〔公　開〕

ＴＲ－Ｃ－００８５

設計知識の構造化と活用

浜　田　雅　樹
Masaki　HAMADA

１９９３　２．１０

ＡＴＲ通信システム研究所

設計知識の構造化と活用

平成5年 1月

浜田雅樹

ATR通信システム研究所

〒619-02 京都府相楽郡精華町光台2-2

Tel : 07749-5-1242

Fax : 07749-8-2013

e-mail : hamada@atr-sw.atr.co.jp

目次

1. はじめに

交換機，銀行のオンラインシステムなど，高度な通信サービスの実現にソフトウェアは不可欠である．ハードウェアが各サービス間で共通する機能を提供するのに対し，ソフトウェアは，個々のサービス毎に対応して作られる．サービスに対する要求は多様化していく傾向にあるため，ソフトウェアはよく修正される運命を持っていると言うことができる．しかし，ソフトウェアの全開発費の約70%が保守費用であるという調査報告が示すように[13]，現状ではソフトウェアの修正は大きな負担となっている．高度情報社会の実現には，通信ソフトウェアの生産性とともに，保守性の向上が不可欠になる．保守においてソフトウェアを修正する場合困難な作業が，ソースコードとドキュメントにおける変更箇所を限定する解析である．この解析を以下，影響波及解析と呼ぶ．本稿では，影響波及解析を支援する手法を提案する．

ドキュメントソフトウェアのドキュメントは，設計途中で作成する設計生産物を中心に構成される．設計生産物には，一般には設計結果のみが記述され，設計に影響を与えた要因，即ち設計の根拠，は記述されない．ソフトウェアの保守に限らず，設計や再利用を行うには，設計に影響を与える要因に関する情報が必要である．現在，ソフトウェア設計の作業標準を設定するための技術は確立されていないため，設計結果から設計に影響を与えた要因を推測するのは容易ではない．従って，設計，再利用や保守を支援するためには，設計生産物に記述されない情報である設計に影響を与える要因を扱う技術の確立が必要である．現在，これに対して2種類のアプローチが存在する．

1つは，設計に影響を与える要因に関する情報を予め用意しておくアプローチである．例えば，知識ベース技術を用いたソフトウェアの設計や保守支援は，設計に影響を与える要因の候補を予めデータベース化しておく．しかし，現状の技術では，予め設計に影響を与える要因を分析し，矛盾無く記述するのは困難なため，これらのアプローチの適用性には限界がある．

もう1つは，設計に影響を与えた要因に関する情報を設計時に残すアプローチである．これらの研究の目的は，設計の根拠(design rationale)の記録法および記録した設計の根拠に関する情報を，ソフトウェアの保守，設計で再利用する技術を確立することにある．確立すべき技術は以下の要求条件を満たす必要がある．

- ソフトウェアの保守や設計に有用な情報を記録することができる，
- ソフトウェアの保守や設計の作業場面に応じて，必要な情報を検索できる，
- 記録のコストが小さい．

今までに，WEB[3]やIBIS (Issue Based Information System)[4]をベースとした記録方法[5][10][14]などが提案されている．これらの研究のほとんどは，記録形式に関する議論が中心で，記録する情報の効果·妥当性，利用方法や記録のコスト等について議論がされていない(関連する研究について付録1で詳細に述べる)．

筆者らは，要望が高い影響波及解析の支援に対象を置き，上記要求条件を満たす設計の根拠の記録，利用手法を確立した．本稿では，その手法について述べる．

提案する手法では，記録のコストを小さくするため，設計のプロセスを記録する．従来の記録手法では，設計者が設計の根拠を説明する文章を作成する必要が有る．これは，設計以外にもう一つの高度な知的活動が要求されることを意味している．これに対し，提案する手法は，設計の各ステップにおける断片的な設計結果を書き留めるだけなので，設計者は設計活動に集中することができる．以上より，提

案手法は，記録のコスト，設計者の負担感が小さいことが期待できる．

主な研究課題を以下に示す．

(1) 設計プロセスの記録形式の確立．

設計プロセスの記録は，以下の条件を満たす必要がある．

条件1: 設計プロセスの記録が，影響波及解析に有用な情報を含む，

条件2: 設計プロセスの記録は，検索のための索引や関連付けの情報を内部に持つ，

条件3: 設計の思考の流れに沿った記録ができる(人間の設計における認知的側面を考慮した設計プロセスのモデルに基づいている)．

(2) 設計の思考に沿った設計プロセスの記録手法および影響波及解析支援手法(影響波及解析の状況に応じた記録情報の検索手法)．

図1に提案する手法の位置付けを示す．既存のCASE(Computer Aided Software Engineering,コンピュータによるソフトウェア開発支援)システムでは，保守者は，設計生産物を参照することしかできず，影響波及解析は困難であった．これに対し，提案手法により，影響波及解析のコストを軽減できる．具体的には，提案する手法は，既存のCASEシステムに対して以下の機能を追加することおよびその方法を提案する．

·設計者が設計プロセスを記録する機能．

·設計に影響を与えた要因の情報と設計生産物のうちソフトウェアの修正に関連するものを検索し設計者に提示する機能．

以下，まず影響波及解析に有用な情報について述べ(2章)，次に設計プロセスのモデルと記録形式について述べる(3章)．更に，設計プロセスを記録する手法(4章)および影響波及解析支援手法(5章)について述べ，最後に，提案する手法の評価結果について述べる(6章)．

2. 影響波及解析に有用な情報

同じ設計法に基づいた設計でも，その内容(結果，プロセス)は設計の度毎に異なるのは，それぞれの設計において設計対象の性質，例えば，要求されている機能項目やソフトウェアが走行するハードウェアの構成等，が異なるためである．ここで，設計対象とは，要求分析や仕様化工程では，設計しているソフトウェアで実現する機能を，概要設計，コーディング工程ではプログラムを指す．即ち，設計対象の性質が設計に影響を与える要因になる．

ソフトウェアの保守では，ソフトウェアに対して，特定の設計対象の性質，例えばハードウェアの構成等，の変更要求が起こる．変更による影響波及解析を行うには，変更された設計対象の性質に基づいて設計されている箇所を解析する必要がある．設計生産物に記述されている設計結果を基に影響波及解析を行うには，どのような設計対象の性質がどのようにその設計結果に影響を与えたかを保守者が推測する必要がある．この作業の困難さが，影響波及解析の困難さにつながっていると考えられる．従って，設計者がどの設計対象の性質に着目し，それを設計にどのように利用して行ったかを追跡することができれば，影響波及解析を容易化できることが期待できる．以下，本追跡を可能にする設計プロセスの記録形式について述べる．

3. 設計プロセスのモデルと記録形式

3.1 設計プロセスのモデル

多くのソフトウェアの設計法では，設計対象を，概念上の実体とそれらの間の関係として表現する．例えば，構造化分析手法[11]では，設計対象を，プロセス[注1]，データ(以上は概念上の実体)とそれらの間のデータフロー関係として記述する．設計で利用する概念上の実体を設計エンティティと総称する．

一般に，設計すべき設計対象は複雑である，即ち，多くの設計エンティティがさまざまな関係を持つ．人間の認知能力の限界により，設計者が一度に捉えられる関係の種類や範囲には限界がある．設計者は，「データAと依存関係にあるデータは何か？」のように，特定の設計エンティティが持つ特定の種類の関係に着目しながら設計していくと考えられる．今，設計者が設計するために着目した，設計エンティティ間の関係の種類を「視点」と呼ぶ．また，特定の設計エンティティを特定の視点から設計したものを設計ビューと呼ぶ．例えば，データAを視点「依存関係」から設計した(データAと依存関係があるデータを調べた設計)設計ビューと呼ぶ．設計ビューは，設計対象の性質の中で，設計者が設計中に着目し設計したものを表している．

設計ビュー間には依存性があり，設計では互いに考慮する必要が有る．例えば，ある機能項目(機能項目とは，あるデータから別のデータを計算する処理を表す)を視点「機能項目」から設計する(機能項目を更に詳細なものに分割する)ためには，その機能項目で処理するデータを視点「依存関係」から設計した設計ビューなどを考慮する必要がある．このように設計ビュー間に，設計における考慮の関係(「利用関係」と呼ぶ)が存在する．

以上の設計活動の特徴より，ソフトウェアの設計プロセスは以下のようにモデル化することができる(図2)．設計者は，まず，ソフトウェア開発依頼者の要求(原要求と呼ぶ，一般に曖昧さや矛盾を含む)に基づき，要求される設計対象の性質を分析する．これは，原要求を利用して設計ビューを設計することに相当する(図2(1))．次に，それらの結果を利用し，設計生産物の作成で必要な設計ビューを順次設計し(図2(2))，最後に設計生産物を作成する(図2(3))．設計者は，以上の設計において自身が持っている領域知識を利用する．領域知識とは，タスクに関する知識，例えば在庫管理や通信プロトコルの知識等，および設計技術に関する知識を指す．このように，設計ビュー，設計生産物，原要求，領域知識間には利用関係がある．

視点は，以下の2種類に分類できる．

1) 設計エンティティ間に存在する関係を分析する視点．例えば，「依存関係」等．

2) 設計エンティティと特定の関係を持つ設計エンティティを新たに定義するための視点．例えば，「機能項目」等．

トップダウン的に行う設計は，2) の視点からの設計を中心に展開される．その設計で考慮が必要な設計エンティティ間の関係を分析する1)の視点からの設計が伴って行われる．例えば，構造化分析手法に基づいた要求分析[注2]では，視点「機能項目」からの設計が中心となる．視点「依存関係」からの設計等が必要に応じて行われ，その結果が視点「機能項目」からの設計で考慮される．

また，視点には，設計法で具体的な指定があり予め決まるものと，設計時に設計者により認識されるものとが存在する．

3.2 設計プロセスの記録形式

[注1] これは，設計のプロセスとは別のもので，データの処理単位を指す．

[注2] 要求をデータフロー図(データが変換される流れを記述)，データ辞書，データ構造図，E-Rダイヤグラムとして記述する要求分析手法．

設計プロセスの記録形式は(図3参照)，設計プロセスのモデルにおける設計ビュー，利用関係および設計生産物を記録する．

記録形式は，ハイパーテキストの構造を持つ．

(1) 設計ビュー

設計ビューは，内容とインデックスより構成される．

- 内容：①ある設計エンティティ(その設計ビューのキー設計エンティティまたは単にキーと呼ぶ)を特定の②視点から設計したもので，キーと他の設計エンティティ間との関係として記述される．
- インデックス:①の設計エンティティ名と②視点名の対で表す(以下，[キー；視点]で示す)．

設計ビューは，インデックスで識別する．

設計エンティティは，その名称で表し，識別する．

(2)利用関係

設計ビューの内容に含まれる設計エンティティ，関係はそれぞれ「利用属性」を持つ．利用属性は，それを持つ設計エンティティまたは関係を設計した際利用した他の設計ビュー，領域知識，原要求の識別子の一覧を値として持つ．利用関係は，利用属性より間接的に表現される．

例えば，図3の(1)の設計ビューは，キー:「出庫処理」，視点:「機能項目」をインデックスとして持ち，その内容は，設計エンティティ:「出庫処理」，「出庫依頼の入力」，「在庫情報の更新」等のpart-of関係である．これらの設計エンティティ，関係は，利用属性を持っている(例えば「出庫依頼の入力」は，[出庫依頼；媒体]と領域知識を用いて設計されたことを表す利用属性値を持っている)．

(3)設計生産物

提案手法では，設計生産物を設計ビューと同じ形式で記述する．例えば，構造化分析手法における設計生産物DFD (Data Flow Diagram)は，その詳細化階層における各ダイアグラムそれぞれを設計ビューとして表す．その場合，DFDのプロセスやデータなどが設計エンティティに相当し，詳細化階層における親プロセス名がキーに，「DFD」が視点になる．

筆者らは，提案手法のコンピュータ支援系のプロトタイプシステムDIG(Design Information Gathering system)を試作した(smalltalk80で開発，UNIXワークステーション上で稼働)．以下では，設計プロセスを記録する方法および影響波及解析支援手法について，DIGでの実行例を示しながら説明する．

4. 設計プロセスの記録方法

設計者の思考の展開に沿った設計プロセスの記録方法が必要になる．本手法は，以下に示す設計者の自然な活動に沿って設計プロセスを記録することを実現している．

①何について②どんな視点から設計を行うかを決定し，それについて③考慮すべき情報を参照しながら④設計する．

①，②により設計ビューのインデックス情報であるキー設計エンティティ，視点を，③により利用属性を，④により設計ビューの内容を記録する．以下，DIGを用いて設計プロセスを記録する具体例を示す(図4参照)．

設計の開始時点において，設計者は原要求ファイル(DIGでは，原要求をテキストファイルとしている)を開いて読みその概要を掴む．設計者は，プログラムの機能項目をまず整理しようと考えたとする．設計者は，原要求に記述されているプログラム名「在庫管理」をテキスト選択し，コマンド「設計進展」を起動する(①)．次に，設計者は，視点「機能項目」をシステムが提示したメニューにより指定する(②)．

事前に定義できる視点については，ソフトウェア開発現場単位で視点のガイドラインを設定することを想定している．これらの視点は，予めDIGに登録しておき，設計者はDIGが提示するメニューから選択する．また，それ以外の視点については，設計者が，視点を記録する際にDIGに入力する．

システムはインデックスとしてキー:「X店在庫管理」，視点:「機能項目」を設定した設計ビュー・エディタ・ウィンドウを開き(図4-a)，その他のウィンドウ(この場合は原要求)を閉じる．他のウィンドウをここで閉じるのは，ユーザに設計の参照行為を陽にさせるためである．次に設計者は，参照が必要な設計ビュー，原要求，領域知識を開き(例では原要求)参照しながら(③)，機能項目として設計エンティティ:「入庫処理」，「出庫処理」およびそれらの間のpart-of関係を設計ビュー用のエディタに記述する(④)．システムは，それらの設計エンティティ，関係の利用属性が「原要求」であることを記録する(図4-b)(現在DIGでは，領域知識のデータベースが未整備のため，領域知識からの利用関係は記録していない)．

5. ソフトウェアの影響波及解析支援

(1) 影響波及解析支援の考え方

保守活動は以下の4種類と言われている[22]．

① 外的要因(動作環境等)の変化への適合．

② 欠陥除去．

③ 機能向上．

④ 将来の保守に備えたソフトウェアの構造の洗練化．

これらの活動によるソフトウェアの変更要求は，以下の3種類に分類することができる．

変更1:原要求により要求されている設計対象の性質が変更される(③)．

変更2:領域知識に基づき設計された設計対象の性質が変更される(①, ④)．

変更3:問題点(バグ)の吸収による変更(②)．

前述の通り，本手法では，設計者が着目した設計対象の性質およびそれらの設計での利用内容を保守者が追跡可能とすることで影響波及解析を容易化する．追跡は，まず，1) 変更要求に関連した設計ビューを検索し，次に，2) 検索した設計ビューを用いて設計されている設計ビュー，設計生産物を追跡していく．

変更1による影響波及解析は，以下の方法で行うことができる．

1) 原要求を利用して設計された設計ビューのうち，変更された設計対象の性質に対応する視点，キー設計エンティティを持つものを限定する．次に，2) 限定した設計ビューを利用して設計されている他の設計ビュー，設計生産物を利用関係を用いて追跡する．

変更2による影響波及解析は，以下の方法で行うことができる．

1) 変更される領域知識と利用関係を持ち，更に変更される領域知識に関連した視点，キーを持つ設計ビューを限定する．次に，2) 限定した設計ビューを利用して設計されている他の設計ビュー，設計生産物を利用関係を用いて追跡する．

変更3による影響波及解析は，以下の方法で行うことができる．

1) 問題が発生した設計ビューまたは設計生産物を設計する際に利用した設計ビューを検索していき，問題の原因となっているものを特定する．次に，2) 特定した設計ビューを利用して設計されている他の設計ビュー，設計生産物を利用関係を用いて追跡する．

(2) 影響波及解析支援手法

影響波及解析支援手法について，DIGを用いた具体例を用いて説明する(図5).

今，在庫管理のプログラムにおいて，出庫依頼が，所定の用紙または電話で来る事になっていたのを，電子メールで来るように変更する例を考える(変更1に該当).

1) 変更に関連した設計ビューの検索

システムは，まず原要求を利用して設計された設計ビューの一覧を表示する(図5①). 保守者は変更に関連した設計ビューをその中から選ぶ(図5②). この選択において，設計ビューのインデックス情報である視点とキーが有効な働きをする．今回の変更は，媒体という性質が変ると考えられるので，保守者は表示されている設計ビューの一覧の中で，視点:「媒体」を持つものにまず着目する．次に，出庫依頼の媒体が変るので，キーとして「出庫依頼」を選ぶことにより，変更に関連した設計ビューを限定することができる(図6-a)，限定した設計ビューをDIGが表示すると(図5③)，その設計ビューの内容の中で原要求を利用して設計された箇所，本例では「電話または出庫依頼書(紙)」，がハイライトされる(図5では太線で表示). 保守者は内容を確認し，この設計ビューを修正の対象としてマークする(図5④).

2) 設計ビュー，設計生産物の追跡

システムは，先ほどマークした設計ビューを利用して設計された設計ビューのリストを表示する(図5⑤). 保守者は，リストにある設計ビューからチェックが必要なものを視点，キーを基に選ぶ(図5⑥). チェックの結果，修正が必要と判断した設計ビューをマークする(図5⑧). 同様に続けて行くと，影響を受ける設計生産物の箇所までたどり着くことができる(図5⑨).

6. 評価

筆者らは，ソフトウェアの開発，修正実験を行い，提案手法のソフトウェア修正工数上の効果および設計プロセスの記録による設計工数の増加率を評価した．実験は，要求分析工程について行い，以下のデータを測定した.

- ソフトウェアの修正工数
  - 従来手法 (設計生産物のみ用いる)
  - 提案手法 (DIGにより設計プロセスの記録を利用)
- 設計プロセスの記録工数

更に，ソフトウェア総開発コスト上の効果について試算した．本評価により，測定データは少ないが，本手法により期待できる効果のオーダーを明かにすることができた.

6.1 ソフトウェアの修正工数

あるソフトウェアを修正する工数について，設計生産物を利用した場合(従来手法)および設計プロセスの記録を利用した場合(提案手法)のそれぞれを測定した.

(1) 実験ソフトウェア

表1に示す.

(2) 従来手法による修正工数

表2に示す.

(3) 提案手法による修正工数

表3に示す.

6.2 設計プロセスの記録工数

被験者や対象ソフトウェアの影響を少なくするため，以下の方法により近似値を求めた．

設計プロセスを記録しながら設計する際の，設計ビューの編集に要した時間と，頭の中で考えた時間をそれぞれ測定した．以上をソフトウェア設計の任意の部分について測定し，設計ビュー1つ当りに対する平均値を算出した(表4)．

6.3 評価結果

(1) ソフトウェア修正工数に対する効果

表2, 3より，提案手法により修正に要する工数を15%以下に減らすことができる．

(2) 設計プロセスの記録による工数増加率

表4の結果を用いて求めた，表1のソフトウェアの設計に要する工数の概算値を表5に示す．

以上より，設計プロセスの記録による設計工数の増加率は約40%．

(3) 期待できるソフトウェア総開発コスト上の効果

得られた実験データから，ソフトウェア総開発費の約35%削減が期待できるという試算結果を得た(図7)．本試算は，保守工程におけるテストの工数が，保守工数の約28%を占める[12]，また，ソフトウェア全開発費の約70%が保守費用[13]という報告に基づいている．

(4) 分析

提案手法における修正工数が従来手法のに比べ大幅に少ないのは以下の理由によると考える．提案手法では，少数の設計ビューを見るだけで影響波及解析を行うことができる．見るべき設計ビューの数を絞り込むことができる理由についてまず分析する．これは，設計ビューのインデックス(特に視点)による絞り込みの効果が大きいためである．提案した影響波及解析支援手法では，ある設計ビューが変更された場合，その設計ビューを利用して設計された設計ビューの一覧を設計者に提示する．前述の評価実験では，全部で138個分の設計ビューの一覧が提示された．しかし，保守者は，この段階で，一覧に表示された視点名とキー名により，見る必要がない設計ビューを除外し，結局内容を参照した設計ビューは15個であった．次に，少数の設計ビューを見て内容が理解できる理由として，利用関係を辿って設計ビューを参照していくため，設計者が設計対象を抽象的で曖昧な像から次第に具体化していく過程に沿って，保守者が設計ビューを参照できることが考えられる．以上の評価実験から，設計ビューのインデックスと利用関係は，見るべき情報の絞り込みに有効であり，かつ利用関係は，見るべき情報を読む順序を誘導するのに役立つことが明かになった．

また，もう一つの理由が被験者の感想を分析して明かになった．それは，提案手法では修正作業の手順がシステムにより誘導されることの効果である．従来手法による修正作業は，修正の糸口を見つけるのに工数がかかっているのに対し，提案手法は，システムの誘導に従い段階的に作業を行うことができる．この効果は，特にスキルが低い保守者に対して期待できる．

設計プロセス記録の工数は，エディタ等の操作性の影響が大きい．DIGは，研究的関心が低い設計プロセスの編集機能や日本語入力機能などは最低限の機能しか有さない．これらの改善により，本評価値より少ない記録工数の実現が期待できる．また，本評価は従来手法の設計でのメモ作成を考慮していないため，実際の設計プロセスの記録による工数増加率は本試算値より小さいことが期待できる．

6.4 その他の定性的評価

設計プロセスの記録では，コスト以外に設計者の負担感が少ないことも必要である．提案手法は，検索で利用するインデックスや利用関係情報を付すための工程が不用なため(設計者の設計活動から獲得する)，他の手法に比べ記録の工数だけでなく，記録者が持つ負担感の点でも有利であることが期待できる．現在までの試用において，以下の点を除けばDNAを記述する負担はあまり感じないという利用者の感想を得ることができた．

- 視点名の決定，
- 単純な内容を持つ設計ビューの参照(頭の中に記憶しているため)．

特に後者は，利用属性の記録漏れを起こす可能性がある．更に，今回の試用では特に見られなかったが，エキスパート設計者によく起こると言われている思考の飛躍により，設計プロセスの記録漏れが予想される．以上の問題点は，運用面として，ソフトウェア開発現場毎の視点ガイドラインの整備および設計者の手法に対する理解を得ること，また，設計プロセスの質を安定化させる技術として，コンピュータによる視点のガイド機能および過去の設計プロセスの再利用支援機能の実現により解決が期待できる(付録3参照)．

7. おわりに

ソフトウェアの影響波及解析を容易化するための手法を提案した．

保守者にとって，ソフトウェアの修正箇所を限定する影響波及解析は，設計の根拠が残されていないため困難な作業になる．この問題点を解決するには，影響波及解析に有用な，設計の根拠の情報を小さいコストで記録，利用する方法を確立する必要がある．筆者らは，この方法を実現したのでここに提案する．

提案する手法では，設計者が設計プロセスの記録を作成しながらソフトウェアを設計する．この記録は，設計の根拠である，設計に影響を与えた要因の情報を含む．保守者は，本手法が実現する設計プロセスの記録の検索機能により，影響波及解析に有用な情報を参照することができる．また，設計プロセスの記録は，設計の思考の流れに沿って作成するため，記録のコストが小さいことが期待できる．

以上の手法を実現したプロトタイプシステムDIGを試作し，試用･評価を行った．その結果，設計工数が設計プロセスの記録により約40%増えるのに対し，ソフトウェアの修正工数を従来の15%以下にできる見通しを得た．また，設計プロセスの記録の内部に付けられた索引と関連付けの情報が，影響波及解析に設計プロセスを利用する上で有効であることを確認した．但し，設計プロセスの記録では，設計者の頭に記憶されている情報について，利用関係が記録しにくい等の問題点があることがわかった．今後は，この問題点を解決するため，過去の設計プロセスを再利用した設計プロセスの記録コストの低減および記録情報の質を安定化させる手法を検討する．

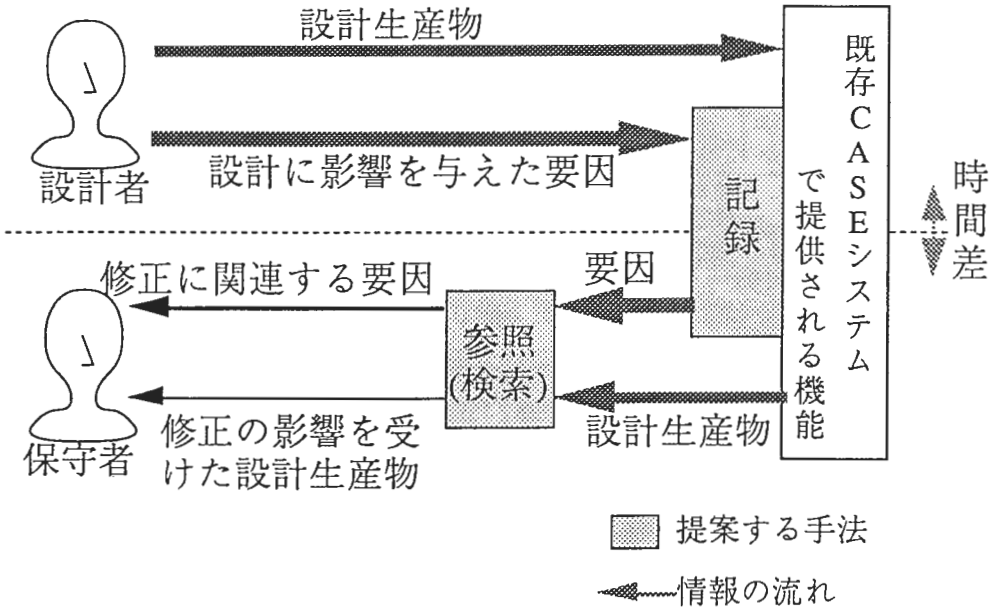


図1 提案する手法の位置付け

Fig.1 Location of The Proposed Method

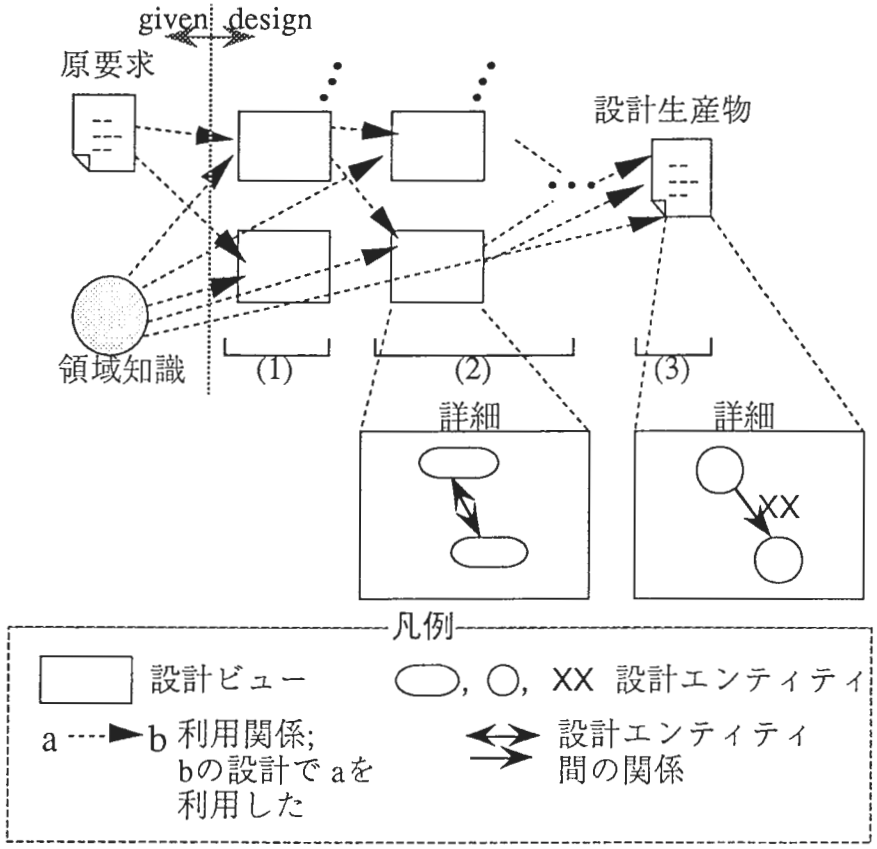


図2 設計プロセスモデル

Fig.2 Design Process Model

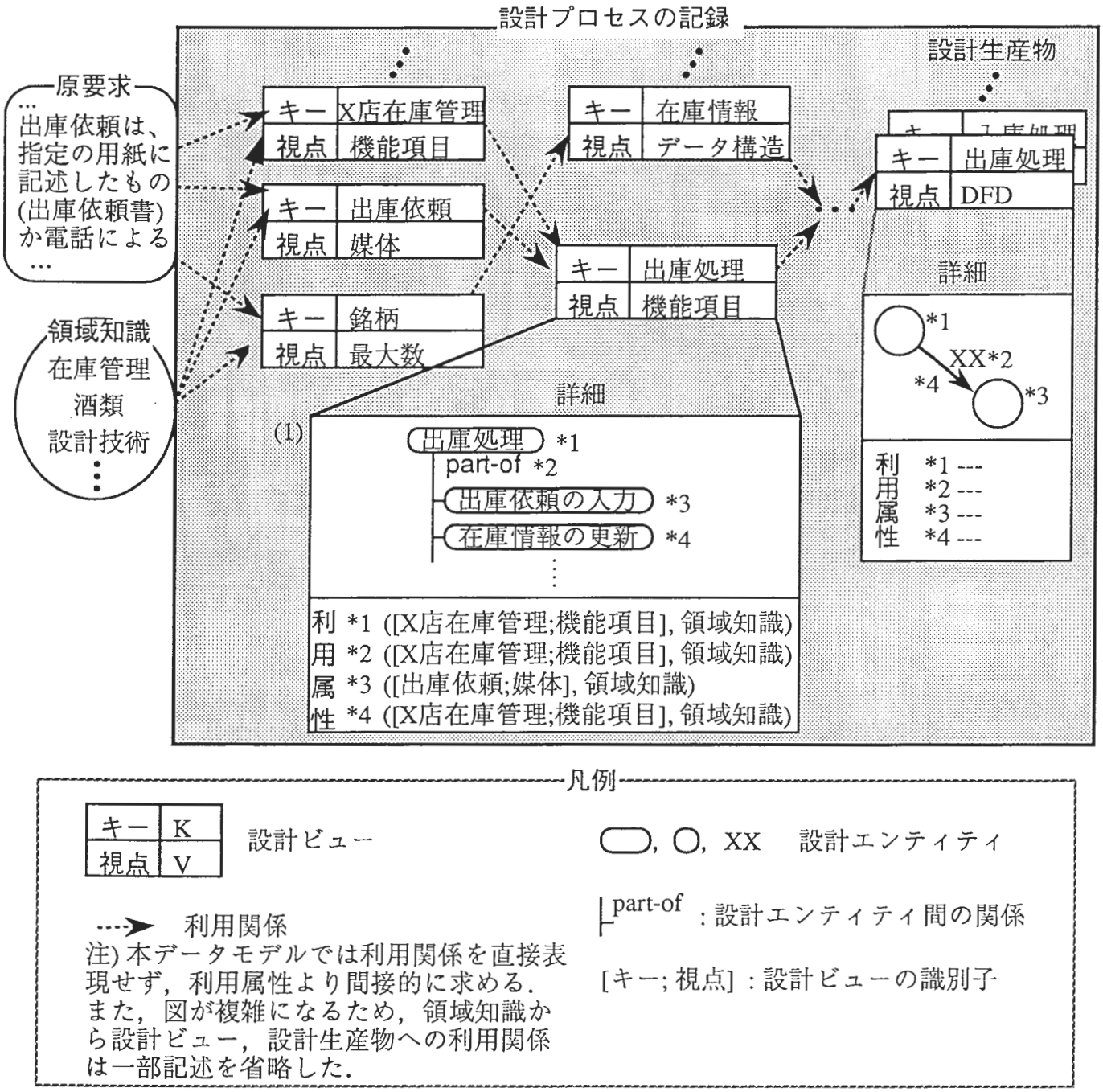


図3 設計プロセスの記録例

Fig.3 Design Process Record Example

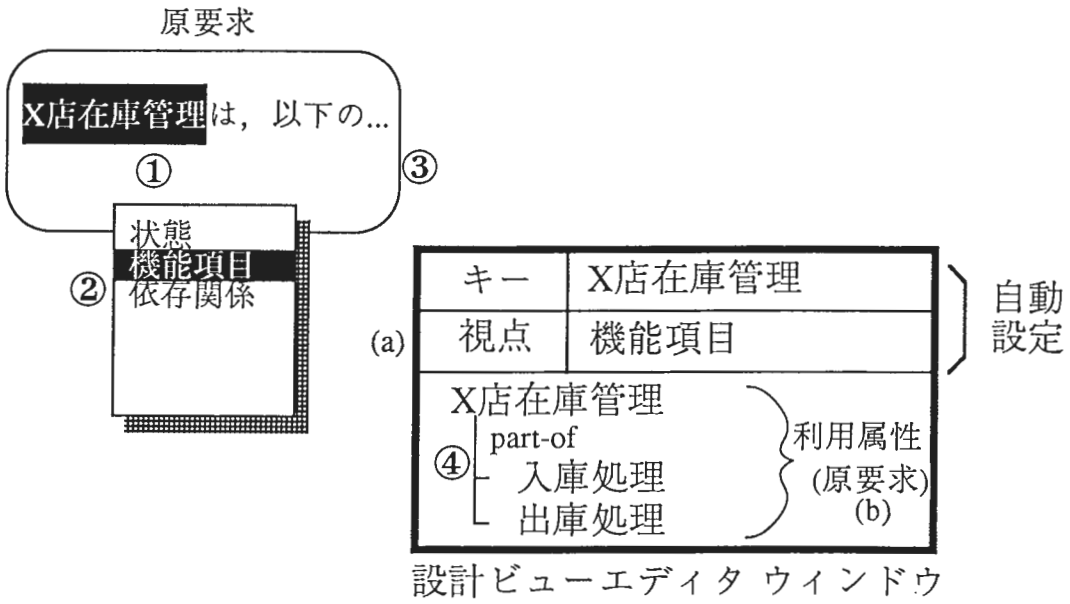


図4 設計プロセスの記録法

Fig.4 Recording Design Process

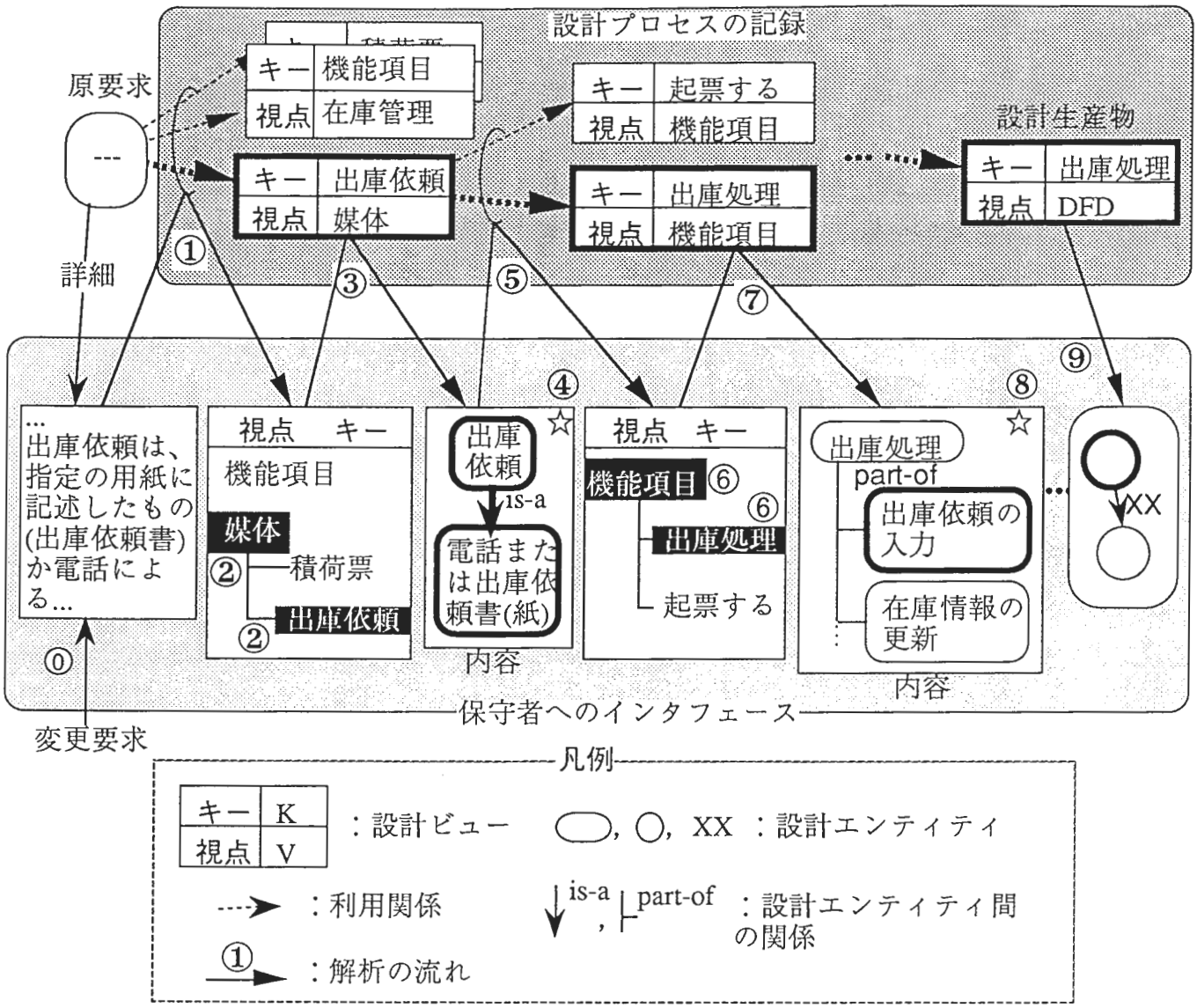
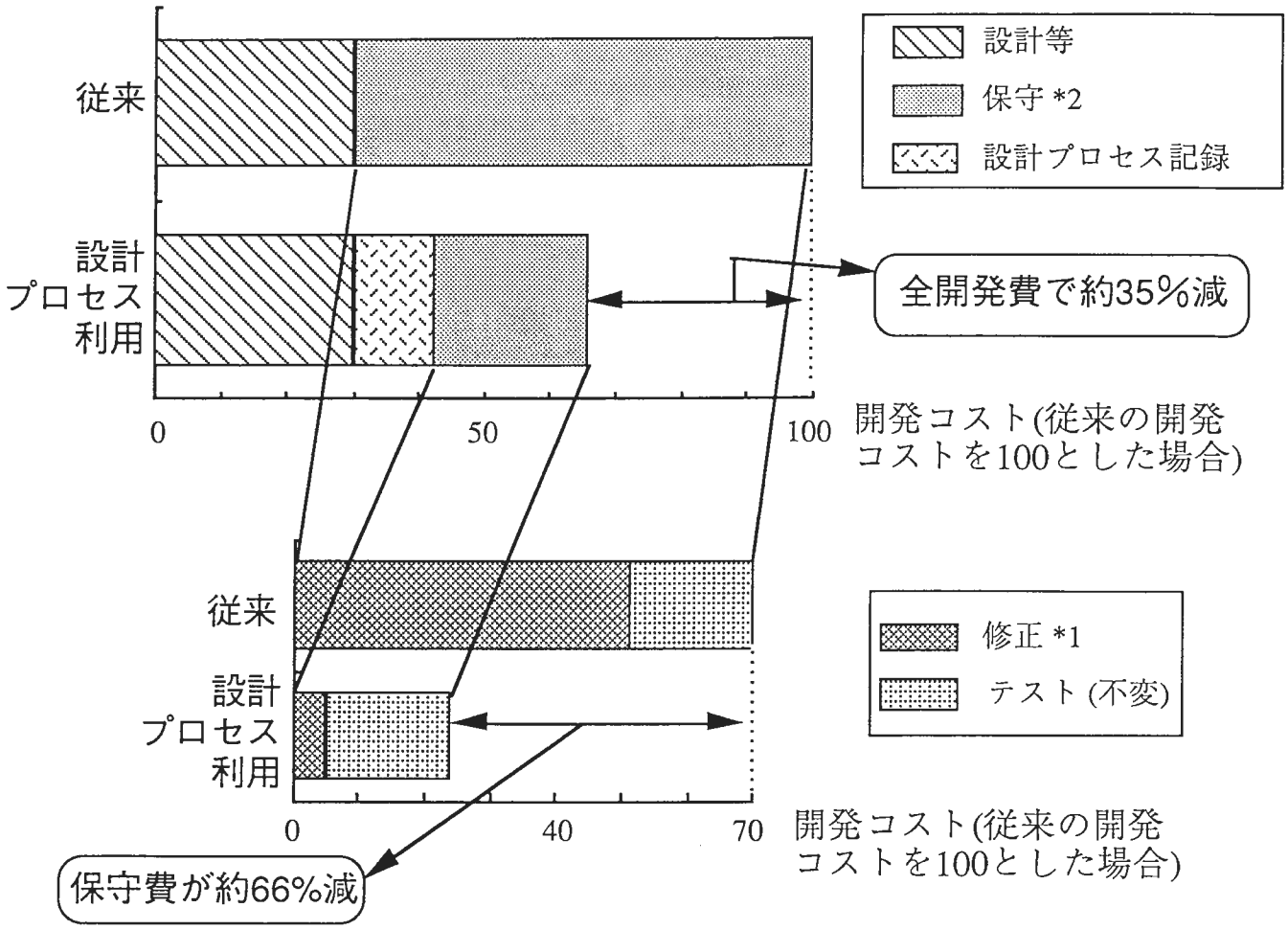


図5 影響波及解析の支援

Fig.5 Change Propagation Analysis Support



試算根拠

・コストは工数に比例する
・要求分析フェーズのみ効果を測定，他の工程でも同じ割合で効果が得られるとして試算

*1: 保守工程における修正:テストは72: 28 (1979年IBM社による調査)
*2: 保守費は開発費の70% (1977年米国での調査)

図6　期待できる効果

Fig.6 Estimated Cost Reduction Made by the Proposed Method

表1　修正実験で用いた例題ソフトウェア

種別	設計生産物と規模	設計プロセス規模	修正の規模
事務処理プログラム	データフローダイヤグラム(DFD)．バブル(プロセス)数約100	①設計ビュー数301 (内②設計生産物用30)	設計生産物上で修正した設計エンティティ数10

表2　従来手法によるソフトウェア修正の工数

被験者(情報処理経験年数)	A(10)	B(2)	C(13)	D(3)
修正時間(分)	240	480	240	260

(注1) 被験者は修正する内容および修正するソフトウェアに関して事前に知識を持たない．
(注2) 修正とは，被修正ソフトウェア理解，修正要求理解，修正箇所限定，修正内容の決定，設計生産物の編集を指し，テストは含んでいない．

表3　DIGによるソフトウェア修正の工数

被験者(情報処理経験年数)	A(10)	E(7)
修正時間(分)	21	35

(注1) 被験者Aは，従来手法による修正を行った後，DIGを使用した実験を行った．
(注2) ここでの修正とは，従来手法における修正に加え，記録した設計プロセスの修正箇所限定，修正内容の決定および編集まで行う．

表4　設計ビューの記録／設計工数

項目	測定値1	測定値2	測定値3	設計ビュー当りの平均
被験者(情報処理経験年数)	F(10)	F(10)	B(3)	
測定設計ビュー数	15	7	10	
設計ビューの編集時間 (sec)	1051	381	542	③ 61.7
思考時間 (sec)	1505	536	558	④ 81.2

(注) 測定値1-3は，すべて修正実験で用いたソフトウェアとは異なるソフトウェアについて測定.

表5　設計プロセスの記録工数

項目	測定，計算値
⑤ 設計生産物の記述に要した時間	17385 sec
⑥ 設計ビューを編集するのに要した時間	①271個×③61.7 sec = 16,721 sec
⑦ 従来手法の設計時間 (頭で設計＋設計生産物を作成)	①301個×④81.2 sec＋⑤17385 sec = 41,826 sec
設計プロセス記録による工数増加率	⑥16,721 sec／⑦41,826 sec =40%

関連する研究

1.1 プロセスプログラミング

プロセスプログラミングは，Osterweilによって提唱されたとされている[1)]．その後注目を浴び，さまざまな研究が行われるようになった．

当初の目的は，ソフトウェア開発プロセスの最適化にある．即ち，設計プロセスを正確に記述することにより，プロセスのコントロール，評価を可能にするというのである．しかし，記述形式は提案はされたが，これらの目的を果たす技術に関する提案はほとんど行われておらず，プロセスプログラミングの研究も下火に成りつつある．その理由として，現在提案されている手法のほとんどは，ソフトウェア設計法で示されている，設計手順のアウトラインを記述の対象にしている．この程度の具体性の情報では，上記目的を達成するために利用できない．また，これより細かい設計手順は，設計するソフトウェアの持つ性格に左右されるため，一般的な記述ができない．このようなジレンマが研究の壁と成っていると考えられる．また，阪大の鳥居先生などの研究室で，プロセス記述に基づき設計者を誘導する仕組などが考えられている[25)]．しかし，上記問題があるため，設計者を設計法のアウトラインについてガイドするにとどまってしまう．

役に立つ設計支援等を行うには，設計法レベルの情報とは別に，問題領域固有の設計ノウハウに関する情報も必要になる．我々の研究は，この情報を扱う方法について研究している．

我々の研究は，ソフトウェアの保守支援である．従って，設計者が誰でも知っているソフトウェア技術に共通する情報よりも，そのソフトウェア固有の特徴に関する情報が重要になる．DNA (Design Dependency Architecture, 我々が提案している設計プロセス記録のデータ構造の名前)は，設計法で明示されている一般的な設計手順に関する情報ではなく，個々のソフトウェアが持つ特徴を記録することで，ソフトウェアの保守を容易化することが目的である．ソフトウェアが持つ特徴とは何かを定義することは難しいが，幸い設計者はソフトウェアを設計できるのであるから，設計の各局面で重要な特徴を認識しながら設計を行っているはずである．それを記録する手法がDNAである．しかし，逆に，DNAの内容は，その設計で用いる設計法により異なってくる．即ち，設計法に依存している．最終的には，設計法依存部と非依存部に分けることが望ましい．現在，全設計プロセスのうち，設計法で与えられる設計の部分を，予めルール形式(視点利用規則)で記述しておき，設計法では決らない，設計の対象などに依存している設計の部分を，過去の開発事例を再利用することにより設計や保守の支援を行う方法を検討している(付録4参照)．前者の部分がプロセスプログラミングの技術になる．

1.2 Design Rationale，設計支援技術

最近とくに盛んに成りつつある研究である．まだ，技術が若いので，定義はないが，基本的には，「設計の根拠の記録手法およびその利用技術の確立」と考えられる．

まず，これらの技術の背景について述べる．

ソフトウェアの設計法というのが存在する．例えば，構造化分析手法，トップダウン設計法，ジャクソン法などさまざまな方法が提案されている．ほとんどすべての設計法に共通することは，何れも1つのソフトウェアを作りっきりにするための技術ということである．即ち，上流工程の出力(設計生産物)は，設計結果(下流工程で作るソフトウェアが満たすべき処理内容)だけを記述していて，その根拠に

ついては下流工程に渡さなくてよい，即ち記録しなくてよいことになっている．これは，ウォーターフォールモデルで一方向的に開発をする場合はあまり問題はないが，ソフトウェアを保守したり，再利用する場合などに問題がある．例えば，ソートを行うモジュール1つをとっても，そのモジュールがそのように設計された背景には，予想されるデータ数，データ構造，要求されるスピードとメモリ効率，開発工数などの要因が存在したはずである．そのモジュールがこれらの要因をどう考慮したものかを知らなければ，違う要因の条件を持つソフトウェアで利用することはできない．

ソフトウェアの設計という問題があまりにも複雑で難しいため，前提を設けて(ウォーターフォールモデルで一方向的な開発をする，また，保守，改造は別の問題である，etc.)，単純な開発モデルの上での設計法論を展開してきた結果，いつの間にか，前提が忘れられ，この単純な開発モデル上での設計法が一人歩きしてしまったと考えられる．

設計法だけではソフトウェア開発に関わる問題が根本的に解決できないと分かると，新しい設計支援のパラダイムが提案されてきた．その1つが再利用技術である．再利用は，設計生産物を再利用することが目的である．設計法上では，設計生産物には，設計の結果しか記述されないため，再利用技術は，設計生産物に情報を付加するアプローチが中心であった．設計支援技術も同様である．設計生産物にない設計の根拠を，予めコンピュータ化しておくアプローチが採られた．プログラムジェネレータは，ソフトウェア設計の根拠を，ジェネレータのアルゴリズムとして記述する方法である．また，知識ベースを用いたソフトウェア設計支援技術などは，設計の根拠を生成することができる論理を予めデータベースとして用意しておくアプローチである．しかし，設計の根拠を汎用的なアルゴリズムで記述したり，設計の根拠の候補を予め用意，記述するのは難しく，これらの手法は限られた範囲でしか適用できない．

予め用意できないのであれば，設計時に残せばよいという考え方が，design rationaleの考え方である．我々の研究もこの考え方によっている．

Design rationaleの代表的な技術として，IBIS (Issue Based Information System)がある．この存在が大きく，他の研究の多くは，これと同じ視点からのみ行われている感が有る．

IBISは，もともと，議論の履歴を保存するための技術である．ある問題を解決するために行った議論をある書式で保存しておくと，後で便利であるというもので，現在は，そのグラフィカルなインタフェースを持ったシステムであるgIBISがこれらの研究のreference的存在となっている(代表的なのは次の文献に掲載されている，Human-computer Interaction, Vol.6, No.3&4, 1991)．IBISは，図1.1に示すような構造で，議論または問題解決のプロセスを記録することを提案したものである．

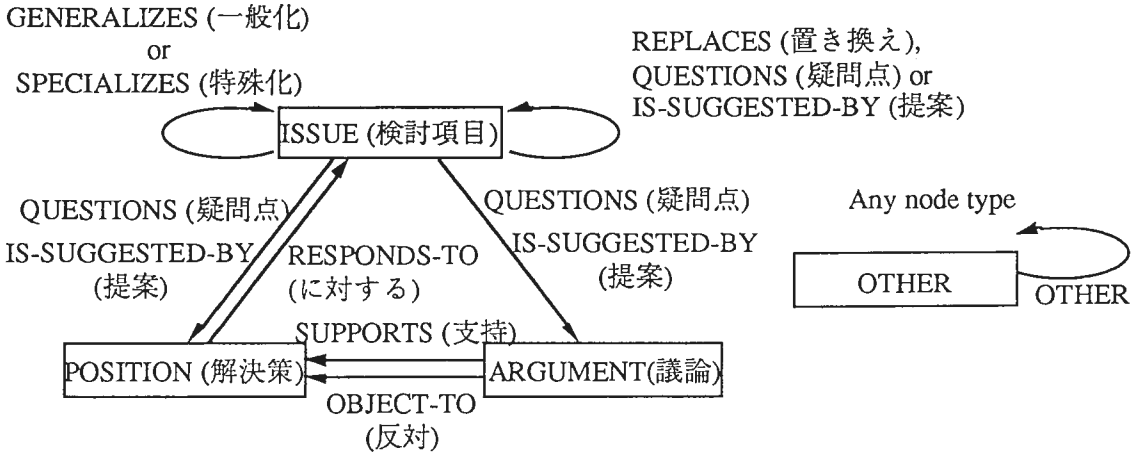


図1.1 IBISのノーテーション

我々の手法と比較して重要な違いは，IBISやその影響を受けた多くのdesign rationale技術は，議論のプロセス(議論で出た意見の位置付け)を記述する，という点にある(ソフトウェア設計に対しても適用できる)．これは，記録したdesign rationaleの再利用を損なう以下の特徴を持つ．

- 記録する情報が，その時の問題の解き方，議論の進め方に大きく依存している．
- 記録される情報に文脈依存性が有る．即ち，ある議論の後半で浮び上がった課題(issue)は，それより前にどのような議論を展開してきて，それがどのような結果を導いてきたかに依存している．しかも，どれに依存しているかがdesign rationaleのデータ構造からは分からないため，人間が記録の頭から読んで理解しなければならない．
- 解いている問題の構造が見えない．記録した情報は，議論の流れにより構造化されているため，別の問題で再利用しようとした場合，解く問題や設計対象が似ているのかどうか，どこが違うのかなどを判断するには，記録を頭から読み人間が理解し，判断する必要がある．これらの記録は一般に膨大な量になるため，これはかなりの負担になり，かつ工数もかかる．言い替えれば，問題(設計)の解き方，内容や議論の進め方，内容は，解くべき問題(ソフトウェア設計の場合は設計対象)が持つ構造(特徴)により決ってくる．その因果の果側の情報だけを見て再利用することはできない．
- 設計生産物とdesign rationaleの対応がわかりにくい(これはその後改善された)．

記録したdesign rationaleの再利用には，問題解決のプロセスの一致性，即ち，問題解決の手順の一致性は本質的な問題ではない．解くべき問題が持つ特徴の一致性が再利用する上で重要になる．

DNAは，設計しているソフトウェアやその機能の特徴がどうなっているのかという情報を記録しているため，過去に設計したソフトウェアとの特徴の比較が可能になる．従って，設計対象が持つ特徴の設計での活かし方，即ちdesign rationaleの再利用が可能になる．また，解くべき問題の性質を記述しているため，基本的には文脈依存性は存在しない．

但し，DNAは，設計技術に関する知識をある程度持っている人にしか理解できない．即ち，設計技術的な情報の全部はDNAに含まれていない．

1.3 Reverse Engineering

リバースエンジニアリングは当初，エンジニアリング的なノウハウを設計生産物から求めるというニュアンスがあったように記憶する．しかし，ソフトウェア工学に限って言えば，現在行われている研究のほとんどは「下流工程の設計生産物から上流工程の設計生産物を生成する」ことである．即ち，上流工程で設計生産物を作るのをさぼって楽をするための技術になってしまっている．これは，短い開発期間に追われて，とにかく早くソフトウェアを作ってしまわなければならない．しかし，後でメンテナンスの需要が出て困ったことになる，という現状をよく反映した結果であると言える．結局，下流の設計生産物を見やすくするために，抽象化したり情報を整理して表示したり，グラフィカルなインタフェースを提供したりという技術に落ち着いてしまう．設計の工程を逆にさかのぼるとは，上流工程の設計生産物へたどり着くこと以上のことがあるはずである．即ち，その工程の間で行われている設計という行為で用いたノウハウ的な情報をrevealしなければ，本当に設計工程を逆にさかのぼったことにならない．

我々が狙っているところは，もっと本質的な情報，即ちノウハウ的な情報を蓄積，再利用することであり，既存のリバースエンジニアリング技術とは扱う領域が違う．

もし，リバースエンジニアリングで本質的な情報を生成しようとすれば，設計の根拠の情報を予め

コンピュータに入れておく必要が生じる．これは前述のように，困難な技術である．では，予めコンピュータに入れられないなら，設計の根拠を設計時に記録する方法が考えられる．しかし，記録時に残すのであればリバースする必要はない．以上を詰めて行くと，過去に蓄積した設計の根拠の情報を後の設計で利用する，即ち，我々が行っている設計ガイド機能に落ち着く．

これは，設計者の設計をシステムが理解する(別の設計事例を基に)という設計機能にも発展できる．

設計が行っている設計を理解して，初めて高度な支援が可能になる．そのための基本技術の確立である．このように，リバースエンジニアリングは，突き詰めて行くと，我々の研究へ組み込まれて行く可能性がある．

1.4 設計支援に関する認知的考察

問題領域におけるノウハウとは何かを解明する場合、問題領域のエキスパートと非エキスパートの設計活動を比較分析してみる方法がある。

Adelson[24]らは、ある特定の問題領域のソフトウェアの設計活動について、その問題領域の知識を持つ設計者(エキスパートと呼ぶ)と持たない設計者(非エキスパートと呼ぶ)を比較している。その報告の中で、非エキスパートはエキスパートと比べ、設計対象の(ふるまいに関する)シミュレーションが出来なかったことを述べている。シミュレーションは、設計者が持つ設計対象のメンタルモデルを用いて行われる。そのメンタルモデルは、設計者が設計対象に関連した制約を認識することにより構築されることから、設計対象が持つ制約を設計者に提示する設計支援機能などの必要性を述べている。

以上から考えると、問題領域の設計ノウハウは、設計対象が持つ、設計上考慮すべき制約の認識技術と関連が深いと考えられる。例えば、エディタの2つの機能の間の制約として、現在メモリ上にある文書の内容が未保存の場合、プログラム終了機能実行前に、ファイルセーブ機能を実行する等が考えられる。この制約は、エディタプログラムの機能構成を設計する場合などに考慮しなければならない。エディタという問題領域に馴染みのないエンジニアはこのような制約を認識する能力が弱いというのである。本手法は、設計対象が持つ、設計上考慮すべき制約を認識する能力が問題領域の設計ノウハウと関連が深い重要な情報と考え、その情報をコンピュータに蓄積し、活用する方法であると言うことができる。

(1) 論理的基盤がある問題領域

論理的基盤がある機械などの設計を支援する場合、定性推論の技術が有効である[17]。これは、定性推論技術により、設計の対象の振舞いを計算機でシミュレートすることが可能となるためである。しかし、現実的には、このように論理的基盤がある問題領域は限られている。

(2) 論理的基盤が弱い問題領域

論理的基盤が弱い問題領域では、SEが持つ設計ノウハウに頼っているため、SE不足や品質のばらつき等の問題を抱えている。支援技術の確立が望まれる領域である。その支援技術としてエキスパートシステムがある。従来型のエキスパートシステム構築技術は、人間のエキスパートが持つノウハウを人間がインタビューし、ルール型知識等にコーディングするといったものである。これは、知識(ノウハウ)獲得の難しさ、学習機能の欠如、新しい問題への適応能力の低さ等の問題点[33]によりごく限られた問題にしか適用できない。そのような問題点を解決するための技術として、CBR(Case Based Reasoning, 事例推論)技術が注目･研究されている[31)33)38)]。これは、人間の経験を利用した問題解決能力や法律等の分野で判例が重視されている事などから、期待されている技術である。

本研究では、エキスパートの設計の事例である設計プロセスを見本としてコンピュータに格納し、他の設計者が別の設計において参照し、そこで利用されている設計ノウハウを理解、活用する方法を検討する。

我々は，ソフトウェアの設計やメンテナンス中に起こる要求や仕様の修正による影響波及の解析に有効な情報を設計プロセスより獲得する方法を研究している。要求分析は、設計対象の持つ諸性質を設計する作業であると考えられる。設計対象の性質間には制約が存在するため､本作業はその制約を認識･充足していく作業と言うことができる。この研究は、設計プロセスより得た、ある設計対象の性質を設計する場合、他のどんな設計対象の性質を考慮したかという情報が、影響波及解析において有効となることを示している。前述のことから、この情報は問題領域の設計ノウハウと大きな関係があると言える。従って、ソフトウェアを設計する場合にも有効な情報となることが期待できる。このように設計対象の性質の設計は，設計者の行動に影響を与える．本手法は，設計者の行動から，その背後にある設計対象の性質が設計に与える影響についてデータベース化する．これは，ドメインモデルを生成する技術の研究でもある．

1.5 定性推論のOntologyとの関係

我々の研究で用いている「視点」は、定性推論の研究で用いられているontology[6]やperspective[7]に類似している。これらの研究は、物理システムを、異なるontology、例えば、device ontologyとcharge-carrier ontology、を使い分けることによりその振舞いを推論するというものである。我々の手法は、設計活動というものを、ソフトウェア設計における種々の抽象レベル設計で行われる異なったontologyの切り替え動作という観点から記録する為に"視点"を用いるものとも考えられる。但し，我々の手法では，人間の認知能力により，設計者が一度に認識できるのは，単一のontologyのさらにその中の限られた範囲としている (= 設計ビュー).

1.6 モデル推論との関係

モデル推論技術のソフトウェア設計支援に対する適用を一言で言えば，設計に対する入力と出力(即ち設計結果)の事例から，設計のロジック(ここでは広い意味でのロジック)を推測する方法の確立となる．もしこの技術が確立されれば，非常に有効な手段となる．しかし，モデル推論技術は，まだ未熟であると言わざるを得ない．

モデル推論技術を用いて設計のロジックを求める研究は，静岡大学の落水教授(現在北陸先端技術大学)，山口氏[32]などが行っているが，なかなか有効な結果が得られないという自己評価であった．

モデル推論の技術動向などを注意し，将来的にはその研究成果などを取入れていくことが望ましい．

1.7 ドキュメンテーション技術の動向

(付録2)

付録2

0

ソフトウェア·ドキュメンテーション研究動向の紹介

ATR 通信システム研究所 通信ソフトウェア研究室

浜田 雅樹

e-mail : hamada@atr-sw.atr.co.jp

1

ソフトウェア·ドキュメンテーション研究動向の紹介

最近の話題	本日の話題
高度な 判断 項目の ドキュメント化	1. 高度な 設計 判断の 記録を 目指して
設計プロセスの記録	判断履歴の 記録と 利用
一貫性 保持・影響範囲解析手法の 確立	2. 異なるドキュメント間 における 一貫性 保持の ための 研究
ドキュメント間の 関係の 管理	
レポジトリに よる 統合化、集中管理	3. レポジトリ
グループ、 分散開発環境の整備	レポジトリを 核とした 開発環境 (概要)
上流工程の ドキュメント 法	(関連技術) 4. ハイパーテキストによる ドキュメンテーション
分析·設計手法 (構造化分析、オブジェクト指向システム分析、JSD)と 供に普及	コードと ドキュメントの 一体管理 他

1. 高度な 設計 判断の 記録を 目指して

設計法や作業標準等で決められている正規生産物に記録できない、設計上の高度な判断事項を残す。

❑ 設計プロセスの記録

設計プロセスを記録することで設計上の判断事項を捉える。

紹介する研究

WEB
IBIS
Pottsのアプローチ
SoftDA
PPK

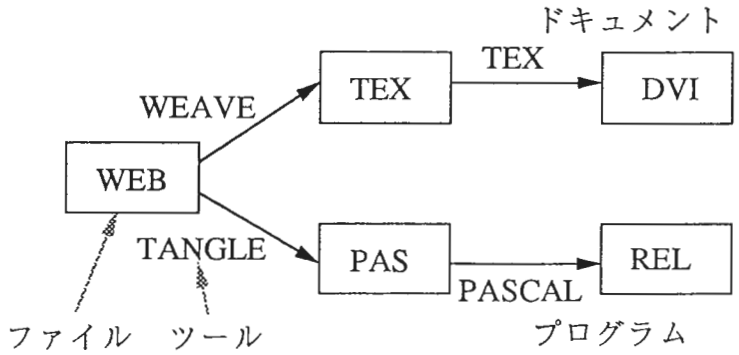
❑ プロセスプログラミング

設計作業を定式化する。生産物間の関係が明確に定義される点が重要。
紹介は省略。

WEB システム

--- 文芸的プログラミング (Literate Programming) ---

Donald E.Knuth, 1984



❑ 設計時の思考を記録。

❑ 奇麗な印刷をドキュメント書きの動機とする。

❑ 自由な設計法·順序

❑ 強力なマクロ

WEB : 設計において思考した事柄を記録 + 設計結果であるソースコードをプログラミング言語で記述

PAS : 高級言語(PASCAL) で記述されたプログラム

TEX : 設計において思考した事柄を清書用言語と供に出力

[WEB 84]

WEB の例

1. 素数の出力 --- WEBの実例　以下のプログラムは本質的にはダイクストラの
...
<最初の1000個の素数を印刷するプログラム 2>

2. このプログラムには入力がない。プログラムを単純化するためである。このプログラムの結果は最初の1000個の素数の表を作ることであり、この表は出力ファイルに置かれる。
入力がないので、m=1000という値をコンパイル時定数として定義しておく。プログラムは任意の正の整数mに対して、計算の容量限界を越えない範囲で、最初のm個の素数を生成する能力がある。
...
<最初の1000個の素数を印刷するプログラム 2>≡
```
program print_primes(output);
  const m=1000;
    <プログラムの他の定数 5>
   var <プログラムの変数 4>
     begin <最初のm個の素数を印刷する 3>
     end.
```
このコードは第1節で用いられる。

3. プログラムの立案　... 素数全部を保持する為の表を設けることとする。そうすれば、生成問題と印刷問題を分離することができるからである。 ...

4. 表pをどのようにして表すべきか？2つの可性がある。...
<プログラムの変数 4>≡
```
  p: array [1..m] of integer; {最初のm個の素数, 昇順で}
```
...

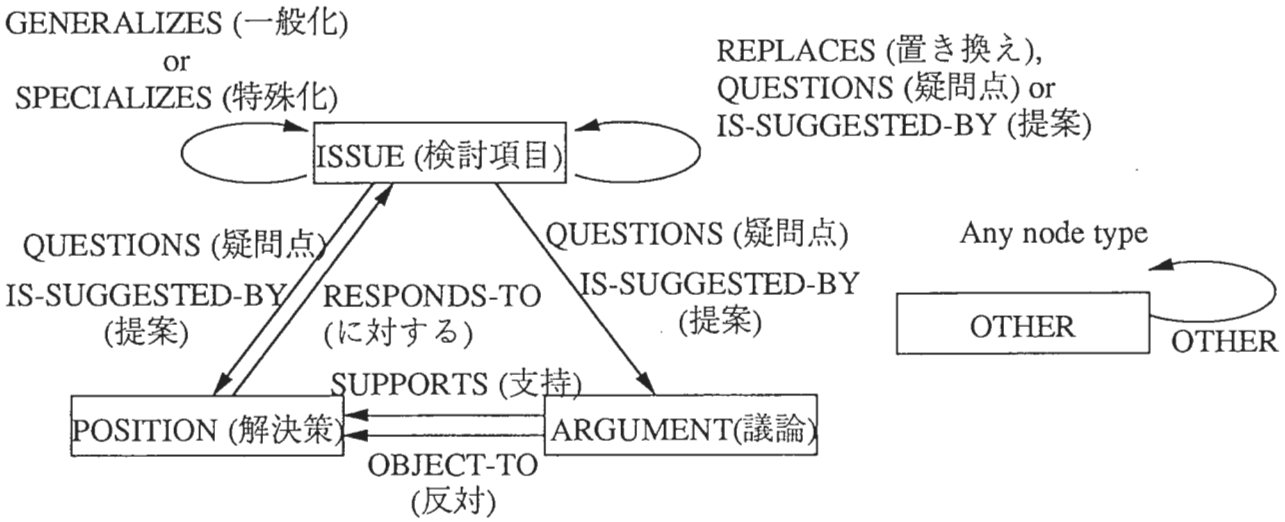
注) 清書コマンドや清書結果は省略してある。

[WEB '84]

IBIS (Issue Based Information Systems)

Horst Rittel, 1970

問題解決プロセスの記録法

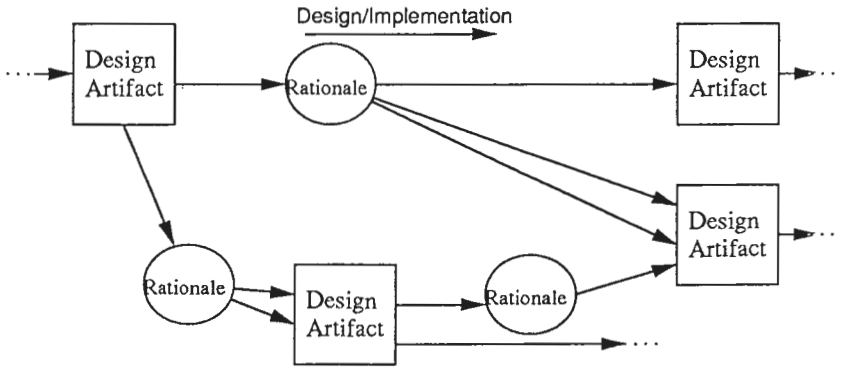


gIBIS : IBIS用 グラフィカルインタフェースツール
Jeff Conklin and Michael L.Begeman (MCC), 1988

[IBIS '70]

Pottsの設計判断記録法

Colin Potts and Glenn Bruns (MCC Software Technology Program), 1988

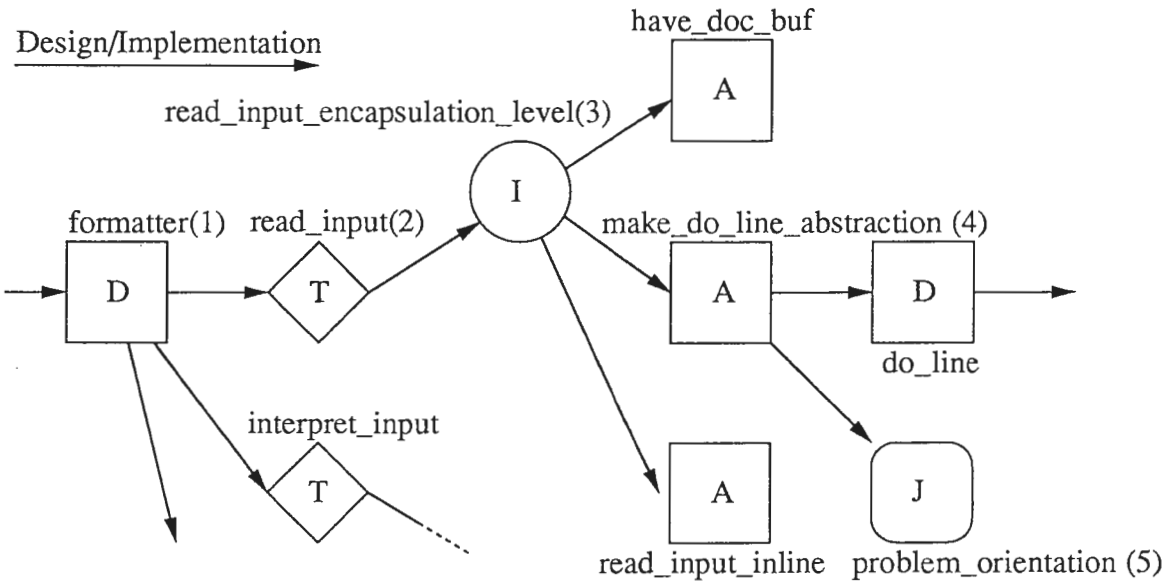


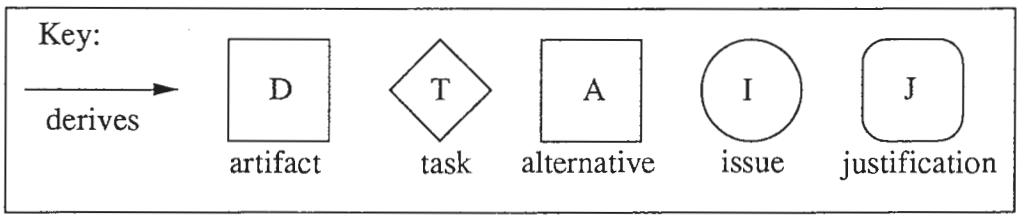
設計活動の意味モデル

[Potts'88]

Pottsの設計判断記録法

--- Liscov and Guttag's 設計法によるtext formatterの設計例 (ネットワーク) ---





[Potts'88]

```
formatter = PROC (ins, outs, errs : stream) SIGNALS (badarg(string))
MODIFIES:
  int, outs, errs
EFFECTS:
  もし、insが読み込みモードで開いていない場合、また、...
  ... ins, outs, errs は リターン前に閉じる。
TASKS:
  read_input
  interpret_input
  procedure_output
EFFICIENCY:
  nをinsにある文字数とすると、計算時間はO(n)、出力(outs)への
  出力規模はO(n)、作業エリアはnよりかなり小さい。
```

(1) formatterのprocedure仕様

```
read_input_encapsulation_level = ISSUE
CONCERNING:
  read_input
SUMMARY:
  read_inputはどのようにカプセル化したら
  良いか
ALTERNATIVES:
  read_input_inline
  have_doc_buffer
  make_do_line_abstraction
```

(3) issue ノード例

```
read_input = TASK
DONE-BY:
  formatter
SUMMARY:
  formatterが入力を読み込む。
```

(2) task ノード例

```
make_do_line_abstraction = ALTERNATIVE
SELECTED
SUMMARY:
  1行単位で読み込むdo_lineという手続きを作りカプセル化する
RESPONSE-TO:
  read_input_encapsulation_level
ARTIFACT:
  do_line
```

(4) design alternative ノード例

```
problem_orientation = JUSTIFICATION
CONCERNING:
  make_do_line_abstraction
SUMMARY:
  1. format処理は行単位で行われる。
  2. 一度にドキュメント全体をバッファに読み
  込むのは、作業エリアの効率上好ましくない。
```

(5) justification ノード例

Pottsの設計判断記録法
--- Liscov and Guttag's 設計法によるtext formatterの設計例 (ノードの記述例) ---

[Potts'88]

SoftDA

山本, 磯田 他(NTT ソフトウェア研究所), 1986

(1) 設計情報モデル

R/D ネットワーク --- コードと設計情報を一体化しネットワーク構造で管理、設計過程で記述

R (要求) パート

○ コマンド名とパラメータの区切り記号は'_' である.
○ パラメータの区切り記号は',' である.
○ 入力文字列の終わりは NULL文字で表す.
...

◆ 詳細化の階層を階層テキストで記述
◆ 関連のあるノードをリンク情報で記述
◆ ユーザが情報をすべて入力

↓利用

☞ 影響範囲解析
☞ 再利用支援 (後述)

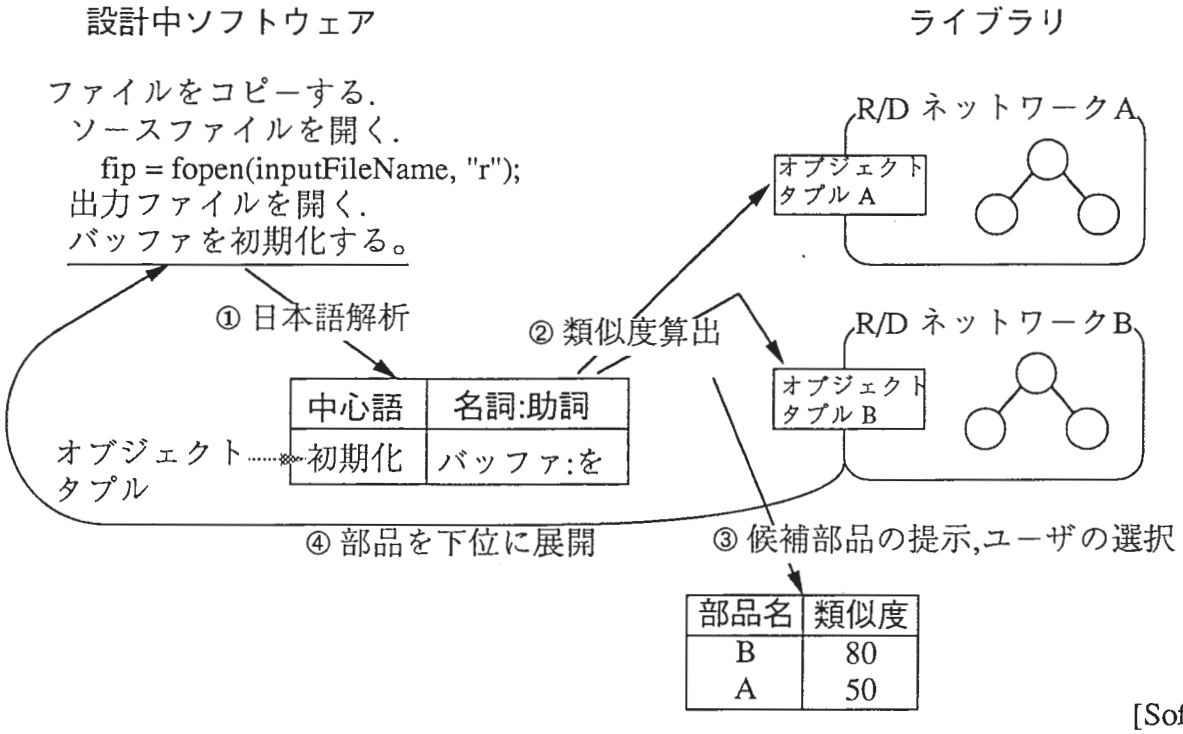
D (設計) パート

エディタ
  テキストファイル
  編集コマンド
    編集コマンド名
      char cmd = ' ';
    区切り記号は '_'
      const char ch = '_';
    ...
    入力文字列の終端まで繰り返す.
    区切り記号だったら ... する.
      if (readCh == ch) ...

[SoftDA '86]

SoftDA

(2) 再利用支援機能

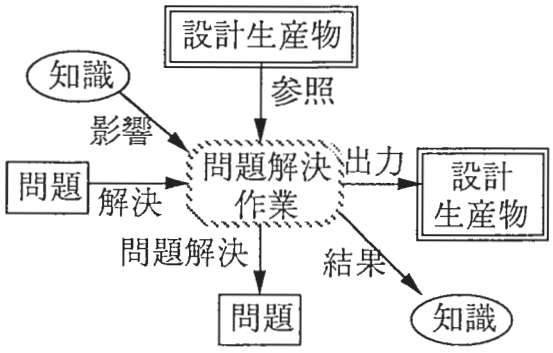


[SoftDA '86]

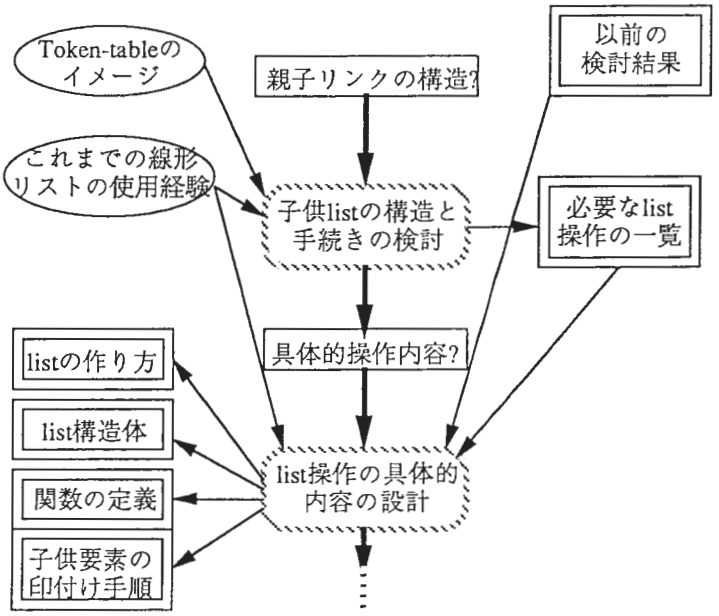
PPK (Problem-Product-Knowledge model)

中島、田村、上原他(三菱電機 情報電子研究所)、1989

目的 --- 設計プロセスを記録し分析、設計支援法を確立する。



PPK モデルの概念図



PPK モデルによる設計プロセス記述例 [PPK '89]

PPKモデルに基づく設計プロセス支援ツール

設計プロセスを記録する (設計支援が目的、HyperCardで試作)

主な機能

- ❑ それまでの設計で得た知識の参照
- ❑ 前提となる問題ノード参照
- ❑ 設計生産物参照
- ❑ 新しい設計生産物、問題、知識を作成する機能
- ❑ 設計における分岐状況の表示、選択

利点

○ 1つの問題を掘り下げて検討するのに適している。
○ 代替案等を違和感なく記録できた。

問題点

○ 情報分割が不十分な記録
○ 検討対象の遷移による混乱が発生
○ 記録されない設計プロセスが存在

[PPK '89]

2. 異なるドキュメント間 における 一貫性 保持の ための 研究

問題点 : 設計生産物間の非直行性 →
- ○ 修正による影響吸収作業の複雑化
- ○ 複数設計者間での矛盾
- ○ 重複作業

軽減化 ↓

- ○ 複数のメンバによる設計における修正支援
- ○ 重複部分の流用による設計作業の効率化

主なアプローチ

- ❑ スキーマレベルでの関係記述

  SoftDA/DD

- ❑ Hypertext 技術等によるインスタンスレベルでの関係付け機能の提供

  KyotoDB

  AESOP

ソフトDA/DD

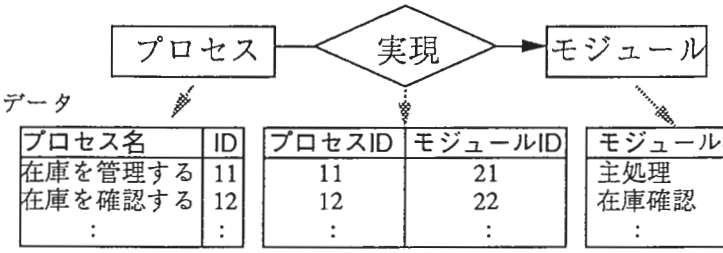
岡, 山本、1990

ドキュメント間の関係に基づく影響範囲解析機能を提供

設計情報の種別
データフロー
機能仕様
モジュール構成図
データ構造図
メッセージ
テスト項目
プログラム
改造要求票
ファイル

設計情報間の関係の例	
モジュール構造図↔ロードモジュール プロセス ↔プロセス仕様 データフロー図 ↔プログラム	1:1
モジュール構成図↔モジュール データフロー図 ↔データストア データ構造図 ↔ファイル	1:n
パラメータ ↔関数 改善項目 ↔関数	n:m

スキーマレベル



主な機能

(1) 関係を用いた設計情報検索

意味(説明)情報
定義位置
依存箇所
参照箇所

(2) 関係 "機能仕様<->その実現" を用いた修正影響範囲解析*1

*1---その他、過去に行った類似する改造事例を参照する機能を提供

[SoftDA/DD '90]

AESOP (Advanced Environment for SOftware Production)

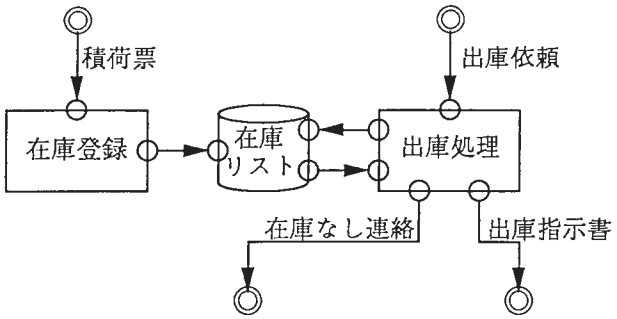
西川、寺田 (大阪大学)、松下電器産業、シャープ、三洋電機、三菱電機 等、1990

仕様から実行可能な高度並列処理プログラムを生成

事務処理分野における仕様記述 --- 6 つの図表から行う

表現形式	記述内容
機能ブロック図	モジュール間の接続関係
シーケンスチャート	入出力データの因果関係、モジュール間のデータ送受の関係
データブロック図	データの包含関係
関係表	関係データ構造
表操作図	(関係演算を中心とする)構造体データ処理
決定表	選択構造

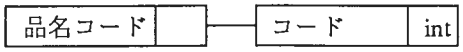
[AESOP '90]



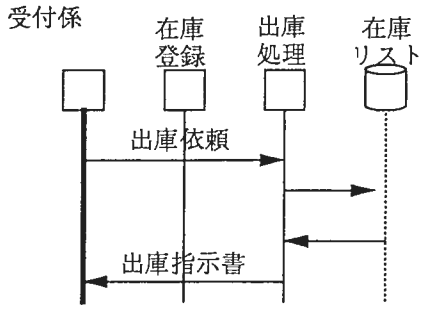
機能ブロック図

品物リスト	
品名コード	数量
int	int

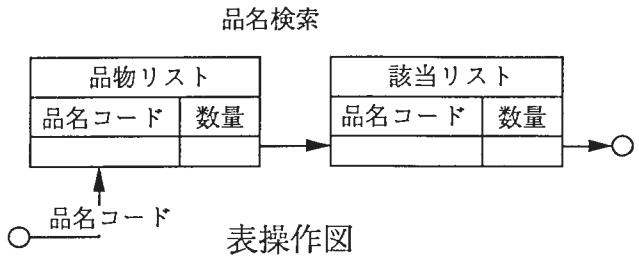
データブロック図



関係表



シーケンスチャート



表操作図

検索キー種別判定		
検索キー種別 == 品名コード	Y	N
品名検索	X	−
数量検索	−	X

決定表

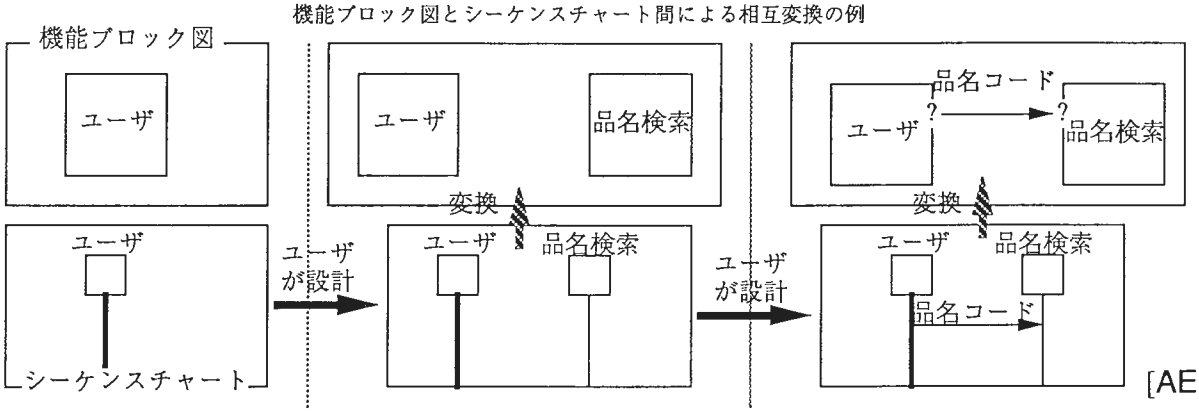
[AESOP '90]

各表現形式間の関係付け、 相互変換機能

多面的な図的表現

核となる情報	機能ブロック図	シーケンスチャート	関係表	データブロック図	表操作図	決定表
モジュールの動作及びモジュール間のデータ接続	✔	✔			✔	✔
モジュールの入出力データの因果関係 (シーケンス情報)		✔				
データ構造の定義(データ構造情報)			✔	✔	✔	

機能ブロック図とシーケンスチャート間による相互変換の例



[AESOP '90]

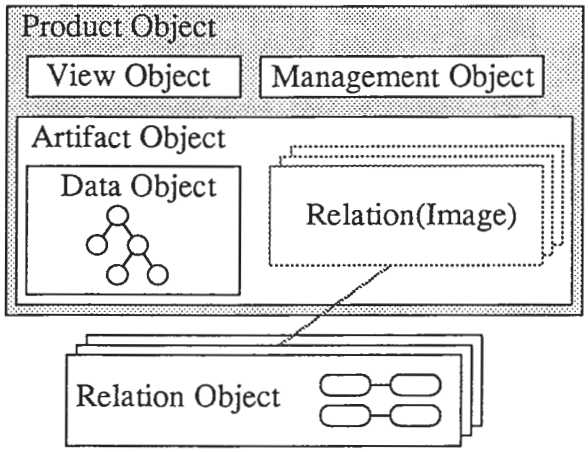
KyotoDB

松本 吉弘 他, 1990

ソフトウェアエンジニアリングデータベース(レポジトリ)

ソフトウェア構成要素とその間の関係の管理 → ソフトウェア構成要素間の一貫性保持

Object指向 (information hiding)



KyotoDBのオブジェクト構成

コラボレーション

ある要素の更新要求があった場合、KyotoDBは、それと依存性を持つ要素を扱っているユーザと要求したユーザとによる話し合いの場を設定

影響解析

影響仕様の変更が下流に及ぼす影響の追跡を容易化

格文法を用いた関係情報記述の定式化手法

(注) 関係はユーザがすべき記述しなければならない。

[KyotoDB '90]

3. リポジトリ

information repository, SEDB(software engineering database), data dictionary etc.

チーム開発においてメンバがソフトウェア設計にかかわる種々の情報を共有する為のツール。

特徴
- ❑ 複数のユーザが共有
- ❑ 種々のドキュメント型からなる
- ❑ 分散環境対応
- ❑ 標準化
- ❑ セキュリィティ/セーフティ

主な機能
- ❑ アクセスサービス (分散環境対応) ←現状
- ❑ 整理 (一貫性保持、流用 etc.) ←技術的課題
- ❑ セキュリィティ/セーフティ ←技術的課題

関連技術
- ❑ オブジェクト指向, 演繹データベース、制約論理プログラミングなど

[REPOSITORY '90]

4. ハイパーテキストによる ドキュメンテーション

複雑な論理構造を持つ設計情報(ドキュメント)の表現, 操作技術として、
また最近はマルチメディア技術としても注目

- 意味構造の表現
  - 技術文書
    - COSMOS
  - 設計情報 (含むドキュメントとコードの一体管理)
    - SoftDA/PC
- ドキュメントテンプレート
  - DIF

Hypertext

考え方

- 非線形テキスト
- テッド· ネルソン (Hypertext の 父)
- Hypertext System : Hypertext 作成、検索(ブラウジング)機能

特徴

- 必要な情報を容易に参照できる。
  - ☞ 仕組まれたオンラインドキュメント (HyperCard的)
    例 : 保守を前提としたドキュメント。しかし、作成工数大。
- 半形式的 -> 意味の一部をコンピュータが操作可能な形で表現
- オープン出版
- 読み書き順序に対する自由度 大 (-> 教材, アイディアプロセッサへの応用)

問題点

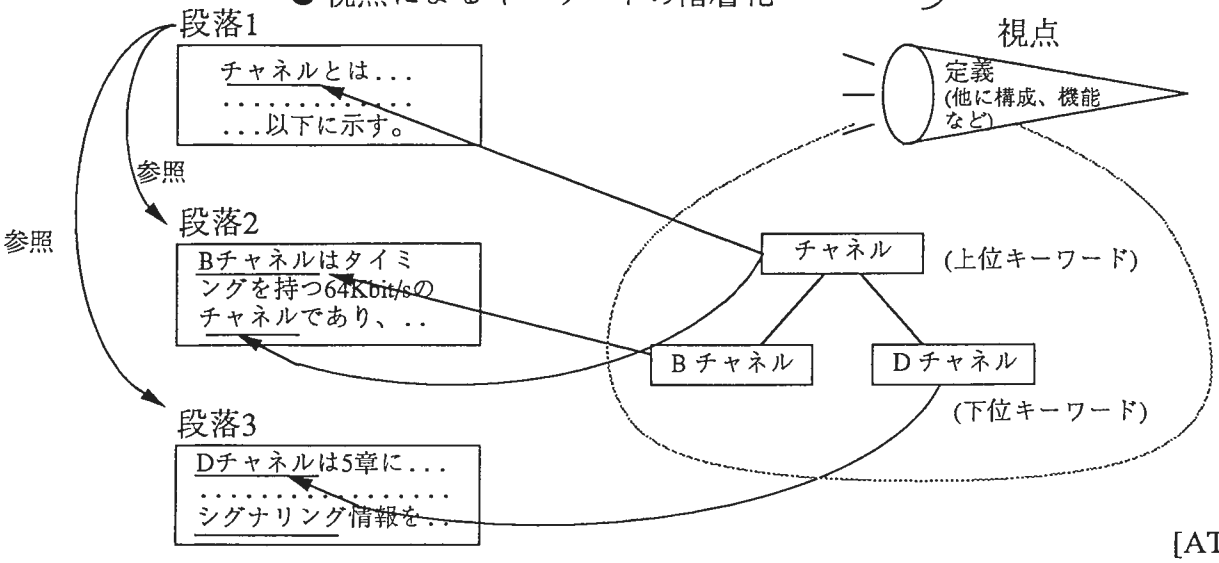
- リンクの完全性
- リンクを張る工数
- スパゲッティ構造
- 自由度が大きい -> 何を記述すれば良いのか、個人差の拡大

通信知識体系化支援システム COSMOS

技術書に出てくる概念を体系化しておき、設計で用いる場合の技術文書の理解性を向上させる。

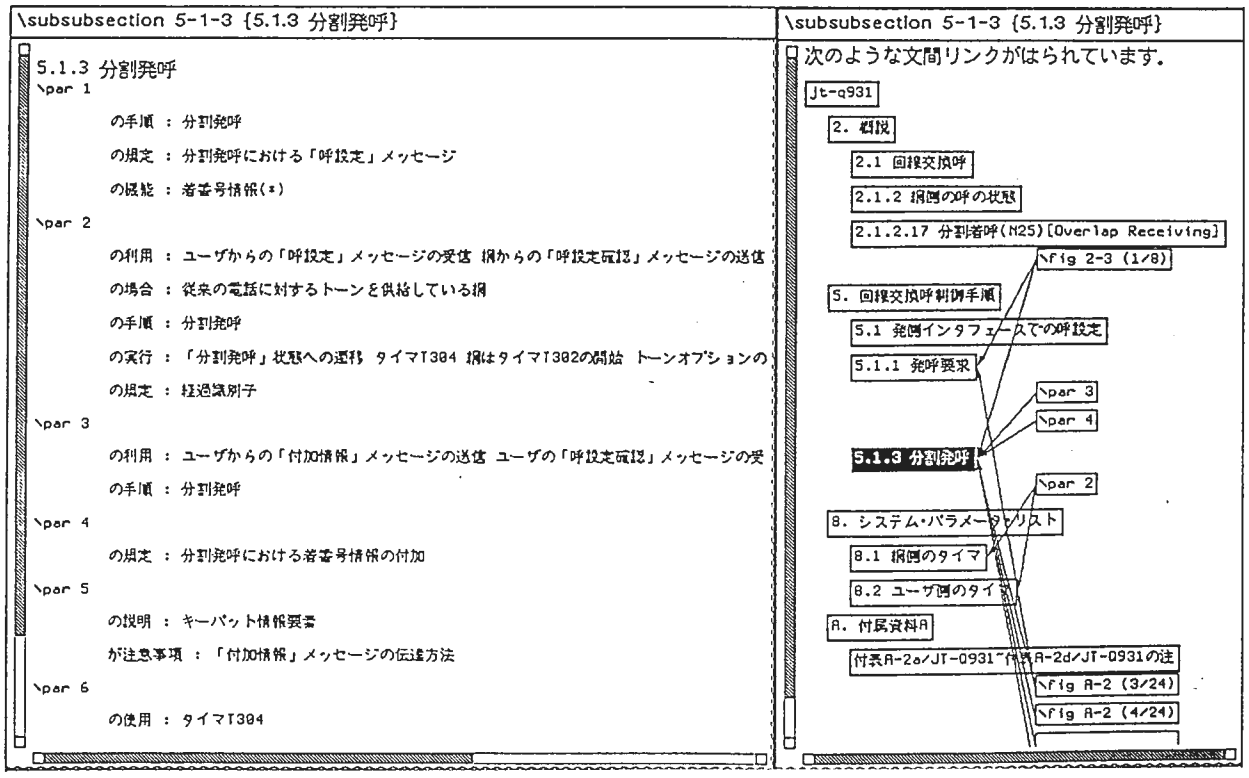
- 目次、章立てなどの文書構造の自動抽出
- 重要語彙･重要語句(キーワード)自動抽出
- 視点によるキーワードの階層化

意味構造



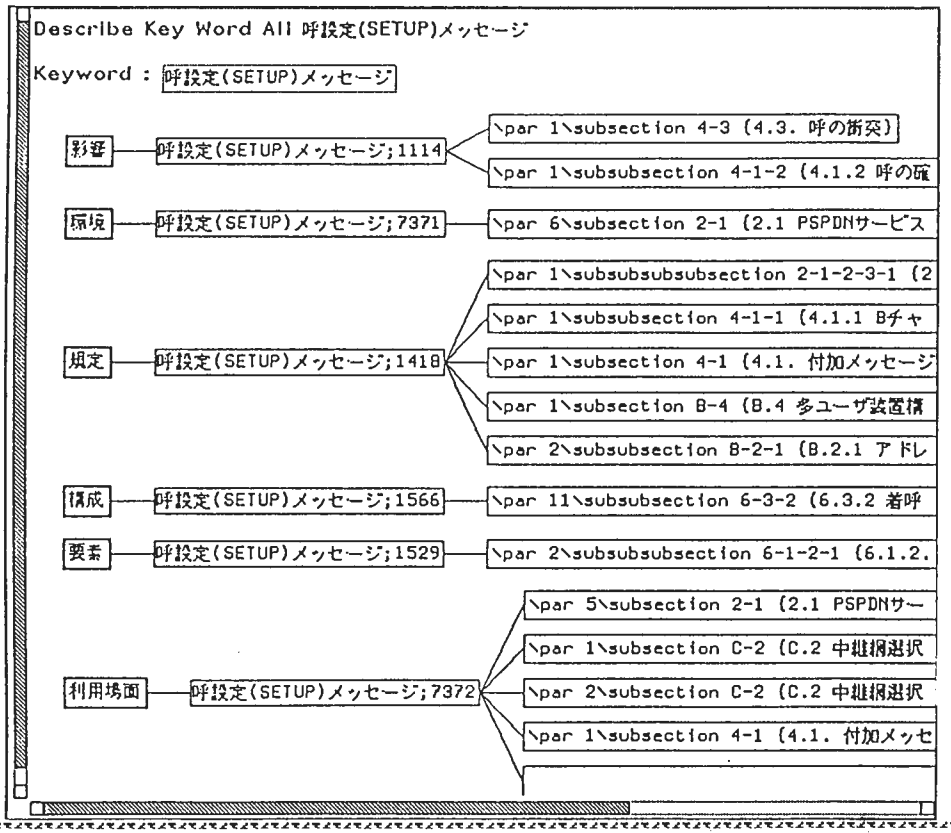
[ATR'89]

COSMOS : 文書構造



[ATR'89]

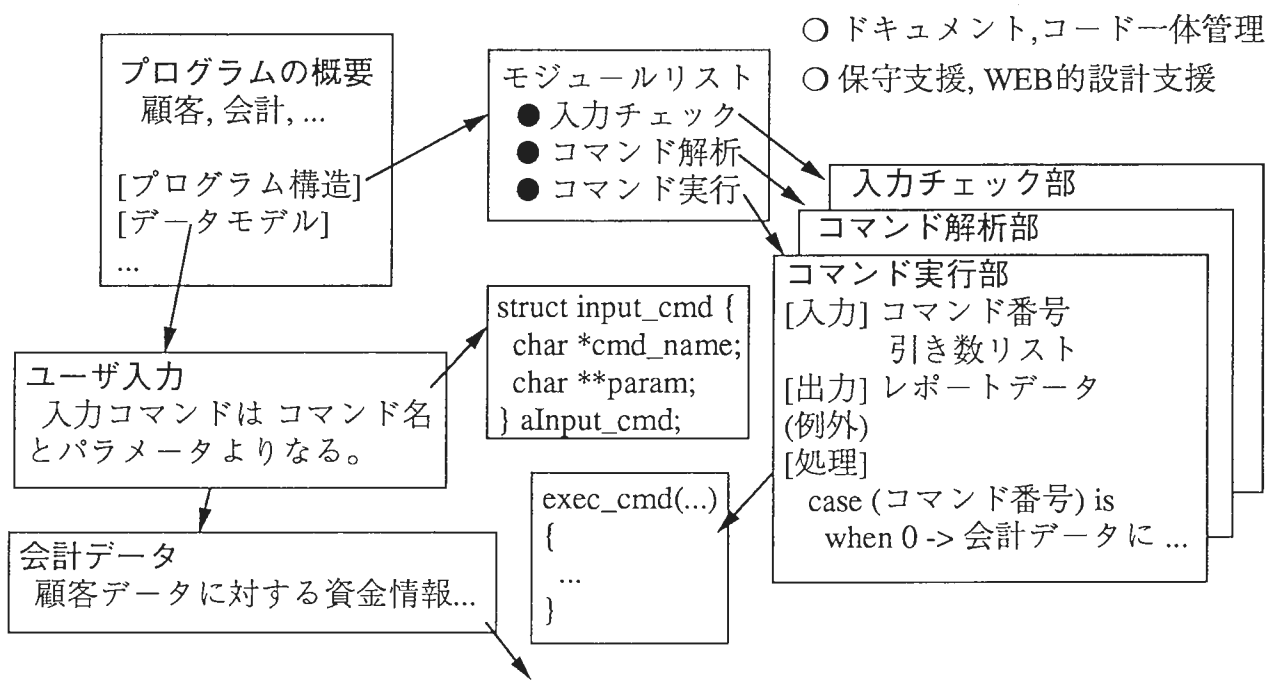
COSMOS : 意味構造



[ATR'89]

SoftDA/PC

設計情報モデル



[SoftDA/PC '87]

DIF (Documents Integration Facility)

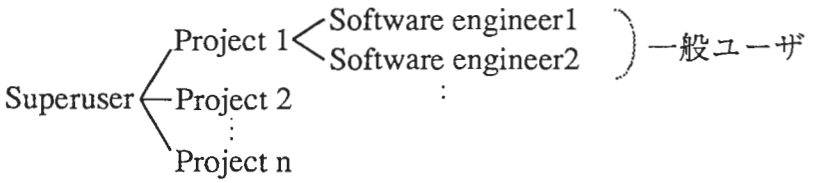
Pankaj K.Garg and Walt Scacchi (Univ. of Southern California), 1990

全ライフサイクルにおけるドキュメントをHypertextで管理

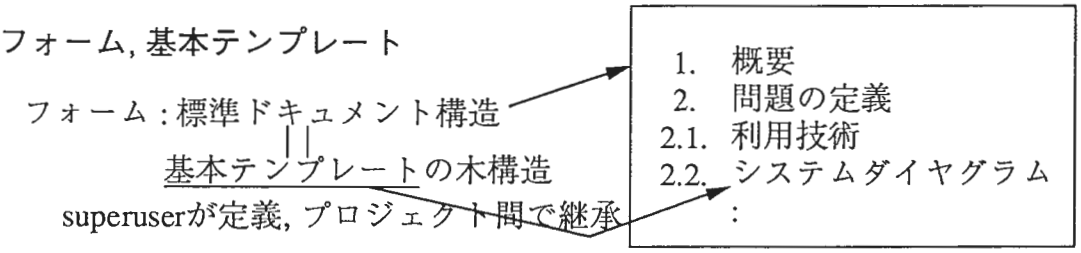
ドキュメント

要求定義	構成仕様	ソースコード	ユーザマニュアル
機能仕様	詳細設計書	テスト仕様書·品質保証書	システムメンテナンスガイド

System Factory structure



フォーム, 基本テンプレート



[DIF '90]

ブラウジング

リンクを辿る, キーワードを用いて検索する。

(1) リンク

基本テンプレート間, ポイントーポイント

(2) キーワード

基本テンプレートに、その意味を表すキーワードを張ることができる(ユーザ毎)。
パターンマッチング等により検索可能
Ingres databaseで管理

コンポジション

ユーザが自由に基本テンプレートを組み合わせたコンポジションを作ることができる(他のプロジェクトに継承されない)。

[DIF '90]

その他

- 設計情報の範囲

  IEEE Standards Project 1016

- 設計情報記述手法の分類

  設計情報記述手法の適用領域

IEEE Standards Project 1016
Design Entity Attributes

1986

Entity Attributes \ Design Users	Project Manager	Configuration Manager	Software Designers	Programmer	Unit Tester	Integration Tester	Maintainer
Identification	✓	✓	✓	✓	✓	✓	✓
Designer		✓		✓			✓
Source	✓						
Type	✓	✓				✓	✓
Purpose	✓	✓					✓
Function	✓		✓			✓	
Subordinates	✓						
Dependencies		✓				✓	✓
Resources	✓	✓				✓	✓
Interfaces			✓	✓	✓	✓	
Processing				✓	✓		
Data				✓	✓		
Tradeoffs							✓
Assumptions							✓
References				✓	✓		

[IEEE 1016 '86]

IEEE Standards Project 1016
DESIGN VIEWS

DESIGN VIEWS	SCOPE	ENTITY ATTRIBUTES	DESIGN USERS
Decomposition Description	System -> design entity への 分割	identification, type, purpose, function, subordinates, resources, source	Project Manager
Dependency Description	entity 間の 関係を 記述する	identification, designer, type, purpose, dependencies, resources, tradeoffs, assumptions	Configuration Manager, Maintainer, Integration Tester
Interface Description	各 entity の インタフェース 仕様	identification, function, interfaces	Designer, Integration Tester
Detail Design Description	各 design entity の 詳細 設計	identification, designer, interfaces, processing, data, references	Unit Tester, Programmer

[IEEE 1016 '86]

設計情報記述手法の適用領域

Dallas E.Webster, 1988

ソフトウェア(上流)							
一般的設計情報	要求	仕様	設計	データベース	Expert Systems	自然言語	汎用
KL-ONE	KL-ONE	KL-ONE	Cake	E-R	Proteus	SNePS	KL-ONE
Plane Text	Plane Text	Plane Text	Plane Text	Orion	CYC	KL-ONE	NIKL
Gist		Gist	Gist	Prolog	MRS	NIKL	Gist
Orion		Draco	Draco		Prolog	Prolog	Proteus
Proteus		PAISLey	PAISLey		KEE		CYC
CERM	RSL	RSL	RSL		ART		
RML	PSL	PSL	PSL				
	RML	SARA	SARA				
		SSA	SD				
	HIPO	HIPO	HIPO				
	SA	SA	SA				
		JSD	JSP/JSD				
		Miranda	Miranda				
		Prolog	Prolog				
		Statecharts	OOD				
		META-IV	META-IV				
		Refine	Refine				
		Raddle	Raddle				
			Petri Nets				

[Webster '88]

Conventional	IEEE Standard 1016	Broad spectrum information concepts (MCC)
Process/Functions	Designer	Objects
Data/Data flow	Sources	Stakeholders
Control/Control flow	Type	Issue Bases
Mechanisms	Purpose	Notes/Commnets
Performance	Function	Artifacts
Resources	Subordinates	Virtual objects
	Dependencies	Relationships
	Resources	Temporal relations
	Interface	Dependencies
	Processing	Constraints
	Data	Aggregates
	Tradeoffs	Decompositions
	References	Working relationships
	Standard views	

参考文献

[PPK]

- 田村 直樹、 中島 毅、 藤岡 卓、 上原 憲二、 高野 彰, "ハイパテキストを 用いた 設計 プロセス 支援 ツールの 試作",情報処理学会 ソフトウェア 工学研究会 68-7 (1989.9.26)
- 中島 毅、 田村 直樹、 上原 憲二 , "JSP法を 用いた 設計 プロセスの 記録と 分析",情報処理学会 ソフトウェア工学 研究会 71-4 (1990.2.8)

[ATR]

- 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支援, 保守方式", 情報処理学会第42回全国大会 投稿中
- 島 健一, "通信技術文書の構造化, 体系化について", 情報処理学会 情報学基礎研究会 Vol.89, No.85, 15-3, 1989

[AESOP]

- 日根 俊治, 芳田 真一、 西川 洋一郎、 山崎 哲男、 稲岡 美恵、 西川 博昭、 寺田 浩詔 他, 超 高位 図的 仕様 記述 環境 (AESOP) プロトタイプ 関連の報告,情報処理学会第41回全国大会 5-265, ..., 5-273

[KyotoDB]

- 沢田 篤史, 松本 吉弘 他, "ソフトウェアエンジニアリングデータベースKyotoDB核のプロトタイピング", 情報処理学会第41回全国大会 5-343, 5-345, 3-353

[ソフトDA/DD]

- 岡 敦子, 山本 修一朗, "設計情報レポジトリを用いた改造支援方法", 情報処理学会第41回全国大会 5-351

[SoftDA]

- Shuichiro Yamamoto and Sadahiro Isoda, "SOFTDA---A REUSE-ORIENTED SOFTWARE DESIGN SYSTEM", Proceedings of COMPSAC-86 October 8-10,1986

[WEB]

- Donald E.Knuth, "Literate Programming", The Computer Journal, Vol.27, No.2, 1984 (日本語訳: 黒川利明, bit Vol.17, No.4)

[DIF]

- Pankaj K.Garg and Walt Scacchi, "A Hypertext System to Manage Software Life-Cycle Documents", IEEE Software, May 1990

[gIBIS]

- 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

[Potts]

- Colin Potts and Glenn Bruns,"Recording the Reasons for Design Decisions","Proceedings of 10th ICSE, 1988

[IEEE 1016]

- H.Jack Barnard, Robert F.Metz, Arthur L.Price (AT&T Bell Laboratories), "A Recommended Practice for Describing Software Designs : IEEE Standards Project 1016", IEEE Transactions on Software Engineering Vol.SE-12, No.2, February, 1986

[その他]

- Dallas E. Webster (MCC), "Mapping the Design Information Representation Terrain", IEEE Computer, 1988.12

設計プロセス獲得支援機能

1. 設計プロセスを記録する上での問題点

本文の4章で述べた設計プロセス記録の方法には，以下の問題点が存在する．

③では，設計者が参照漏れを注意する必要がある．また，①～③では，設計順序の試行錯誤(参照が必要な設計ビューが未設計)により設計の中断，再開が発生し記録作業が繁雑になる．更に，④では，従来設計生産物に記述していなかった情報を設計者が記述する必要が有る．

これら設計プロセス記録上の工数増加の要因への対策として設計ガイド機能を提供する．

2. 設計ガイド機能

本機能は，以下の2つの機能より構成される．

・ 視点ガイド：本文で述べたように，視点には，設計法で具体的な指定があり予め決まるものと，設計時に設計者により認識されるものとが存在する．本機能は，前者を，ソフトウェア開発現場単位に整理した視点ガイドラインを利用して，後者を，過去の設計プロセスの記録を利用してガイドする．具体的には，視点ガイドラインを，視点利用規則として記述する．視点ガイドはまず，視点利用規則を評価し，設計目標<キー設計エンティティ，視点>(キー設計エンティティと視点の対を表す)を設者に提案する．ユーザはその提案を基に，設計目標を設定する．次に，視点ガイドは，設計目標を設計するまでに途中設計すべき設計ビューと，その際考慮すべき設計ビューを，順を追ってアドバイスする(上記①～③の支援)．

・ 設計事例検索：再利用可能と思われる設計ビューを設計事例データベース(以下事例ベースと呼ぶ)より検索し，設計者に提示する(上記④の支援)．

2.1 設計目標の計算方法

まず、設計目標の計算で利用する視点利用規則の記述形式について説明し、次に設計目標の計算の手順について述べる。

(1) 視点利用規則

１つの視点ガイドラインは、視点利用規則の集合により表される。

視点利用規則はPrologの表記法および評価機構を用いる．具体的には以下の書式を持つ．

<ルールインデックス> :- <設計生産物の条件>の並び.

<ルールインデックス>は，rule(通番)の書式を持つ．通番は，ルールを評価する順番を表す自然数である．<設計生産物の条件>は，<設計生産物が満たすべき条件>または<設計目標の提案>から成る．ルールの右辺には1つ以上の<設計目標の提案>が存在する．意味は，ルール右辺にある<設計目標の提案>は，それより左にある<設計生産物が満たすべき条件>がすべて満たされた場合に(設計目標の提案が)行われることを表している．

視点利用規則として、ルール形式を採用したのは、設計者による設計目標提示機能の選択的な利用を可能にするためである。即ち、ルール形式の記述により、本機能が任意の時点で起動された場合、それまでの設計プロセスを追跡していなくても設計目標の計算が可能になることによる。

視点利用規則の記述例を以下に示す。例えば、構造化設計手法(変換分析手法)における、「(ある)モ

ジュールについてデータ要素という視点から設計した結果を基に、そのモジュールの従属モジュールそれぞれについて、入出力データを設計する」という依存関係は、次のように記述する。

```
rule(1) :-  design_view(X, データ要素),
            design_entity(X, 従属モジュール, Y),
            design_all(Y, [入出力データ]).
```

大文字の英字(例ではXとY)は変数を表す．右辺は，最後の項が<設計目標の提案>であり，それ以外の2つの項が<設計生産物が満たすべき条件>である．例では，まず最初の項で，設計プロセスの記録の中で視点「データ要素」を持つ設計ビューのキー名が変数Xとユニファイされる．次の項design_entityでは，Xで表されるキーと視点「従属モジュール」を持つ設計ビューの内容にある設計エンティティのリストとYがユニファイされる．最後の項，design_all(キーのリスト, 視点のリスト)は，キーのリストと視点のリストの各要素の積を設計目標として提案することを示している．例では，Yの各要素をキー，視点「入出力」の対を設計目標とすることを表している．

(2) 設計目標の計算手順

Prologの評価機構を用いて，ルールインデックスのついている各節を通番順に評価していき，その過程で<設計目標の提案>項を実行した結果として設計目標が得られる．設計生産物の条件を満たすケースはすべて求める(ルールの節を意識的にfailさせて計算する).

例えば、前述の視点利用規則が上記条件を満たすかどうかは以下のようにして調べる (図4.1)。

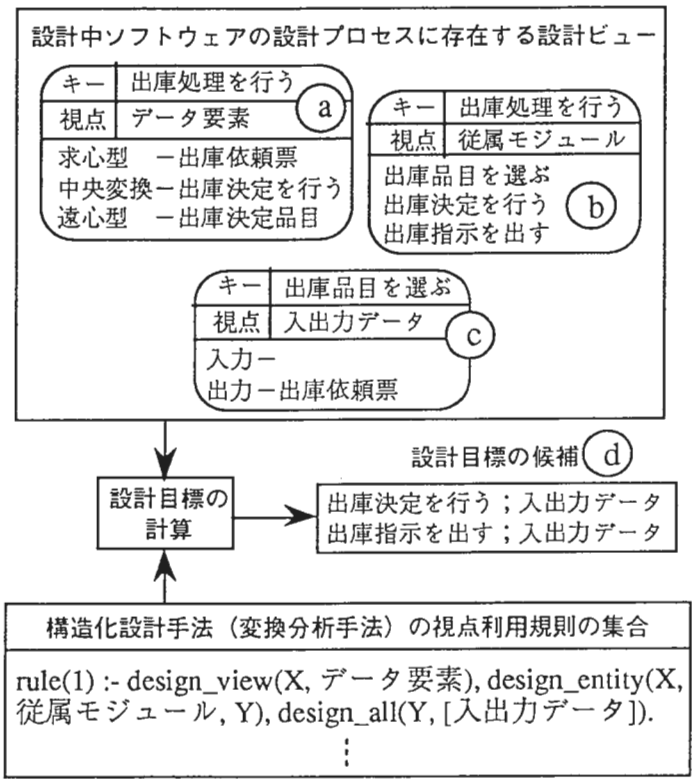


図4.1 設計目標計算例

まず、右辺の第一項の評価では、設計プロセスの記録から、視点「データ要素」を持つ設計ビューを1つ検索する。図4.1 の例では(a)が見つかる。見つかった設計ビューのキー設計エンティティ(例では「出庫処理を行う」)とXがユニファイされる。

次に右辺第2項の評価では、キー設計エンティティ「出庫処理を行う」、視点 「従属モジュール」を持つ設計ビュー(図中b)の値に含まれる設計エンティティ(図4.1の例では、[出庫品目を選ぶ, 出庫決定

を行う, 出庫指示を出す])とYがユニファイされる.

右辺第3項の評価では, Yそれぞれの要素をキー設計エンティティ、視点として「入出力データ」を持つ設計ビューの中で、設計中の設計プロセスに存在しないものを設計目標とする(図中cが存在するため、設計目標はdに示す通りとなる)。以上を満たす, 設計プロセスの記録の他のデータの組み合せについても求める.

2.2 視点ガイド機能

設計目標を設計するまでに途中設計すべき設計ビューと, その際考慮すべき設計ビューを, 順を追ってアドバイスする方法について述べる.

本機能および次に説明する設計事例検索は, 事例ベースを用いて実現する. 事例ベースは, 問題領域毎に記録した複数の設計プロセスを, <キー設計エンティティ, 視点>に対する設計結果のバリエーションと, そのバリエーション間の利用関係として構造化して蓄積している. 在庫管理プログラムの設計事例2つを格納した事例ベースの例を図4.2に示す.

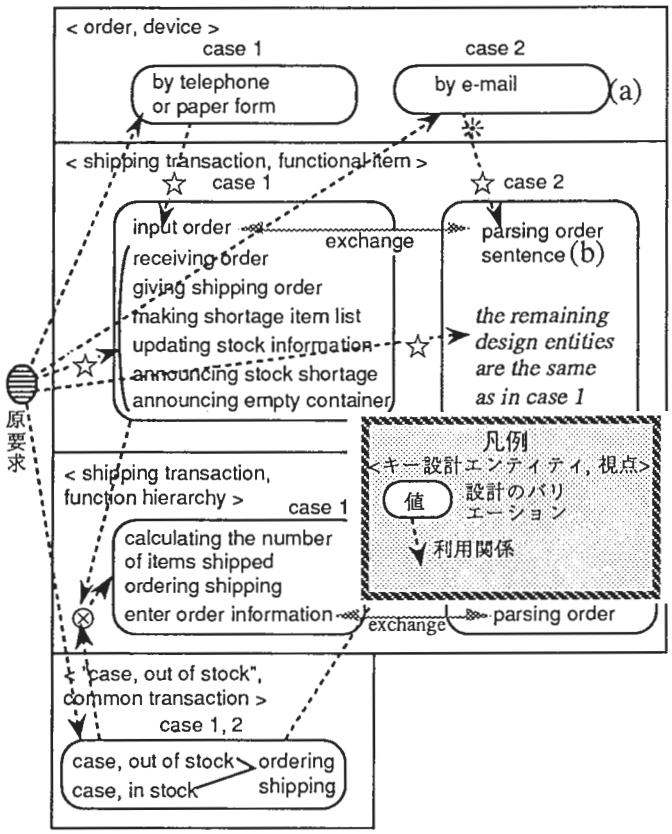


図4.2 設計事例データベース

視点ガイド機能は, 特定の設計ビューの設計において過去利用された設計ビューに関する情報を提示する. 以下に手順を示す. まず, システムは, 事例ベースの利用関係から,"<キー設計エンティティ, 視点>=>(左辺で表される設計ビューの設計に利用した設計ビューの<キー設計エンティティ, 視点>)"から成る規則(プロダクションルール)を作成する. 図4.2の場合, 以下に示す規則が得られる(例えば, 最初の規則は図中の利用関係☆より得られる).

<shipping transaction, functional item> => <order, device>, <o-req>

<order, device> => <o-req>

<shipping transaction, function hierarchy> => <shipping transaction, functional item>, <"case, out of stock",

common transaction>

<"case, out of stock", common transaction> => <o-req>

ここで，<o-req>は原要求を表す．次に，< >を1つの項と見なし，ユーザが設定した設計目標<キー設計エンティティ，視点>に対して上記規則による展開(左辺を右辺で)を繰返し，その過程を設計目標をルートとする木構造(設計展開木と呼ぶ)で表す．項<キー設計エンティティ，視点>が以下の条件を満たす時，その項の展開を終了する．

- 設計中の設計プロセスにおいて設計済または設計展開木中に既に存在する
- 適用できる規則が存在しない

図4.2の例において設計目標が<shipping transaction, function hierarchy>の場合の設計展開木を図4.3に示す．システムは，この設計展開木に基づき，それまでに設計されている設計ビューから設計目標を設計するまでに設計すべき設計ビューと，その際考慮すべき設計ビューを，順を追ってアドバイスする(図4.3の例では「まず，原要求を基にorderを視点deviceから設計し，次にその結果と原要求を考慮してshipping transactionを視点functional itemから設計する，...」となる)．

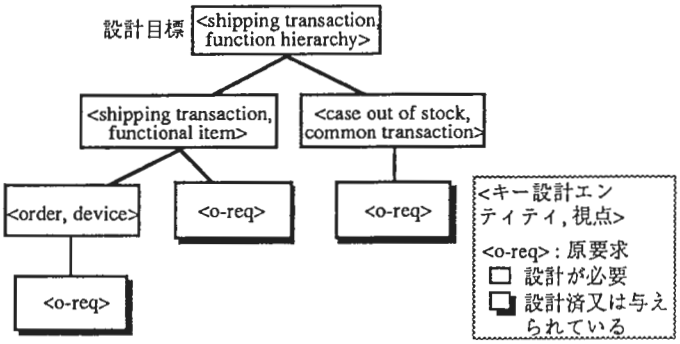


図4.3　設計展開木の具体例

2.3　設計事例検索

設計事例検索では，特定の設計ビューAへの再利用候補を以下の方法で決定する．まず，設計ビューAの設計で考慮すべき設計ビューBの内容と一致する事例ベースの設計のバリエーションを検索する．次にその一致した設計のバリエーションと利用関係を持つ設計のバリエーション(またはその一部)を検索し，再利用候補とする．例えば，図4.2において，設計者は，<shipping transaction, functional item>の設計の際，考慮すべき<order, device> に対して(a)と同じ設計結果を記述したとする．システムは，<shipping transaction, functional item> の設計例として，"parsing order sentence"を提案する(図中の利用関係※を利用)．

完全に一致する設計のバリエーションが存在しない場合は再利用候補を以下の方法で決定する．今，利用関係の事例を"設計ビュー→ 設計ビュー"(左辺を利用して右辺を設計)と記す．利用関係の複数の事例において，左辺の設計ビューの事例間での共通部分は，右辺の事例間での共通部分を設計した原因である可能性が高い．この考え方に基づき再利用候補を求める(図4.4)．事例ベース内の利用関係の事例のうち，左辺，右辺の設計ビューの<キー設計エンティティ，視点>が設計ビューB，Aそれぞれのものと一致するものの中から，左辺の設計ビューに共通部分があるものを限定する(図4.4①)．次に，その限定した利用関係の右辺の事例間での共通部分を設計ビューAの再利用候補とする(図4.4②)．

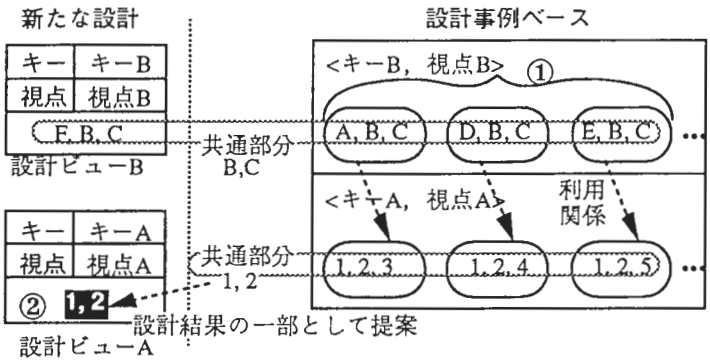


図4.4　設計事例検索の例

参考文献

1) L. Osterweil: Software Processes are Software Too, Proceedings of the Ninth International Conference on Software Engineering (1987).

2) L.Williams: Software Process Modeling : A Behavioral Approach, Proceedings of the Tenth International Conference on Software Engineering (1988).

3) D. Knuth: Literate Programming, The Computer Journal, Vol.27, No.2 (1984).

4) J.Conklin, M.Begeman: gIBIS: A Hypertext Tool for Exploratory Policy Discussion, ACM Transactions on Office Information Systems, Vol.6, No.4, October (1988).

5) C. Potts and G. Bruns: Recording the Reasons for Design Decisions, Proceedings of 10th ICSE (1988).

6) Z. Liu, A. Farley: Shifting Ontological Perspectives in Reasoning About Physical Systems, AAAI-90 (1990).

7) B. Falkenhainer, K.Forbus: Setting up Large-Scale Qualitative Models, AAAI-88 (1988).

8) V. Dhar, M. Jarke: Using Teleological Design Knowledge For Large Systems Development And Maintenance, Proceedings of the International Workshop on Expert Systems&Their Applications (1986).

9) R. Guidon: A Model of Cognitive Processes in Software Design,MCC Technical Report STP-283-87 (1987).

10) M. Lubars: Representing Design Dependencies in an Issue-Based Style, IEEE Software July (1991)

11) T. Demarco: Structured Analysis and System Specification, Yourdon Press (1986).

12) R. Fjeldstad, W. Hamlen: Application program maintenance study - report to our respondents, IBM Corporation (1979).

13) M. Lehman, Programs: Life Cycles, and Laws of Software Evolution, Proceedings of IEEE, Vol.68, No.9, September (1980).

14) J. Lee: Extending the Potts and Bruns Model for Recording Design Rationale, Proceedings of the 13th International Conference on Software Engineering (1991).

15) Hideaki Takeda, Paul Veerkamp, Tetsuo Tomiyama, Hiroyuki Yoshikawa: Modeling Design Processes, AI Magazine Vol.11 No.4 Winter (1990).

16) A. Czuchry, D.Harris: KBRA: A New Paradigm for Requirement Engineering, IEEE Expert, Vol.3, No.4, Winter (1988).

17) 浜田, 安達, 竹中: 設計プロセスの蓄積·利用による設計支援法について, 情報処理学会ソフトウェア工学研究会Vol.91, No.66 (1991).

18) 安達, 浜田: 保守支援のための設計プロセス獲得システム、ソフトウェアシンポジウム'92(1992).

19) Guillermo Arango, Laurent Bruneau, Jean-Francois Cloarec, Albert Feroldi: A Tool Shell For Tracking Design Decisions, IEEE Software, March (1991).

20) Balasubramaniam Ramesh, Vasant Dhar: Supporting Systems Development by Capturing Deliberations During Requirements Engineering, IEEE Transactions on Software Engineering, Vol.18, No.6, June (1992)

21)島健一:通信技術文書体系化システム-COSMOS，電子情報通信学会データ工学研究会DE89-15(1989)

22)Jean Hartmann, David J.Robson: Techniques for Selective Revalidation, IEEE Software, January 1990.

23)Guillermo Arango,Peter Freeman, and Christopher Pidgeion: TMM:Software Maintenance by Transformation, IEEE Software, May 1986.

24) B. Adelson, E. Soloway: The Role of Domain Experience in Software Design, IEEE Transactions on Software

Engineering, Vol. SE-11, No.11 (1985).

25) K. Inoue, T. Ogihara, T. Kikuno, K. Torii: A Formal Adaptation Method for Process Descriptions, the Eleventh International Conference on Software Engineering (1989).

26) P. Garg, W. Scacchi, A Hypertext System to Manage Software Life-Cycle Documents, IEEE Software, May (1990).

27) J. Conklin, Hypertext:An Introduction and Survey, IEEE Computer, September (1987).

28) R. Prieto-Diaz: Domain Analysis for Reusability, COMPSAC'87

29) K. Okamoto, M. Hashimoto:Visual Programming with Reusable Specifications Described by a Conceptual Data Model- and Constraint-Based Language, Human Aspects in Computing (Elsevier) (1991)

30) Kouichi Kishida, et.al.: SDA: A Novel Approach to Software Environment Design and Construction, Proceedings of 10th ICSE (1988)

31) D.Navinchandra: Case Based Reasoning in CYCLOPS, a Design Problem Solver, Proceedings on CASE-BASED REASONING WORKSHOP, May (1988)

32) 山口, 落水: 事例ベース推論とモデル推論の相互作用に基づくソフトウェアプロセスモデル獲得支援環境, 知能ソフトウェア工学の動向と展望シンポジウム(電子情報通信学会) (1992)

33) Stephen Slade: Case-Based Reasoning: A Research Paradigm, AI Magazine, Spring (1991)

34) Mitchell D.Lubars: The IDeA Design Environment, Proceedings of 11th ICSE, (1989)

35) Ruben Prieto-Diaz, Peter Freeman: Classifying Software for Reusability, IEEE Software, January (1987)

36) 浜田, 安達: 設計プロセスの再利用による設計ガイド方式, 情報処理学会ソフトウェア再利用技術シンポジウム (1992)

37) 石坂裕毅, 有川節夫: モデル推論, 情報処理学会誌 Vol.32, No.3, March (1991)

38) 吉浦 裕: 問題と事例の分割に基づいて部分的類似例を利用する事例ベース推論方式,情報処理学会論文誌 Vol.32, No.5, May (1991)

付録5

今後の研究課題

今後の研究課題としては，設計プロセスの記録のコストを低く抑え，かつ記録するデータの質を安定化させる方法について検討を進める必要がある(設計プロセス獲得支援)．現在，過去の設計プロセスの再利用および予め設計法で指示されている設計手順を表した視点利用規則を利用して，これらを実現する方法を検討している．この方法を実現するためには，更に以下の3つの大きな課題があると考えられる．

1) 検索技術
2) 変更支援
3) 設計事例データベースの整理機能

以下，それぞれについて簡単に解説する．

(1) 検索技術

検索技術については，付録3で概ね述べた．一般に，検索技術の課題として，キーワードの類似性やキーワードの限定の問題がある．影響波及解析支援では，ユーザが欲しい情報をユーザが設計プロセスから探すのではなく，利用関係により候補として挙がってきた設計ビューから，修正に関連するものを選ぶために視点やキーを用いる．いわゆる目次として利用されている．保守者は，目次の中から修正に関連ありそうな場所を選んで参照する．従って，視点やキーは，通常の情報検索で用いられるキーワードとは異なる．通常の情報検索では，ユーザが欲しい情報というのをシステムに告げる手段としてキーワードを用いる．この方式では，キーワードの類似性などによりユーザの意図をシステムが解釈してやる必要が生じてくる．しかし，視点やキーについてはこのような問題はおきない．まず，利用関係により目次になるキー，視点の数を絞っているため，ユーザによる選択が可能になる．事実，今までの試用においてもそのような問題は生じていない．

では，設計プロセスを再利用する場合，再利用できる設計ビューの検索でこのような問題が生じるか．基本的には，影響波及解析支援と同じである．即ち，まず，利用関係で，再利用の可能性がある設計ビューの候補が絞られる．絞られた候補から，設計者は選べば良い．従って，まず利用関係により絞られる再利用可能な設計ビューの候補の数がある程度小さければ，選択方式による検索が可能となり，キーや視点の類似性による．

再利用の場合の問題点は，検索の問題よりも，むしろ，視点の枠の不偏性がない可能性があることである．即ち，例えば，ある設計では，設計者は，視点V1，V2などの設計を行っていた．別の設計者は，V1+V2の内容を違う観点で分けたV3, V4という視点から設計したくなるような場合が存在する場合，問題となる．

しかし，視点の多くは(「多く」の割合については今後評価が必要)設計法で陽に与えられること，また，設計ビューの情報の単位が比較的小さいこと，また，過去の設計プロセスとの差分を新しい設計で入力していくこと，更に，もし，どうしても設計上視点V3, V4からの設計が必要なら，視点V1, V2からの設計結果を参照しながら視点V3, V4からの設計を行う形で展開されることなどから，視点枠の可変性についてはあまり問題がないと考えている．

検索技術でもう一つ課題となるのが，事例の再利用性を広げる手法として，キーの類似性を導入するということである．類似したキーに対する設計事例を再利用するというものである．一つは，キーのis-

a階層をデータベースとして予め作成しておく(以後，設計で登場するキーをこの階層に追加していく)方法が考えられる．しかし，設計プロセスの情報があるので，これを基にキーのis-a関係を自動的に生成する方法が有効と考えられる．例えば，別々の設計事例において，同じキー，視点からの設計結果として書かれている設計エンティティe1, e2それぞれは，型が同じである可能性が高い．この関係を利用して，設計エンティティ間のis-a階層を設計事例データベースを基に自動生成する方法がある．関係の生成についてはダイナミックな追加機構が必要である．例えば，設計事例データベースにない設計エンティティを設計者が入力した場合，その時点で，設計事例データベースを解析し，その設計エンティティが，is-a階層のどこに位置すべきかを自動的に決定する．以後は，その設計エンティティについての設計支援を，is-a階層上で近い設計エンティティに関する設計事例を用いて支援する．このダイナミックな解析を適切なスピードで実現することも検討課題となる．

(2)　変更支援

設計プロセスの最適化の問題がある．過去の設計での視点の単位および利用関係が別の設計において使うことが適切かどうかが，または最適化するための支援などが必要かどうかなどである．

但し，本手法の特徴として，本手法の設計プロセスの記録は，設計の手続きを表しているのではなく，設計上の考慮の必要性を表しているため，利用関係については，有る程度不偏性があると考えられる．では，前述の，視点の枠の不偏性という問題になる．これは，述べたように視点の枠を過去のと同じにしても新しい設計はできるだろうということになる．但し，それが最適な設計になるかどうかは保証できない．設計のコントロールについては，この問題については今後の検討が必要である．基本的には，設計のコントロールとは，多くの設計事例を集めることで蓄積された利用関係のバリエーションから，新しい設計で採用するもんを選択する問題に帰着または近似できるのではないかと考えている．

(3)　設計事例データベースの整理機能

本研究では、まず、設計およびソフトウェアの修正を支援する為に役立つ情報の明確化およびその情報を設計中に獲得する方法について検討を行っており、実用化にはさらに以下の課題を解決する必要がある。

a)　事例ベースの整理(汎化、矛盾の発見･解消など)技術

b)　事例の選別方法

c)　設計プロセスの理解性

これらは事例ベース推論技術一般の課題であり、本研究では、今後AI研究での成果を参考にしながら取り組んで行く予定である。

現段階では、a)に対して、事例間における矛盾(考慮した設計ビューが同じで、その結果設計された設計ビューが異なる事例)を発見し、それらの事例に対して考慮した設計ビューの追加を促す方式を、b)に対して、事例ベースへの設計プロセスの登録を指定する機構等を検討している。