情報システム 42－5
（1993. 1. 19）

リエンジニアリングを活用した開発保守支援

園部正幸 上原三八 吉野利明 直田繁樹 秋山友子 木村美奈子 石崎あゆみ 大久保隆夫 川辺敬子
富士通研究所

リエンジニアリングを活用した，ソフトウェアのメンテナンスワークベンチについて報告する．
第一に，仕様の効率的電子化のために，ワーク変数消去，条件名生成，表生成，日本語生成の技術を開発して，ソースから仕様を逆生成するリバースエンジニアリングツールの要約度向上を達成した．
第二に，電子化仕様を利用するツール・環境に関してのモデルを作り，プロトタイプを開発した．ハイパテキスト，電子しおりなど非線型性を導入し，また，リバースエンジニアリングツールで生成される下流から上流への対応情報を効果的に活用することで検索効率を向上した．

キーワード： *CASE，リエンジニアリング，リバースエンジニアリング，ソフトウェアプロセス，統合，保守*

Software Development and Maintenance Support Using Re-engineering

Masayuki Sonobe Sanya Uehara Toshiaki Yoshino Shigeki Suguta Yuko Akiyama
Minako Kimura Ayumi Ishizaki Takao Okubo Keiko Kawabe
FUJITSU Laboratories Ltd.
Kamikodanaka 1015, Nakahara, Kawasaki, Kanagawa 211 Japan

Software development and maintenance support using re-engineering is reported.
First, abstraction level improvement of source-to-specification reverse engineering tool is accomplished by developing technologies of work variable elimination, condition name generation, table generation and Japanese generation for efficient electronization of specification.
Secondly, a model of tools and environment to utilize electronized specification and a prototype are developed. Retrieval efficiency is improved by introduction of nonlinearity such as hypertext and electronized bookmark, and by effectively usage of correspondence information from lower stream to upper stream, generated by the reverse engineering tool.

Keywords:
CASE, re-engineering, reverse engineering, software process, integration, maintenance

1．はじめに

官公庁，企業では，数千本のプログラムの大規模アプリケーションシステムが珍しくない．困難な大規模開発が完了しても，すぐ保守・拡張開発の困難さに直面する．あるプロジェクトでは1万本のプログラムがあり，ソースが10年間で7倍になるほどの激しい拡張を安全に行うために800人の部隊を擁している．計算機を活用してこうした既存資産の開発・保守を支援する分散環境の実現が急務である．

そこで，ソースプログラムから手続き的仕様を自動的に逆生成するリバースエンジニアリングツールと，この逆生成時に得られる情報を活用して，WS画面でプログラム，仕様書等の文書を効率よく検索・修正できるようにする開発・保守支援システム（仮称：メンテナンスワークベンチ）をS-4，UNIX の上で試作した．

2．課題

小さな開発の繰返しである保守が大変なのは，新規開発より一層，既存部分の理解や，想定する修正部分と既存部分の関係の迅速かつ正確な理解が必要な点である．このためにソースに伴って仕様書が保守されている．しかし，紙と鉛筆の仕様書保守が，非効率，不正確の原因になり，テストや障害対応を含めて多大な人件費と時間を要している．

そこで，既存資産の保守の短時間化，高信頼化のためには，

1） どのように仕様を電子化したら効率的か，

2） どのようなツール・環境により電子化仕様の保守を支援すれば効率的か，

を明らかにすることが課題である．1)，2)の連携による相乗効果も検討する．

3．解決アプローチ

上記課題を解決するアプローチを述べる．

3．1　仕様の電子化

第一の課題である仕様の効率的電子化には，次のようなアプローチが考えられる：

a） **イメージ入力** —— 仕様書をイメージリーダで入力して画面で保守する．文字コードで処理するCASEツールが使えないので効果は小さい．

b） **文字認識** —— イメージ入力し，文字認識でコードに変換する方法．認識の限界のため人手の修正がかかり過ぎる．その上，形式を標準化し，プログラムと一致させるためのチェックと修正が必要．

c） **コード入力** —— 仕様書を見て人がコード入力する方法．形式を標準化し，内容をプログラムと一致させるためのチェックと修正が必要．

d） **仕様逆生成** —— ソースから仕様ファイルを逆生成するツールを開発する．自動化すれば形式は標準化され，内容はプログラムと一致する．最も自動化率，信頼性が高い．また，ソースが修正されたときに，再実行により簡単に新仕様書を得ることができる．

d)の仕様逆生成が最も効果が大きい．しかし，従来，生成仕様の要約度が低いという問題があり，要約機能の高度化が必要である．この逆生成（reverse engineering）技術はソフトウェアの理解を支援するものであるが，同時にリエンジニアリング（re-engineering）[1] 技術の一部と考えている．すなわち，この仕様からプログラムを順生成（forward engineering）するコンパイラを開発して，保守性の高い新システムに移行（migration）することが重要である．

従来のリエンジニアリング技術として，富士通のYPSジェネレータは，COBOLソースを入力して，構造図に基づく言語である YPS仕様書を自動的に逆生成している（YPS:Yet Another Chart Ⅱ Programming System）．

YPS仕様書を入力とするコンパイラも提供されているので，生産性の高い，YPS言語だけの保守に移行可能である．ただし，COBOLソースの一命令が基本的にYPS ソースの一命令に変換されるので，要約度が高くならない．本研究では，「プログラムの開発（発注）時に書かれるプログラム仕様書に近い要約度」を目標とする．

3．2　ツール・環境

仕様が電子化されても，紙より効率的になる作業と非効率になる作業がある．後者を解決し，また，種々の効率化対策を講じて「総合的に紙を越える保守効率性を得ること」を目標とする．

ここで，保守効率性(maintenance efficiency)＝「修正機能量」／「案件または障害が生じてから，システムが修正されて修正に伴う障害解決が完了するまでの工数」という定義を提案したい．

従来，電子化文書のWSでの操作が紙に劣る理由として，特に，

1)　検索効率が悪い

2)　どこをどう修正したのかが一覧できない

という点が言われているので，その解決を図る．

これは従来のフラットファイル（flat file）の考え方に限界があり，ハイパテキスト（hypertext），しおり（bookmark）といった非線型性を導入すれば解決できるのではないかと考えた．

また，仕様コンパイラが未提供の場合は仕様とソースの二元管理である．このとき，

3)　仕様とソースの両方を検索するのが大変

という問題がある．

この問題の解決策として，リエンジニアリングで生成する下流から上流への情報の対応情報を検索システムで活用することが相乗効果を生むのではないかと考えた．

4．メンテナンスワークベンチの設計思想

前節のアプローチにより，ソースプログラムから手続き的仕様を自動的に逆生成するリバースエンジニアリングツールと，この逆生成時に得られる情報を活用して，WS画面で仕様書，プログラム，その他の文書を効率よく検索・修正できる開発・保守支援システム（仮称：メンテナンスワークベンチ）を設計した．

メンテナンスワークベンチのモデルは図1のとおりである．

「総合的に紙を越える保守効率性を得ること」という目標を実現するには，リエンジニアリングツール等種々のCASEツールを容易にスロットインしていける標準環境が必要である．この問題に対してメンテナンスワークベンチは，データ，ユーザインタフェース，タスクの各統合機構を設けて対応する．

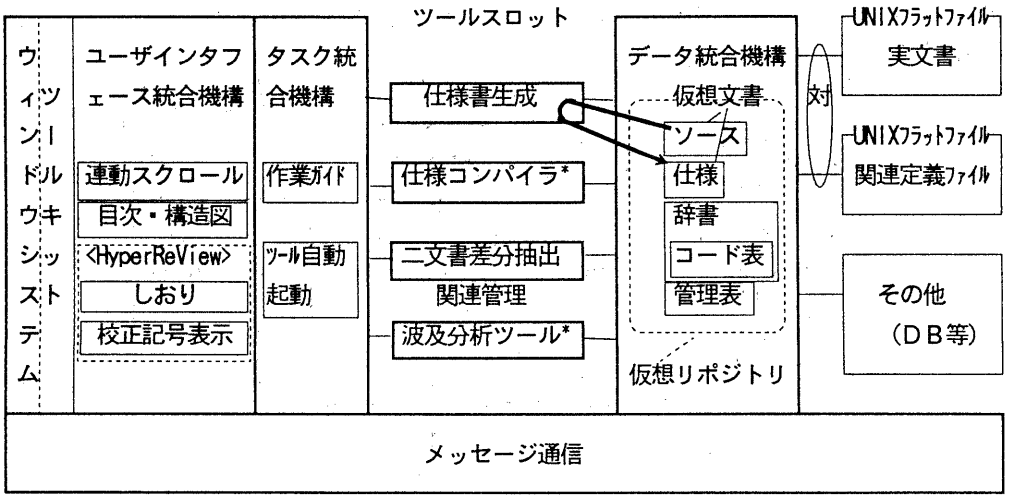

図1　メンテナンスワークベンチのモデル　　(*：将来)

5．メンテナンスワークベンチの実現

メンテナンスワークベンチのプロトタイプを，S-4，UNIX で試作した[2]．

5．1　仕様書生成ツール

仕様書生成ツール[3] は，英数字の読みにくい COBOLプログラムをコピーライブラリ（登録集）とともに入力すると，開発（発注）時の仕様書に近い要約度の仕様を逆生成する（**図2**）．辞書とコード表と変換規則を用意しておくことにより，完全自動で生成する．

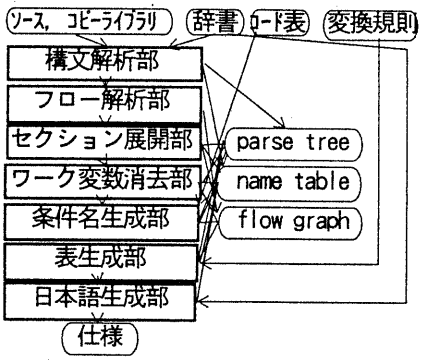

図2　仕様書生成ツールの構造

従来のリバースエンジニアリングツール「YPS ジェネレータ」に比べ要約度向上を図った（**図3**）．

図2のように，要約度向上の実現は，ワーク変数消去，条件名生成，表生成，日本語生成の各部に順次処理させることにより，各機能の独立発展性を確保した．

(1) ワーク変数消去による要約

プログラミングテクニックとして，ソースには多くのワーク変数（work variable）が定義され操作されているが，機能仕様の理解には細かすぎるといわれる．そこで，プロジェクトごとの標準化規約に基づいてワーク変数名を認識し，次にその使用法をコントロールフローとデータフローから解析する．次のような条件を満たす場合，ワーク変数の操作命令を仕様書に表さない．その代わり，そのほかの処理説明の中で，処理目的を表す．

1)　データの分解／合成の作業領域（図3）

2)　ファイル終端（end of file)検出時にセットし繰返しを終わるためのスイッチ

例：ソース「MOVE 0 TO SW-EOF
　　PERFORM ～ UNTIL SW-EOF = 1 …
　　READ～ AT END MOVE 1 TO SW-EOF～」
　仕様　「ファイルの終端まで～を行う．…
　　～を読み込む．～」

(2) 条件名生成による要約

プログラミングテクニックとして，変数の値または値の範囲それぞれに意味づけする方法がある．COBOL 言語は「変数　＝　値」などの条件式に条件名を与えることが可能であるが，これを使わず，（条件式・意味）の組を「コード表」として紙で

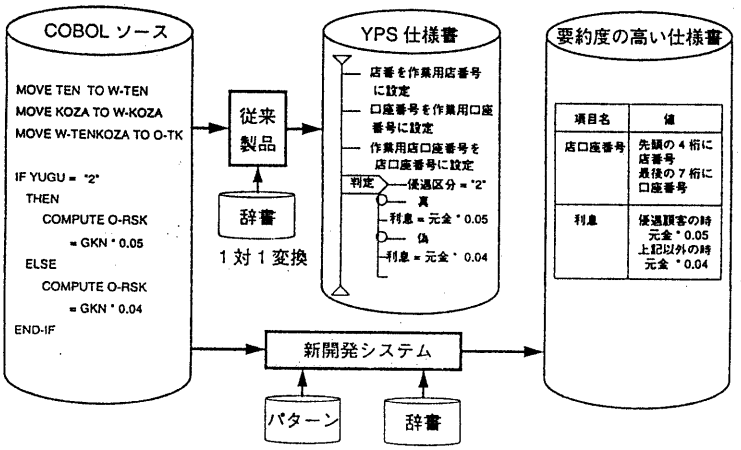

図3　要約度の高い仕様書の生成

維持しているプロジェクトが多い．そこで，あらかじめコード表をファイルとして用意しておけば本システムは「変数　＝　値」などの条件式をその意味に変換して仕様上に表す（例：図3の「優遇顧客」）．また，変数に値を代入するときに条件名の条件が成立するときに，意味を仕様上に表すことも可能である．

例：ｺｰﾄﾞ表「KIBO=1:中小企業,2:大企業」
　　ソース「MOVE 1 TO KIBO　…
　　　　IF KIBO = 2　～」
　　仕様　「中小企業とする　…
　　　　もし大企業なら　～」

(3) 表生成による要約

表生成部は，パターンマッチングにより，中間表現から要約された表現に変換する．この際，パターンのネストに対し再帰的に処理する．したがって，二つ以上の条件が組み合わされるときも自動的に大から小に細分化される階層的な表を生成できる．

また，ひとつの集団項目への代入を伴う命令が連続している場合などに，それらの変数名と値を左右にならべてひとつの表を生成する．モジュール呼出しの前処理として行われるパラメタ群のセットも，この機能によりひとつの表になることが多い．

表にならない部分も，パターンにマッチすれば要約表現の文章に変換され，パターン変数の値が文章に代入される．パターンにマッチしなくても，各命令構文に対応した日本語構文に変換されるので，サポートしている言語文法範囲内である限り，抜けのない正確な仕様書が生成される．

(4) 日本語生成

英数字名標を日本語名標に対応させる辞書が用意されていれば，自動的に変換される．

辞書がない場合には，ユーザが作成しなければならないが，YPS リエンジニアリングでは，COBOL 資産を分析，整理して英数字名標を洗い出し，人が対応する日本語名標を入力すれば辞書が作れるツールを提供している．

英数字名標の部分文字列が辞書にあれば，各要素を日本語名標に変換してから合成する．さらに 1) ローマ字の表記規則，2) 区切り文字の標準化規約，3) 末尾の枝番規約，4) 接頭語・接尾語，5) 連濁規則（～日→～BI等），6) 子音文字を抜き出す（CODE→CD等）等の略語規則，を用いて，限られた辞書からより多くの英数字名標が変換可能と考えている．半角カナ文字から漢字への変換も，同様の仕組みで可能と考えている．

辞書にない場合も，綴りの特徴からひらがな，カタカナ，英数字のままと，変換し分けることが可能である．したがって，辞書をもっていないプロジェクトでも，メーカ提供辞書だけで，多くの英数字名標が妥当な意味の日本語名標に自動変換できると期待される．

コメントの自動日本語化技術については，名標の自動日本語化と同様の論理で，小さい辞書によって漢字かな混じり文に変換するシステムをすでに実用化している．

5．2　データ統合機構

データ統合機構は，CASEツールの処理対象である仕様などの形式を標準化し，さらに，データの実際の形態がUNIXのフラットファイルやデータベースなどさまざまであっても，CASEツールに対して統一化して見せるインタフェースである．この機構により，新規CASEツールの接続性がよくなる．

(1) ハイパテキスト形式の仮想文書

文書要素が木構造をなし，リンクを自由に張れるハイパテキストは，構造をもった（非線型の）文書データベースである．ワードプロセッサを上回る検索効率を得る目的で，仕様書などの文書を，実際にはフラットファイルであってもハイパテキストでユーザに見せる設計を採用した．これが，データ統合機構が提供する仮想文書である．仮想文書と実文書との対応は，マッピング技法によって実現した．

(2) 関連管理

ハイパテキストのマッピング情報のほか，後述する連動スクロールおよびHyperReView のように，ドキュメントの内部および外部との関係情報を管理する必要がある．そこで，実文書と一対一にその関連定義ファイルというものを対応付け，仮想文書に必要な情報を保持している．

(3) 仮想リポジトリ

メンテナンスワークベンチにおいては，辞書のほかにソース，仕様，コード表，管理表などがあり，仮想的なリポジトリ（倉庫）を構成している．

5．3　ユーザインタフェース統合機構

メンテナンスワークベンチは，ホストアプリケーションの仕様書，プログラム等をWSに向かいながら統一的に操作できる快適な環境を提供する．

図4 に画面例を示す．

(1) 連動スクロール

関連文書間で一方向または両方向に連動スクロールモードを設定しておけば，文書の視点を動かすにつれて関連文書の表示も対応するように動くのが「連動スクロール機能」である．たとえば，ユーザがソースの表示ウィンドウでスクロール操作を行って視点を動かせば，そのプログラムの仕様書のウィンドウでは，対応する部分が点線枠で囲まれて画面に出てくる（図4）．仕様の方を動かせばソースが同様に動く．この実現のために，仕様書生成ツールが，ソースステートメントごとに生成仕様との対応関係を関連定義ファイルに出力する．この情報をユーザインタフェース統合機構が読んで連動させている．

このようにリエンジニアリング情報の活用は，人間に関係付けの負担をかけずに，検索のはやい環境が得られるので，よい方法と思われる．

(2) 目次・構造図

これも仕様書生成での解析結果を利用して，目次，モジュール構造図（図4）を自動作成する．表示上で要素をマウスクリックして，該当の仕様部分やモジュール関連文書が検索できる．

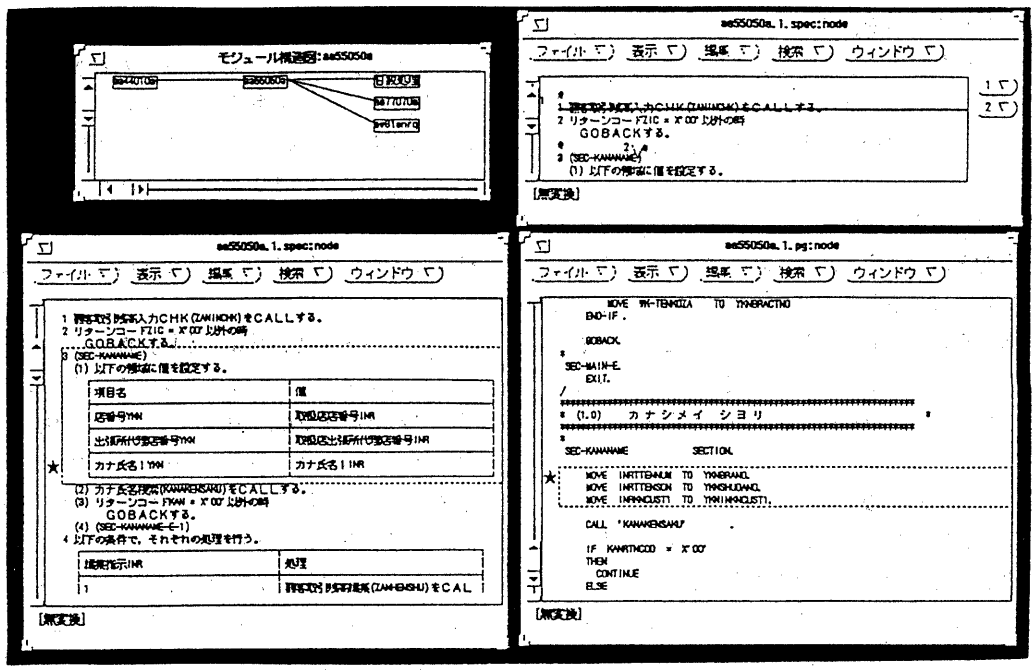

図4　メンテナンスワークベンチの画面例

(3) HyperReView

複数文書にわたる修正箇所の関連管理機能と，レビュー支援機能の総称をHyperReView [4] と名付けた．しおりで柔軟な情報付加を可能とし，検索効率向上を図った．

仕様，ソースなど一般の文書において，要注意部分や修正予定部分や修正完了部分にしおりを貼り付ける操作を行うと，そのしおり情報が関連定義ファイル中に記憶される．以後，文書表示ウィンドウにボタンでしおりの番号が表示され（図4），クリックするとその部分が表示されるので，「頭出し」ができる．しおりには順序をつけたりグループ化したり変更できるので，レビュー者が執筆者の意図した順序で，修正案とコメントが読める．したがって，紙の仕様書を持参したり，電話で説明しなくても，電子メールで仮想文書を送るだけで効率的なレビューができる．レビュー結果もしおりで記録できる．さらに，しおりに修正の時刻，場所，人，修正／案件名，理由，方法など5W1H情報の大部分を自動的に記録することも可能である．

また，HyperReView の校正記号表示機能は，利用者が紙の感覚で修正履歴が一覧できるよう，修正が赤い校正記号で表示される機能である．利用者の操作は，削除や挿入のキー操作であるが，修正結果にしたがって，削除は字の上二重線，挿入は文字列および挿入位置を示すスプライン曲線，移動は移動元と先を結ぶ矢印などで表示される．

文字列操作は一定の距離内，一定文字数内といった制限をつけることによって，校正記号で表示して見やすくなる範囲の処理にとどめている．

エディタで修正操作を行うとしおりが生成されるので，申し送り事項の書き込みを忘れにくい．

5．4　タスク統合機構

開発・保守の効率化のためには，個人作業においては，定型作業手順のメニュー化・自動化が有効である．これを実現するために，作業者がツールメニューではなく作業メニューをクリックすれば，システムが状態に応じて適切なツールの自動起動を行うようにした．

さらにグループ作業においても，依頼／報告，申請／承認等からなる定型プロセスがあって伝票処理，打合せが多いので，プロセスの自動化が有効と考え，その機構を開発した．

たとえば，仕様の修正が完了すると，システムがレビュー担当者を調べてレビュー依頼書・仕様・ソースを電子メールで発送する．この際，レビュー依頼書のような定型文書のほとんどの欄が自動記入可能であることが分かった．レビュー担当者が作業メニューを開くと，レビュー作業が追加されている．このように作業ガイドされるという形で，時空間の制約に縛られない快適なグループ作業が実現する．

6．運用と期待効果

メンテナンスワークベンチを用いた保守工程の運用パターンを示す（**図5**）．

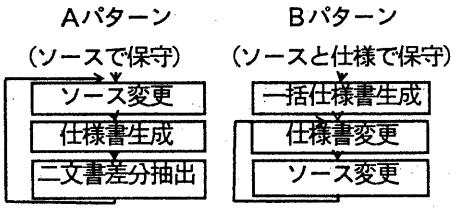

図5　運用パターン

Aパターンは，「ソースで保守」というパターンであり，プログラムを修正するたびに仕様書を自動生成する．旧版との差を調べるために「二文書差分抽出ツール」を開発したので，仕様書の連続性がないためのチェックの手数をカバーできる．この場合も差分は校正記号で表示され，しおりも自動的につくので，申し送り事項が記入しやすい．

Bパターンは「ソースと仕様で保守」という形である．この運用形態に移行する際に，対象プログラム群から一回だけ一括して仕様書を生成し，以後，ソースと仕様書を保守していく形である．紙での運用に近いので移行が容易である．この場

合，設計者が仕様を変更してしおりをつけて発注するとか，開発者が仕様変更をしおりで提案して設計者が承認するとか，設計者が開発者の変更案を参考に仕様の正本を保守するなど，いろいろな運用形態を支援することができる．

もしA，B以外の「仕様だけで保守」というパターンが実現すれば保守効率性は最大になるが，仕様コンパイラがない段階ではA，Bどちらかを選択しなければならない．

さて，メンテナンスワークベンチには，「プログラム検収支援」という活用法もあることが分かった．すなわち，発注者は納品されたソースから分かりやすい仕様書を生成し，旧版の仕様書があれば差分を表示してみるとよい．これにより，発注者の思いどおりにできているかどうか，標準化規約が破られていないか，修正要件に関係ない部分が壊されていないか，不正な目的のコーディングがされていないか，ステップが冗長に増やされていないか，といったことが，より簡単にチェックできる．

メンテナンスワークベンチを利用すると，ホストコンピュータに負荷をかけずに，WSで自由に仕様やプログラムを検索・修正することができる．このため，中規模から大規模のソフトウェアの開発部門に最適な，低コストかつ高信頼な分散開発環境が実現できると期待される．

7．評価

実際のパッケージソフトウェアを使って，メンテナンスワークベンチの機能，性能，操作性の評価実験を保守担当SEチームと実施している．

現在までの結果では，仕様書生成について実験した範囲で正確性を確認できた．また，生成仕様の抽象度について，現在の仕様書より分かりやすいものもあるという評価を得た．しかし，業務の流れを把握するのに現在の仕様書の2～3倍の時間がかかったという指摘があり，さらに要約機能の強化が必要ということが明らかになった．

仕様書生成の性能評価では，コピー句展開後のCOBOL 5,000行のソースを約60秒で仕様書に変換することができ（S-4/2使用）実用的速度を達成した．

文書統合機能の機能，操作性，レスポンスは概ね好評であった．

メンテナンスワークベンチの導入によって，総合的な保守効率性がフラットファイルのワードプロセッサを越えるのは確実であるが，紙を越えることができるかどうかは，実際に運用して検証しなければならない．

8．まとめ

開発・保守を支援するために，仕様電子化と，それを生かすツール・環境構築をどのように行えばよいかということを，メンテナンスワークベンチの試作を通じて明らかにした．

第一に，仕様の効率的電子化のために，ワーク変数消去，条件名生成，表生成，日本語生成の技法を開発して，ソースから仕様を逆生成するツールの要約度向上に成功した．今後も「プログラムの開発（発注）時に書かれるプログラム仕様書に近い要約度」という目標に向かって，より広域的に分散した仕様情報をソースから抽出する研究を進める．

第二に，電子化仕様を利用するツール・環境のコンセプトを作り，プロトタイプを試作した．ハイパテキスト，しおりといった非線型性を導入し，また，連動スクロール，目次・構造図生成などリエンジニアリングで生成される下流から上流への対応情報を効果的に活用することで，検索効率，修正履歴一覧性の問題をある程度解決できることが分かった．

今後「総合的に紙を越える保守効率性を得る」という目標に向かって，仕様コンパイラ，波及分析ツールなどのCASEツールの充実と，CASE環境の高度化を図る．特に，タスク統合技術のソフトウェアプロセス[5][6]を応用し，開発プロセス[7]を計算機支援することにより，プロジェクトから個人にいたるまでの開発・保守の質的改善を目指したい．

参 考 文 献

[1] Carma McClure: Three Rs of Software Automation, Prentice Hall (1992).

[2] 上原三八, 園部正幸, 吉野利明, 直田繁樹, 石崎あゆみ, 川辺敬子: プログラムと仕様書の統合管理による開発保守支援システム, 情報処理学会ソフトウェア工学研究会, 87-2 (1992.9).

[3] 吉野利明, 上原三八, 直田繁樹, 野呂正明, 大久保隆夫: 事務処理プログラムからの仕様抽出, 人工知能学会第6回全国大会, D1-1.4 (1992.6).

[4] 川辺敬子, 上原三八, 吉野利明, 石崎あゆみ: 分散環境におけるレビュー支援, 情報処理学会第45回全国大会, 1U-9 (1992.10).

[5] Leon Osterweil: Software processes are software too., Proc. of 9th ICSE (1987).

[6] Peiwei Mi and Walt Scacchi: Process Integration in CASE Environments, IEEE Software, Vol. 9, No.2, pp.45-53 (1992). 同書評: 山本枝里子: 92-29 CASE環境におけるプロセス統合, 情報処理, Vol.33, No.9, pp.1099-1100 (1992).

[7] 村上憲稔: ソフトウェアのライフサイクル管理, 情報処理, Vol.33, No.8, pp.912-921 (1992).