響の波及解析を容易化する手法について提案する。筆者らは、要求分析工程において、設計プロセスから影響波及解析を行う上で有効になると思われる情報を獲得、参照することで影響波及解析を容易化する方法について検討している。本論文では、本手法をシステム設計工程(データフロー図から構造図を設計)まで拡張する。本手法では、設計プロセスとして、設計者の視点でインデックス付けされた、設計者が設計中着目した設計対象の性質と、それらの間の設計における利用関係を記録する。保守者がこれらの情報を参照することで、データフロー図が修正された場合の構造図への影響波及解析が容易化される。

Software Change Propagation Analysis Using A Design Process

Hisato Adachi　Masaki Hamada　Toyofumi Takenaka
ATR Communication Systems Research Laboratories
Sanpeidani, Inuidani, Seika-cho, Soraku-gun, Kyoto 619-02

A method which facilitates change propagation analysis caused by software design product changes is proposed.The authors are investigating a method which facilitates the change propagation analysis using the recorded design process for requirement analysis phase. In this paper, this method is extended to include the phase of designing structure charts from dataflow diagrams.In this method, design subject characteristics, which are indexed by designer's viewpoints, and design reference relationships among them are recorded as the design process. By maintainer's referring this information, the analysis of change propagation to structure charts caused by dataflow diagram changes is facilitated.

1. はじめに

ソフトウェアのメンテナンス時に行われる要求／仕様の修正の影響を吸収する作業は複雑であり、その効率化が熱望されている。この影響を吸収する作業の複雑さの原因の1つとして、影響波及解析の難しさが考えられる。現在、ソースコードやデータフローダイヤグラム等における文法レベルの影響波及解析の研究・開発は行われているが、意味レベル迄踏み込んだ影響波及解析を行う為の技術は未確立である。

筆者らは、設計工程における要求分析工程を例に、設計プロセスからこの影響波及解析を行う上で有効になると思われる情報を獲得、参照することで影響波及解析を容易化する方法について検討している[10]。本論文では、本手法をデータフロー図(DFD)から構造図(SC)を作成するシステム設計工程まで拡張し、要求／仕様の修正でDFDが修正された場合、構造図(SC)への影響波及解析を容易化する方法について検討する。

以下、まず筆者らが提案している要求分析工程における手法について簡単に説明する。次に、本手法をシステム設計工程に拡張し、データフロー図(DFD)が修正された場合の構造図(SC)への影響波及解析を支援する修正支援方式について述べる。

2. 設計履歴を利用したソフトウェア修正支援 --- 要求分析工程

ソフトウェアの保守中に行われる変更を考えてみる。ほとんどの変更は、以下の何れかに属すると考えることができる。

(1) 原要求に記述されている"設計対象の性質"が変更される場合。ここで、設計対象とは、要求を実現するソフトウェアを指す。また、原要求とは、要求者から見たソフトウェアに対する要求の記述を指す。このような変更として、例えば、機能追加などがある。

(2) 領域知識を用いて設計された設計対象の性質が変更される場合。ここで、領域知識とは、タスク固有の知識(例えば在庫管理など)、一般的な知識、設計に関する知識等を指す。このような例として、例えば、通信プロトコルが変更される場合等がある。

(3) 問題点(バグ)の吸収により設計対象の性質が変更される場合。

従って、ソフトウェア開発における修正作業を支援する為には、以下の解析を容易化する必要がある。

解析1: 原要求に記述されているどんな"設計対象の性質"が設計で利用され、それが設計結果にどのように影響を与えているか。

解析2: 領域知識を用いてどんな"設計対象の性質"が設計され、それが設計結果にどのように影響を与えているか。

解析3: 問題点の原因と問題点吸収による副作用。

しかし、上述の解析の全てを自動化することは、領域知識を形式化した領域モデルのコンピュータ化が必要なため現状では困難である。また、人が行う場合も、膨大な領域知識の中から設計結果に影響を与えたと思われる設計対象の性質を推測するという困難な作業となる。

一般に、設計生産物は、設計対象の性質のある限られた側面を記述したものと考えることができる。設計やメンテナンス中に行われる変更は、設計対象の性質の変更である。その影響は、関連する設計対象の性質間に波及し、その結果が設計生産物の変更として表れる。従って、設計生産物の背後に隠されている設計対象の性質とその間の関係を、設計プロセスを記録することにより捉えることが、上記の解析作業を容易化する上で重要である。以下、まず、設計プロセスのモデルについて述べ、次にそれに基づいて記録した設計プロセスを用いて上記の解析作業を容易化する方法を述べる。

2.1 ソフトウェア設計プロセスモデル

設計プロセスを、設計対象モデルと設計生産物の作成プロセスと定義する(fig.01)。

設計対象モデルは、(1)設計ビューとそれらの間の(2)利用関係から構成される。

(1) 設計ビュー

モジュールを分割する場合、まずそのモジュールを、インターラクション、呼びだし関係、処理ケース等の視点から設計してから行う。このように、ソフトウェアを設計する場合、設計対象に関するさまざまな性質を設計しなければならない。設計者が一度に扱うこれらの設計対象の性質は、人間の認知能力の限界より比較的小さいと考えられる。この限られた大きさを持つ設計対象の性質を設計ビューと呼ぶ。

設計ビューは、設計エンティティとそれらの間の関係の記述と定義する。

設計エンティティは、システム、関数、データ、モジュール等、ソフトウェアの各抽象レベル

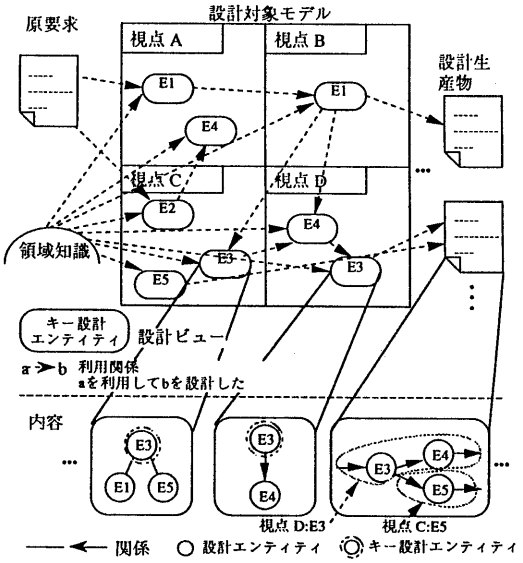

Fig.01 ソフトウェア設計プロセスモデル

での設計を構成する要素を指す。設計ビューに含まれる幾つかの設計エンティティはその設計ビューのキー設計エンティティになる。キー設計エンティティとは、設計者がその設計ビューを設計する際に着目した箇所を表す。例えば、モジュールAの呼びだし関係を設計した設計ビューのキー設計エンティティはモジュールAである。

関係は、ある特定の視点で定義される。例えば、設計ビュー"X店の在庫管理は入庫処理と出庫処理よりなる"は、3つの設計エンティティ"X店の在庫管理"、"入庫処理"、"出庫処理"のpart-of関係を視点"機能構成"から設計したものと考える。

(2) 利用関係

設計ビューXの全体または一部が、設計ビューYを用いて設計された場合、設計ビューXとYの間に利用関係があると呼ぶ。

全ての設計ビューは何等かの領域知識、例えばタスク固有の知識やプログラミング技術等、を用いて設計される。幾つかの設計ビューは更に原要求を用いて設計される。

また、設計者は、設計対象モデルがあるレベルに達した段階で、設計対象モデルの設計ビューを利用して設計生産物を作成する。

この設計プロセスモデルに基づいた場合、2章で述べた修正作業における解析作業の困難さは以下の方法で解決することができる。

解析1 -> 原要求を利用して設計された設計ビューを検索し、更にその設計ビューから設計の展開を追跡する。

解析2 -> 領域知識に関連した設計ビューを検索し、更にその設計ビューから設計の展開を追跡する。

解析3-> 問題が発生した設計ビューを設計する際に利用した設計ビューを検索し、原因となっているものを特定する。更に、特定した設計ビューから設計の展開を追跡する。

設計の展開の追跡は、設計ビューからどんな設計ビューが設計されているかを利用関係に基づき解析することで可能となる。

従って、以下の情報をソフトウェア設計過程で収集する必要がある。

(a) 設計対象モデル

(b) 設計生産物

(c) (a)と(b)間、(a)と原要求間の利用関係

(注)領域知識と(a)間の利用関係を記録するのは現状では難しい。これは領域知識のデータベース化が困難な為である。

2.2 設計プロセスを利用した修正支援

2章で述べた修正のうち、原要求に記述されている設計対象の性質が変更された場合の修正支援方法について述べる(fig.02)。(数字)は図における参照箇所を示す。

1. 原要求が変更される(1)。
2. まず、変更に関連がある設計ビューを以下の方法で検索する。システムが、原要求と利用関係を持っている設計ビューを検索し、その設計ビューの視点の一覧を設計者に提示する(2)。設計者は、提示された視点の中から変更に関係のあるものを選ぶ(3)。
3. 次に、設計者は、得た設計ビューを変更内容に合わせて修正する(4)。

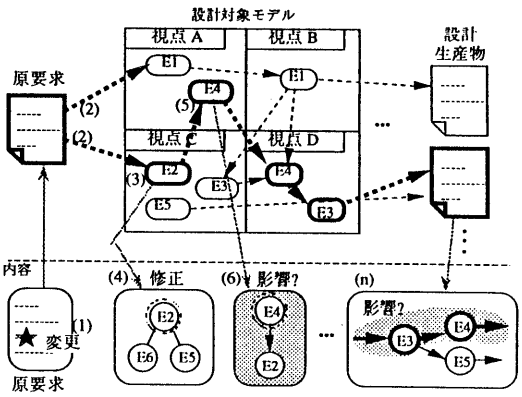

Fig.02 修正支援機能(要求分析工程)

4. 更にその設計ビューから設計の展開を以下の手順で追跡する。システムは、変更された設計ビューを用いて設計されている設計ビューを検索し、設計者に提示する(5)。更に、変更された設計ビューを用いて設計されている箇所がマークされる。見つかった設計ビューを解析し、必要なら修正する(6)。中略
5. 設計者は、修正された設計ビューを用いて設計されている設計生産物を解析し、必要なら修正する(n)。

3. 設計履歴を利用したソフトウェア修正支援 --- システム設計工程

本章では、上記の手法を、データフロー図(DFD)から構造図(SC)を作成するシステム設計工程まで拡張し、DFDが変更された場合のSCへの影響波及解析を支援する方法について検討する。

DFDの変更とは、DFDに記述されているある設計対象の性質が変化する事である。従って、その影響波及解析では、以下の解析作業が必要となる。

解析4: DFDに記述されているどんな"設計対象の性質"が設計で利用され、それが設計結果にどのように影響を与えているか。

以下、変換分析手法[11]を用いた場合を例に述べる。まず、変換分析について簡単に説明し、次に、記録される設計プロセスとそれを用いた修正支援について具体例を用いて説明する。

3.1 変換分析手法

変換分析手法について以下に簡単に説明する。

変換分析手法はデータフロー図(DFD)から構造図(SC)を作成するための手法の１つである。

この手法の目的は、SCを完成させるための雛型を作成することにある。したがって、設計者は変換分析手法によって作成された雛型を再構成することで、構造図を完成させていくという作業を行う。

変換分析手法では、データの流れがプログラムの目的とする機能に向って流れているのか、それとも遠ざかる方向で流れているのかを調べ、それらの流れの境界を決めることでプログラムを分割し、それぞれの部分を、独立した機能として捉える。これにより、プログラムを入力に関する部分、変換に関する部分、出力に関する部分という３つの部分に分割する。そして、それぞれの部分を、モジュールの単位とする。入力に関する部分に対応するモジュールを求心型、変換に関する部分に対応するモジュールを変換型、出力に関する部分に対応するモジュールを遠心型モジュールと呼ぶ。また、それぞれの部分に入るデータ要素をそれぞれ「求心型データ要素」、「遠心型データ要素」と呼ぶ。

3.2 設計プロセス

変換分析手法により、データフロー図(DFD)から構造図(SC)を設計するプロセスを構成する視点とその間の利用関係をFig.03に示す。

本手法では、前述の手法に基づき、DFDの設計プロセスが記録されていることとする。DFDは、データ構造、機能構成およびプロセス・インタラクションといった視点から設計した結果（それぞれは、DFDに記述されている設計対象の性質と考えることができる）をまとめたものということができる。変換分析手法を用いた作業では、これらの設計対象の性質を利用して設計作業を進めていく。以下、Fig.03に基づき設計プロセスを簡単に説明する。

(1)視点" プロセス・インタラクション" と視点" 機能構成" からの設計結果を利用して、データフロー図に描かれているバブル全てに対して、それぞれのバブルがどのような処理を行うのかを調べる（視点" バブルの処理" ）。

(2)視点" バブルの処理" からの設計結果と視点" プロセス・インタラクション" からの設計結果を利用して、処理の中心となるデータの流れをデータフロー図(DFD)から抽出する。（視点" 主要な流れ" ）。

(3-1)視点" 主要な流れ" での設計結果から「処理の目的は何か」を考える（視点" 処理目的" ）。この設計結果を変換機能に当るバブルを見つける際の判断情報とする。

(3-2)視点" 主要な流れ" の設計結果に存在する各バブルとその入出力データに対して、そのバブルの出力データを作成するためには、その入力データだけで処理が可能かどうかを、視点" データ構造" での設計結果と視点" 機能構成" での設計結果を利用して調べる。（視点" 内部処理可能位置" ）。この設計結果は変換機能に当るバブルを見つける際の判断情報とする。

(4)視点" 処理目的" または視点" 内部処理可能位置" からの設計結果を利用して、視点" バブルの処理" からの設計結果のうち、どのバブルが変換機能に当るバブルかを判断する（視点" 該当バブル" ）。

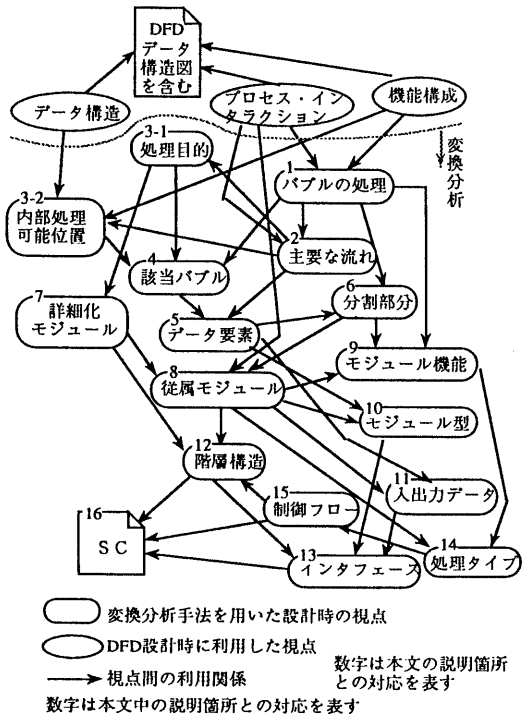

Fig.03　設計プロセス

(5)視点" 主要な流れ" での設計結果と視点" 該当バブル" の設計結果から、該当バブルへの入力データを求心型データ要素、該当バブルからの出力データを遠心型データ要素、該当バブルを中央変換として識別する（視点" データ要素" ）。

(6)視点" データ要素" からの設計結果を利用して（場合によっては視点" バブルの処理" からの設計結果も利用して）、データ要素を基に分割した、それぞれの部分を入力、変換、出力という機能を持つ部分として捉える（視点" 分割部分" ）。

(7) 視点" 処理目的" からの設計結果を利用して、モジュール構造を設計するための基準となるモジュール名を決める。（視点" 詳細化モジュール" ）。

(8) (7)で決めたモジュールの従属モジュールとして、視点" 分割部分" での設計結果のそれぞれの部分（入力、変換、出力）に対応したモジュールを設定し、名称を決める。（視点" 従属モジュール" ）。場合によっては" 視点" プロセス・インタラクション" からの設計結果も利用する）。

(9)(8)で考えた各モジュールがどのような機能を持つか、視点" 分割部分" に属するバブルについて、それぞれ視点" バブルの処理" からの設計結果を利用して考える（視点" モジュール機能" ）。

(10)視点" 従属モジュール" からの設計結果である各モジュールについて、視点" データ要素" からの設計結果を利用して、「求心型データ要素を出力するモジュールは求心型モジュール…」というようにモジュールの型を割り当てる（視点" モジュール型）。

(11)視点" 従属モジュール" からの設計結果を利用して、(8)の各モジュールの入出力データを見つける（視点" 入出力データ" ）。

(12)(7)と(8)を（条件によっては、視点" 制御フロー" からの設計結果も）利用して上位モジュールと従属モジュール間の階層構造を決める（視点" 階層構造" ）。

(13)視点" 入出力データ" として設計された結果から、視点" 階層構造" として設計された上位モジュールと従属モジュールに該当するものを取り出し、これを基に、上位モジュールと従属モジュールの間の入出力データを決める（視点" インタフェース" ）。上位モジュールと従属モジュールのインタフェースは更に、視点" モジュール型" からの設計結果も利用して決めることができる場合がある。

(14)(8)での設計結果であるモジュールと、そのモジュールに対応するモジュールの機能を視点" モジュール機能" での設計結果より求め、それぞれの機能が順次、選択、ケース等のどの処理を行う機能なのかを調べることで、処理のタイプを決める（視点" 処理タイプ" ）。

(15)(14)の設計結果で、処理のタイプが選択やケースとされたモジュールについては、上位モジュールに直接従属する従属モジュールに対し、ある従属モジュールの実行結果によって次に実行されるモジュールが異ると考え、このような状態を表すものとして制御フラグが必要であるという情報を作成する（視点" 制御フロー" ）。

(16)上記(1)～(15)の結果を適宜、構造図として記述する（視点" 構造図" ）。

3.3　具体例

変換分析を用いた設計プロセスの具体例を示す(fig.04)。図は設計技法の比較に用いられる、酒屋の在庫管理問題を設計した例の一部を示している。

(1)設計者は、DFD（キー設計エンティティは「出庫処理を行う」）に存在するバブルについて、視点" プロセス・インタラクション" と視点" 機能

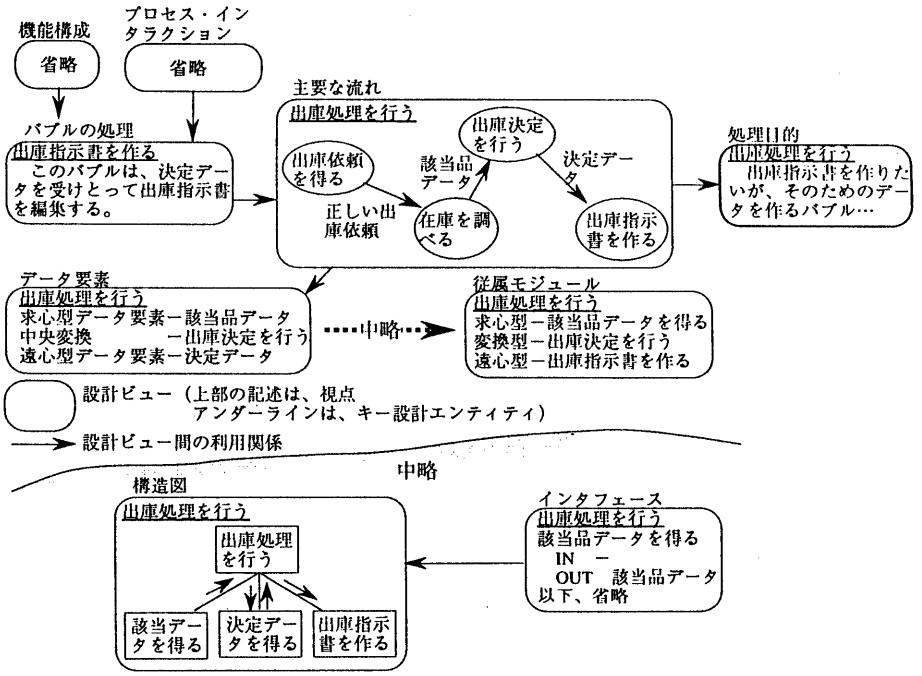

Fig.04　設計プロセスの具体例

構成" からの設計結果を利用して、それぞれのバブルがどのような処理を行っているのかを考えた（視点" バブルの処理" ）。

(2)設計者は「出庫処理を行う」のデータの流れの中で主要だと思われるデータの流れを、視点" バブルの処理" と視点" プロセス・インタラクション" からの設計結果を利用して設計した（視点" 主要な流れ" ）。

(3)設計者は、視点" 主要な流れ" からの設計結果を基に、プログラムが行う処理の目的を考えた（視点" 処理目的" ）。中略。

(4)設計者は、視点" 主要な流れ" と視点" 該当バブル" （Fig.04では省略）からの設計結果を利用して、該当品データおよび決定データで、３つの部分に分割可能であると考え、該当品データは求心型データ要素で、決定データは遠心型データ要素として考えた（視点" データ要素" ）。

(5)設計者は、視点" 分割部分" （Fig.04では省略）からの設計結果を利用して、入力機能部分を、変換分析手法におけるモジュール分割の規則に従い、１つのモジュールとして捉え、「該当品データを得る」というモジュールにした（視点" 従属モジュール" ）。

(6)設計者は、視点" データ要素" と視点" 従属モジュール" からの設計結果を利用して、「該当品データを得る」は求心型データ要素を出力することから、求心型モジュールであると考えた（視点" モジュール型" －Fig.04では省略）。中略。

(7)設計者は、視点" 階層構造" と視点" モジュール型" および視点" 入出力データ" からの設計結果を利用し、「求心型モジュールは求心型データ要素を上位モジュールに出力データとしてわたす…」という知識に基づき、「出庫処理を行う」とその従属モジュール間の入出力データの関係を設計した（視点" インタフェース" ）。

(9)以下、同様な処理を行い、「該当品データを得る」の構造図を作成した（視点" 構造図" ）。

3.4　修正支援機能

DFDの修正は、２章で述べた手法を利用して行われているものとする。即ち、データ構造、機能構成、プロセス・インタラクション、およびDFDが修正されていることとする。

DFDが修正されたことによる影響波及解析作業は、DFD作成時に利用した設計ビューが、変換分析手法を用いた設計結果に、どのような影響を与えたかを解析する作業となる。このような作業を、以下の機能を用いて支援する。

機能１：DFD作成時に利用された設計ビューを利用して、設計を行った設計ビューまたは設計生産物を見つける。

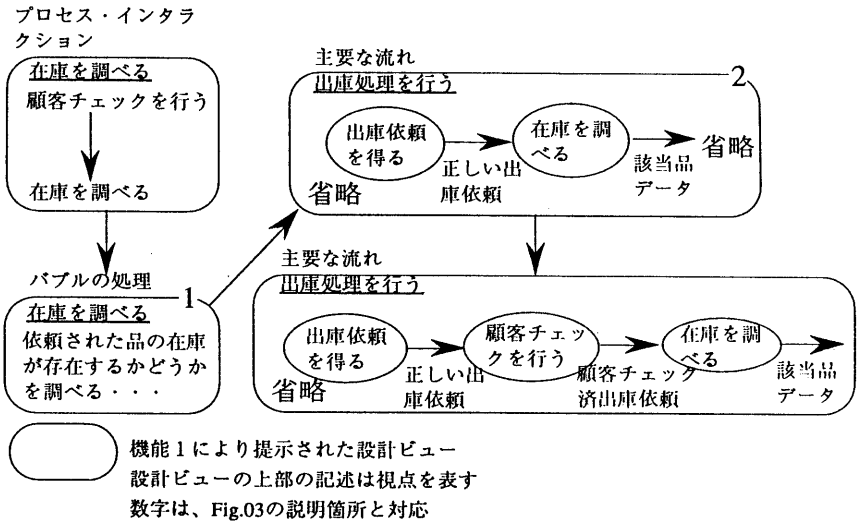

Fig.05　修正支援機能を用いた修正例

このような修正支援機能を、Fig.03の設計プロセスのモデルを利用することで、「設計ビューAが変更された場合には設計ビューBをチェックする必要がある」といったように解析を行うことができる。以下に、DFDが修正された場合の具体例を示す。(Fig.05)

1. 設計者は、「顧客チェックを行う」という機能が追加されたことから、DFD設計時に利用された設計対象のすべての性質（機能構成、データ構造、プロセス・インタラクション）が、変更されたと推測し、まず視点" プロセス・インタラクション" からの設計結果が変更されたことによる影響波及解析を行う。
2. 設計者は機能1より、視点" プロセス・インタラクション" を持つ設計ビューを利用して設計を行っている設計ビューとして、視点" バブルの処理" を持つ設計ビューを得る。
   - 機能1は、Fig.03の設計プロセスに従って設計ビューを検索し、視点" プロセス・インタラクション" からの設計結果を利用して設計を行った設計ビューとして、視点" バブルの処理" を持つ設計ビューを設計者に提示した。
   - 設計者は、視点" バブルの処理" からの設計を修正した。
3. 設計者は、機能1を利用し、視点" バブルの処理" を持つ設計ビューを修正した結果、次に視点" 主要な流れ" を持つ設計ビューをチェックする必要があるとわかり、これを修正した。

以後、上記のような影響波及解析を機能1を利用することで行う。

4.今後の課題

本論文では、設計プロセスを記録し利用することで修正作業における影響波及解析を支援する方法について述べた。実際の設計作業においては複数の人間が、協調して作業を行っている。したがって、本手法をグループ開発における修正支援に拡張して行く予定である。

【謝辞】

日頃ご指導を頂く桑原耕平会長、山下紘一社長に深謝します。また、プロトタイプシステムの開発にご協力いただいた日本電子計算（株）の浜中氏、吉岡氏、辻井氏に深謝します。

【参考文献】

1. L. Osterweil, "Software Processes Are Software-Too," Proceedings of the Ninth International Conference on Software Engineering, 1987
2. Lloyd G.Williams, "Software Process Modeling : A Behavioral Approach, " Proceedings of the Tenth International Conference on Software Engineering, 1988
3. Donald E. Knuth, "Literate Programming," The Computer Journal, Vol.27, No.2, 1984
4. Jeff Conklin, Michael L.Begeman,"gIBIS: A

Hypertext Tool for Exploratory Policy Discussion," ACM Transactions on Office Information Systems, Vol.6, No.4, October 1988

5. Colin Potts and Glenn Bruns, "Recording the Reasons for Design Decisions," Proceedings of 10th ICSE, 1988
6. Zheng-Yang Liu, Arthur M.Farley, "Shifting Ontological Perspectives in Reasoning About Physical Systems," AAAI-90, 1990
7. Brian Falkenhainer, Kenneth D.Forbus, "Setting up Large-Scale Qualitative Models," AAAI-88, 1988
8. Vasant Dhar, Matthias Jarke, "USING TELEOLOGICAL DESIGN KNOWLEDGE FOR LARGE SYSTEMS DEVELOPMENT AND MAINTENANCE," Proceedings of the International Workshop on Expert Systems & Their Applications, 1986
9. 島 健一, "ソフトウェア設計のための判断履歴の蓄積, 利用について", 情報処理学会 ソフトウェア工学研究会 Vol.89, No.101, 1989
10. 浜田 雅樹, 竹中 豊文, "設計履歴を利用したソフトウェア設計・保守支援方式", ソフトウェアシンポジウム'91, 1991
11. Yourdon Constantin, (原田実,久保未沙　訳) "ソフトウェアの構造化設計法"、日本コンピュータ協会
12. Glenford J.Myers　久保未沙　国友義久　訳 "RELIABLE SOFTWARE THROUGH COMPOSITE DESIGN"、高信頼性ソフトウェアー複合化設計、近代科学社