情報システム 42－1
（1993. 1. 19）

ソフトウェア再エンジニアリングの技術の動向とその要件

竹下 亨
中部大学経営情報学部

再エンジニアリングとは、「対象システムを調べて、新しい形態に変形し、それを実現すること」であり、「再構造化、逆エンジニアリングおよびその結果行われるフォワードエンジニアリングによるシステム再生成を包含する総括的技術」である。近年になってこの分野の技術が注目され、各種の研究・開発が行われて、若干の成果を上げつつある。これらの発展の背景を技術とニーズの両面から考察し、その中で、プログラム理解、逆エンジニアリング、再構造化を取り上げ、現時点で実用化されているもの、もしくはそれに近い技術を検討する。その結果として、再エンジニアリングのツールの機能として実現化可能な機能を纏める。そして、それらが具体的にどのように実現されているかを確認するために、現在使用可能なツールの例をサーベイする。最後に再エンジニアリングの技術に、さらに何が要求されているかを論じ、今後の見通しとする。

Software Reengineering Trends and Requirements

Toru Takeshita
College of Business Administration and Information Science, Chubu University

Re-engineering means 'to analyze an existing system, to transform it into a new form, and to implement the new system,' or 'a comprehensive technology encompassing system re-structuring, reverse engineering, and system re-creation as a result of forward engineering.' Various software re-engineering technologies have recently been developed to achieve some success. These progresses are reviewed both from the technological and user needs aspects. Then, focus is placed on program understanding, reverse engineering, and restrucuring technologies which have been more or less put into practical use, though some of them are not well proven yet in their effectiveness. Then, the paper takes a look at those functions which can be developed as as (CASE) tool functions, and surveys those re-engineering tools which are available in the marketplace with a few exceptions. Lastly it lists up the further requirements to be met by these tools and their future versions, which are the author's perspective in this technical field.

1　まえがき

１９９０年に「ＣＡＳＥ概論」を出版したが、その本の執筆中にＣＡＳＥ環境の普及と活用のためには、再エンジニアリングと再利用の技術を開発し、実用化して、ＣＡＳＥ環境に組み込む重要性を痛感し、それらをサーベイして、「ソフトウェアの保守・再開発と再利用」という小著をまとめた。

本研究は上記をベースとして、再エンジニアリング技術の発展の背景を分析し、主要な技術を追求し、ツールに対する要件を纏め、将来方向を見定めることにした。

最初に、関連する重要用語の定義を記述し、それらの技術の背景を基本技術とユーザーニーズの両面から捉え、各種の技術がどのように進化・発展したかを整理した。

つぎに、プログラム理解、逆エンジニアリング、再構造化の三つの分野の技法に焦点を当てて、それらを考察する。清書的技法を含め、プログラムを見やすくする工夫、再文書化、数学的手法を用いた再構造化，ＧＵＩによる図式かつマルチビューなどが特徴となっている。

そして、ツール化可能な機能を取り上げ、それらが具体的にどのように実現化されているかを見るために、現在利用可能なツールをサーベイする。すでに、２０種を越える製品が出荷されている。

最後に、これら再エンジニアリングのツールにさらに要求されていることを検討し、それらを満たすべく、Ｒ＆Ｄが継続されるとして、今後の見通しとする。

2　ソフトウェアの保守・再開発関連の定義

ソフトウェアの保守とは、ソフトウェアの出荷後の欠陥の除去、性能や他の属性の向上のための修正、環境の変化に適応するための修正などを意味している。

ＩＥＥ　Ｓｏｆｔｗａｒｅ誌１９９０年１月号に掲載されたChikofskyとCross IIの定義によれば、ソフトウェアの保守(maintenance)は、つぎを含むとしている。

逆エンジニアリング
再構造化
再エンジニアリング

逆エンジニアリング(reverse engineering)とは、システム構成要素の解明を行い、他の形態か高水準のシステム表現を生成することである。そのなかには、再文書化(re-documentation)、設計復元(design recovery)、関数抽象化(function abstraction)、業務規則(business rules)の抽出・表現が含まれる。

再構造化(restructuring)とは、抽象度を変えないで、ある表現から他の表現に変形することである。

再エンジニアリング(re-engineering)は、既存システムの分析・理解を行い、それを新しい形態に変形し、その結果を実現することである。

ChikofskyとCross II（ＩＥＥＥ　Ｓｏｆｔｗａｒｅ誌１９９０年１月号）の定義によると、それは「再構造化、逆エンジニアリング、およびその結果行われるフォワードエンジニアリングによるシステム再生成を包括する総括的技術」である。

ＧＵＩＤＥ（ＩＢＭのメインフレームのユーザーの団体）はその定義として、つぎの三つをあげている。

1）　コードを一層構造化して、複数のプログラムに利用するのに共通的に見やすくする助けとなり、既存のソフトウェア・システムの寿命を延ばす方法。

2）　既存の機能、技術基盤やアーキテクチャに影響を及ぼすことなく、現在のシステムを改良する手段。

3）　逆エンジニアリングのためにシステムを準備する方法。

3　再エンジニアリング技術の発展の背景

ソフトウェア再エンジニアリングに関連する研究・開発は１９８０年代の中ごろから活発になり、万能薬的なものからはほど遠いが、ある程度実用化しているものもある。

発展を促した要因を技術面から捉えると、次のような技術の進歩をあげることができる。

1）　デバッグとのテストに関連した手法・技法

2）　テキスト情報の編集・清書技術

3）　コンパイラやジェネレータの構文解析、静的意味解析技術

4）　構造化プログラミングの理論と実際

5）　グラフィカルインタフェース、マルチウインドウなど

6）　データベース、リポジトリの技術

7）　増大する規模と複雑さを制御する工学的技法（分割統治）

発展を促した要因のユーザニーズ面には、下記が含まれる。

1） ソフトウェアの保守量の増大

2） 理解・修正が困難なコードの存在

3） 不十分な文書化と標準化

4） 保守コストの増大が新規開発を阻害

5） 新しい開発環境（ＣＡＳＥ）への移行の必要性

6） ビジネスの再開発を短期間に低コストで

7） 標準化を含む再文書化の必要性の認識

4 再エンジニアリング技術の展開

上記のような背景から様々な再エンジニアリングの技術が進化・発展してきた。手法や技法を識別し、それらがどのように進歩・発展して来たかを考察し、下記に整理してみた。

1） 定義、使用、流れ、対応や関連付けなどの分析－＞上位レベルの情報の選択・抽出

2） テキストベースの解析・表現－＞制御やデータの流れ、データ構造をグラフィックベースで表現、可視化

3） 非数学的・非形式的解析手法－＞数学的・形式的、言語学的解析手法

4） 制御の流れが中心－＞データにも着目

5） 定義一覧表、関連図表、相互参照表など－＞影響分析、検証やテストを支援する情報

6） 同一（ソースコード）レベルの再文書化－＞上位抽象レベルの仕様（－＞設計の復元）

7） あるがままの解析－＞再構造化して理解と修正の容易度の向上

8） ソースコード中心の解析－＞ファイルやデータベースの解析

9） プログラムと直接処理対象の解析－＞アプリケーション全体の調査・分析

10） 単一のビュウー－＞マルチビュウー（同一や異なるレベル間）

11） 追跡・走査のための煩雑なコマンド入力と目視による識別－＞ナビゲーション、ハイパーテキストなど

12） 単独に使われる単体ツール－＞共通のリポジトリやユーザーインタフェースにより統合化

13） 特定言語、技法に限定－＞複数言語、技法に適用、カスタマイズも多少可能

14） 逆エンジニアリングのみ－＞フォワードエンジニアリングに直結

15） 品質評価基準が不在－＞数値的評価基準（ソフトウエアメトリックス）

16） 特化、固有、私的技術－＞汎用、一般化、商品化技術

4 プログラムの理解

保守のコスト上昇の原因として、つぎのようなことがあげられる。

1） プログラムの変更によるエラーは微妙であり、それを修正するコストはさらに大きい。

2） 影響分析が不十分のままでプログラムの変更は、設計を軟弱にし、頑健さ(robustness)を壊し、後で必要となる保守のコストを押し上げる。

そこで、保守作業の中で、プログラムの理解が極めて重要であり、また困難であり、従来から保守作業の労力・時間の約５０％を費やすといわれている。

プログラムの理解(program understanding)とは、「コードから情報を抽出し、設計の抽象的概念を再現するための、組織化されたプロセス」である。

抽象的概念を再現する工程により、つぎのことが可能である。

1） 逐次的に高いレベルのプログラミングの概念を得る。

2） プログラム領域(domain)の概念を問題領域内の概念に対応付ける。

プログラムの理解の方法として、つぎの三つがある。

1） トップダウンのやり方
仮説を立てて確認

2） ボトムアップのやり方

プログラムの数学的仕様の正しさの証明(correctness proof)

3） 依存関係を辿りプログラマの目論見(plan)を判別

プログラム理解のための検討対象として、各種の依存（従属）関係がある。

1） データ項目の依存関係

2） 副プログラムの依存関係

3） ソースファイルの依存関係

4） その他の構造部分の依存関係

そのための技法やツールとして、つぎがあげられる。

1） 相互参照(cross reference)

2） 連鎖を示すグラフや木構造チャート、階層チャート

3） プログラムスライシング

4） 漣（さざなみ）効果分析

ソフトウェアの理解を容易にする強力な手段として、様々なプログラムの視覚化(visualization)が行われている。下記はその例である。

1） 注釈付きのプログラムリスト
注釈付き、標準の書式に編成された相互参照リスト

2） 制約付きの照会
欲しい情報だけに限定して検索

3） 複数ウインドウのブラウザ
関連情報やソースコードのある区域を他のウインドウから

4） クラスタ化技法
幾つかの構成部分を高いレベルに纏める

5） 格子の仕組み(lattice mechanism)

部分(part)間の相互反応の可能性を纏めたシステム構造の図的表現

5 ソフトウェアの逆エンジニアリング

ＧＵＩＤＥの定義によると、逆エンジニアリングは、「データ記述とプログラム論理を実現に依存しない形で抽出し、標準化・文書化し、自動化ソフトウェアエンジニアリング環境に移行する工程」である。

逆エンジニアリングは対象システムを分析して、下記を行う。

1） システムの構成要素とそれらの間の関係を明確化。

2） 他の形態もしくは抽象度の高いレベルでシステムを表現。

逆エンジニアリングの主要な作業は、つぎの二つである。

1） 再文書化

2） 設計復元
Ted Biggerstaffは「ソースコード、（もし存在すれば）設計文書、個人的経験、問題やアプリケーション領域(domain)に関する一般的知識の組み合わせから、設計の抽象概念を再生することであり、．．．．」と定義している。

逆エンジニアリングの主目的として、つぎがあげられる。

1） 複雑性への対応

2） 別の姿の生成

3） 失った情報の復元

4） 副作用の検出

5） 高水準抽象概念の合成

6） 再利用の容易化

ソフトウェア逆エンジニアリングの最も重要な成果物として、再文書化がある。再文書化の技法として、つぎのようなものがある。

1）　ソースコードの清書
　　字下げをする。
　　頭を揃える。
　　字体や色を変える。

2）　相互参照表
　　定義箇所と使用箇所、
　　名札の出現箇所など。

3）　設計仕様チャートや図の生成
　　木構造の詳細設計仕様、
　　制御流れ図、
　　データ流れ図、
　　データ構造図、
　　プログラム構造チャート。

ソースプログラムの清書の方法の一つに本のパラダイムがあるが、その利点はつぎのとおりである。

1）　日常見慣れている文書の形態は認識しやすい。

2）　高いレベルでは、編集上の手がかりが容易に判別。

3）　低いレベルでは、編集上の塊(chunk)や目印が容易に判別。

4）　目次や索引が複数の参照経路となる。

逆エンジニアリングプロセスには、つぎの４段階がある。

1）　対象とするアプリケーションの選択と準備
　　企業の中心的データを処理する度合いと保守の頻度とコスト
　　共通なデータ項目の定義を情報モデルから選択
　　シソーラスの作成（同義語、異義語、略語の解決）

2）　ソースコードの分析と設計情報の抽出
　　構文解析や再構造化、プログラムの制御構造やデータ構造、
　　データの流れなどの分析と抽出
　　重複や死コードの検出

3）　標準化と文書化

検索と更新の支援機能が必要

4） 抽出した情報をリポジトリに格納

逆エンジニアリングの効果の最大化には、つぎのようなことが重要と考えられる。

1） ツール関連の制御を一点に集中化

2） 逆エンジニアリングの成果をリポジトリに格納

3） 修正・拡張用のツールにそのまま入力

4） 再利用可能な仕様の共通の集合を共用

5） 抽出された仕様を重複なしに維持
フォワードエンジニアリングツールと共用のスキーマ

6 ソフトウェアの再構造化

ソフトウェア再構造化とは、ある表現から相対的に同一な抽象化レベルの他の表現へ変換することである。対象システムの外部的振る舞い（機能や意味規則）は不変である。

プログラムの再構造化変形はソースプログラムを構造化設計や構造化プログラミングの形態に変形することである。

再構造化の範囲は、スパゲッチコードから構造化コードへの変形に限定されずに、データモデル、設計計画、要件仕様構成の作り直しなども含まれる。

ソフトウェアの再構造化は予防保守とも見なせる。

構造化の理論的基礎は、Ｌｉｎｇｅｒ，Ｍｉｌｌｓ，Ｗｉｔｔの構造理論で、任意の流れ図は３種の制御構造から構成されることせある。

1） 連接(sequence)　：Ｂｅｇｉｎ－Ｅｎｄ

2） 条件(alternation)　：Ｉｆ－ＴｈｅｎやＩｆ－Ｔｈｅｎ－Ｅｌｓｅ

3） 繰り返し(iteration)：Ｗｈｉｌｅ－Ｄｏ，Ｒｅｐｅａｔ－Ｕｎｔｉｌなど

プログラムを構造化するツールの機能（例）には、下記がある。

1） 複雑な制御の流れを単純化

2） 整合性のとれた形態（書式）で表示（印刷）

3） ソースコードの複雑性の分析

4） 構造化されたプログラムの構造チャートの生成

5） 各段落にコメントを付けて、それに達するすべての可能性の表示

6） ソースコードのモジュール化を提案

構造化されたＣＯＢＯＬプログラムはつぎのような形態になっている。

1） プログラムはＰＥＲＦＯＲＭされる手続きのトップダウン階層。
下記は存在しない。
- 手続きの間にＧＯＴＯ文やＡＬＴＥＲ文
- ＰＥＲＦＯＲＭ手続き名－１ＴＨＲＵ手続き名－２
- ＰＥＲＦＯＲＭの重複
- ＰＥＲＦＯＲＭの再帰

2） 手続きは一つの入口と一つの出口をもつ。
３種の基本構造により構成
- 連接、条件、繰り返し

3） 手続きの中に次の文はない。
- ＧＯＴＯ文
- ＧＯＴＯ ＤＥＰＥＮＤＩＮＧ文

7 再エンジニアリングツールの機能

再エンジニアリングツールの機能を整理するとつぎのようになる。

1） ソースコードのレベルで見やすくする機能
見出しや字下げ、章節と索引、使用箇所の表示、制御の流れの表示、細部などの除去、有効範囲の表示、定義・使用箇所の表示、データ代入やファイル入出力の一覧表、使われてないルーチンやデータの

一覧表など

2） 詳細設計レベルの情報の抽出
設計言語や擬コードで表現、呼出関係を表示、制御流れ図、
データ構造図、データモデル図、構造チャート、データの流れ
解析／追跡、階層・継承関係、ＤＢの概念・論理仕様など

3） コードの再構造化
ソースコードを構造化プログラムへ変換

4） 高水準設計仕様や業務規則の抽出
システム流れ図、データ流れ図、関数抽象化、自然文に近い形で
表現など

5） 影響分析
プログラムの中の要素を変更したときに影響を受ける可能性の箇所、
アプリケーションシステムの中の要素を変更したときに影響を受ける
可能性のあるもの

6） その他の機能
ヘルプ、注記入、ＧＵＩ、マルチビュー、ナビゲーション、
ハイパーテキスト、再利用可能モジュールの生成、リポジトリへの
格納、ソフトウェア品質メトリックスなど

なお、データに着目して、プログラムを部分的に再構成する技法として、データの再エンジニアリングが提案されている。これはデータ使用抽象化(data usage abstraction)と箱構造抽象化(box structure abstruction)の理論を組み合わせたものであり、つぎのようなことに使用される。

1） データの流れの変則(anomaly)の除去

2） データの通用範囲の縮小

3） 共通サービスとしての再利用可能なデータオブジェクトの形成

箱構造は情報システムの分析と設計用の理論や方法論で、データに着目して、３種類の箱がソフトウェアの構成体を表現するのに使われている。

1） 黒箱： 刺激の履歴から反応を生ずる変遷において、外部振る舞いだけでデータの抽象化を定義

2）　状態箱：　　刺激と内部状態から反応と新しい内部状態への変遷でデータ抽象化を定義

3）　クリア箱：　内部状態をアクセスし、分割の次のレベルの黒箱を呼び出すことのある手続きを定義

データの変則とは、次のことをいう。

1）　初期化されてないデータのｒｅａｄ
ｗｒｉｔｅに先行されてないでｒｅａｄされるデータ項目

2）　使われてないデータのｗｒｉｔｅ
データ項目がｗｒｉｔｅされ、次の値がｗｒｉｔｅされる前に新しい値がｒｅａｄされてない

3）　オーバーロードのデータ項目
二つのｒｅａｄの間に繰り返しｗｒｉｔｅされるデータ項目

4）　その他のデータの流れの変則
許されていないデータの使用など

8　再エンジニアリングのツール

再エンジニアリングのツールにはつぎのようなものがある。（順序不同）

1）ＲＥＣＯＤＥＲ
１９８５年に初版、Language Technology 社(KnowledgeWare社に吸収)
ＣＯＢＯＬプログラムを機能的等価な構造化プログラムに変換
ソースコード－＞抽象的構文木－＞有向グラフ－＞
３種の構成体の入れ子の組み合わせ－＞
新しい抽象的構文木－＞最終的抽象構文木－＞ソースコード

2）ＩＮＳＰＥＣＴＯＲ
１９８５年（？）に初版、Language Technology 社
ソフトウェアメトリックスを出力

3）ＣＯＢＯＬ／ＳＦ
１９８６年に初版、ＩＢＭ社
ＯＳ／ＶＳ　ＣＯＢＯＬとＶＳ　ＣＯＢＯＬ　ＩＩを構造化
不適切なＰＥＲＦＯＲＭステートメントを自動的に修正
ＰＥＲＦＯＲＭ階層構造の図式表示
細分化や再開発を助けるモジュール化分析

4) Ｃｈｒｙｓａｌｉｓ　ＰＭ
１９９０年に初版，Language Technology社
ＩＢＭのＯＳ／２とインタフェースし、ＡＤ／Ｃｙｃｌｅで稼働
構造化されたＣＯＢＯＬプログラムと構造チャート、制御流れ図
３種類の複雑性メトリックスを出力

5) Ｅｘｃｅｌｅｒａｔｏｒ　ｆｏｒ　Ｄｅｓｉｇｎ　Ｒｅｃｏｖｅｒｙ
１９９０年に初版，Index Technology社(合併によりInterSolv社)
ＣＯＢＯＬのソースコードから構造チャート
ＩＭＳのＤＢＤからデータモデル図
データの部などからデータ定義
ＩＭＳ／ＭＦＳのメッセージやＣＩＣＳ／ＤＭＳから画面の設計
参照されていない段落の一覧表
参照されていないデータの一覧表
データ項目代入の一覧表
ファイル入出力の一覧表
巡回複雑性

6) Ｖｉａ／Ｃｅｎｔｅｒ
１９９０年（？）に初版，ViaSoft社
知的エディタと組み合わせた対話的経路分析、論理分析、文書化
プログラムに含まれるオブジェクトの定義や関連をＡＫＲに格納
対話的テストを支援

7) Ｖｉａ／Ｒｅｎａｉｓｓａｎｃｅ
１９９０年（？）に初版，ViaSoft社
プログラムを解析して単一の業務規則やタスクを行うモジュールに分離
関連する論理とデータを分離して再利用可能なモジュールの生成

8) ＹＰＳリエンジニアリング
１９９１年に初版、富士通（株）
不要や欠落のモジュールを識別
日本語に変換するため字引を作成
英語のＣＯＢＯＬソースコードから日本語のＹＰＳ仕様を生成
ＹＰＳ仕様上で効率的な保守作業

9) ＲＥ／Ｃｙｃｌｅ
１９９１年に初版、ＩＢＭ東京基礎研究所の研究プロジェクト
ＣＯＢＯＬ、ＰＬ／ＩおよびＪＣＬ

システム流れ図
再構造化
構造チャート
データカップル
アクションダイアグラム
データ構造チャート
影響分析
ハイパーテキスト風インタフェース

10) SMARTsystem
PROCASE社 （エーエスアール（株））
オブジェクト指向DBを使うC用の保守と再エンジニアリング
a) SMARTview： テキストベースの解析
b) SMARTcheck： 文法と意味の漸増的チェック
c) SMARTstore： オブジェクト指向DB
d) SMARTgraph： プログラム構造を視覚化
e) SMARTmake： 漸増的make機能とシンボリックデバッグ
f) SMARTreport： ソースプログラムの大きさや複雑性

11) Software Refinery
Reasoning Systems社 （（株）SRA）
a) DIALECT： 構文解析、結果を知識ベースに格納
b) REFINE： オブジェクト指向データベース
c) REFINE言語： オブジェクト処理用、プログラム変換に使用
d) INTERVISTA：知識ベースの内容の図形表示

11) REFINE/COBOL
Reasoning Systems社 （（株）SRA社）
Software Refineryで開発
ANSI COBOL 74のソースコードを構文解析
プログラム構造図
制御流れ図
データ項目の相互参照表
PostScriptプリンタ印刷
プログラム分割とリンケージ節の自動生成
Software through Pictures (StP) へ
プログラム構造図データを移送

12) FOUNDATION
Andersen Consulting （アンダーセンコンサルティング）
ＣＯＢＯＬ／ＳＲＷ(COBOL/Software Reverse Workbench)
ＢＡＬ／ＳＲＷ(Basic Assembler Language/Software Reverse Workbench)
ＣＯＢＯＬやアセンブラのソースプログラムとＪＣＬ文を入力として、
内部構造の抽出、分析、構造チャートを作り、リポジトリに格納

13) ＡＳ／ＳＥＴ(AS/400 Software Engineering Technology) Ｒｅｖ
１９９１年に初版、ＳＳＡ社(System Software Associates, Inc.)
ＲＰＧのソースコードを分析して、データモデル、画面と帳票の書式、
アクションダイアグラム

14) ＰＵＮＤＩＴ(Program UNDerstanding Investigation Tool)
１９９０年に初版、IBMT.J. Watson Research Center での研究
プロジェクト
ソースコードを分析した結果を格納するデータベース
表示する機能 － ＮＡＲＣ(Nodes and Arcs)

15) ＦａｓｔＲｅｆ／２
１９９１年に初版、ＩＢＭ社
プログラムに関連する開発・保守資源に関する情報をリポジトリに格納
ための情報モデルと表示するツール

16) ＲＥＳＣＵＥ(Reverse Engineering System for Comfortable User's Environment)
１９９１年に初版、ケース・テクノロジー（株）
a）静的解析機能： ソースプログラム、ＪＣＬ、ＤＢ定義、
画面定義，ＣＯＰＹ定義、ＳＹＳ定義の管理
台帳を作成し、相互関連から影響分析
b） 動的解析機能： 制御の流れやデータの値の変化を追跡
c） 品質保証機能： ソースコードを解析してテストケースを抽出、
保守前後のテスト結果を比較

17) Ｔｅａｍｗｏｒｋ／Ｃ Ｒｅｖ
１９９１年に初版、Cadre Technologies Inc. （（株）東陽テクニカ）
Ｃのソースコードから構造チャート

18) Ｈｉｎｄｓｉｇｈｔ
１９９１年に初版。Advanced Software Automation社
（（株）エイ・エス・エイ、セイコー電子工業（株）など）

ソースコードから構造図と論理の流れ図
プログラムの複雑度
各種静的解析情報

19) PAC(Program Analyzer for C)
(株) エリック
関数のモジュール木、関数・変数の相互参照

20) FORTRANソースコード静的解析プログラム
横河ディジタルコンピュータ (株)
文数、行数、文の種類、プログラムの構成図、
親子一覧表、COMMONの名前と長さの一覧表

9 データベースの再エンジニアリング

RDBの構築と再エンジニアリングを図示すると、つぎのようになる。

1) 実体、関連、属性－＞実体関連図
　　|　　^
　　v　　|
2) 実体関連図－＞論理および物理モデルへ変換
　　|　　^
　　v　　|
3) 物理モデル－＞DDL(Data Base Defintion Language)

データベースの再エンジニアリングツールの例を下にあげる。

BACHMAN／Re-Engineering Product Set
1) データベースアドミニストレータ
論理／物理DBの設計と文書化、DDL－＞論理／物理図、
DB設計－＞実体関連図など
2) カタログ抽出
情報の抽出、ファイルの生成、IDによるDBアクセスの制御など
3) データアナリスト
情報モデルの文書化、新規設計、保守およびIMS，VSAMからの移行
4) アナリストキャプチャ
COBOLデータ記述、IMSのDB記述－＞情報モデル、
IMS－＞DB2，
IMSおよびCOBOLデータと情報モデルデータとの関係を管理
5) ワークステーションマネージャ

ＧＵＩの提供

Ｔｅａｍｗｏｒｋ／ＤＢ　Ｄｅｓｉｇｎｅｒ
１９９１年（？）に初版、Ｃａｄｒｅ社　（（株）東陽テクニカ）
１）　ＤＢファイル　＋　ＲＤＢカタログ　＋　ＥＲ図－＞正規化ＲＤＢ設計
２）　非ＲＤＢ－＞逆エンジニアリング－＞ＲＤＢ
３）　データ表やデータカタログを解析－＞冗長データや重複フィールドを検出
　　－＞３次の正規化－＞効率的、高性能ＤＢ
４）　サンプルデータによる設計の確認
５）　操作しやすいＧＵＩ
６）　ＳＱＬのデータ定義文を自動生成

ＤＢｐｒｏｍｐｔ
１９９２年学会発表、ＮＴＴソフトウェア研究所
　　情報資源ディクショナリと支援ツール
１）　ＲＴＲＶ：　ＤＢ設計シートからＤＢ設計情報を抽出
２）　ＮＡＭＥ：　データ項目の名前をチェックして、同義語と異義語を除き、標準化
３）　ＡＮＬＺ：　概念設計
４）　ＤＳＧＮ：　論理設計
５）　ＧＮＴＬ：　物理設計

１０　ソフトウェア再エンジニアリングツールの要件と展望

これまでに利用可能になった再エンジニアリングのツールは、まだ完全ではなく、多くの改良の余地があると考えられる。ユーザのニーズと技術的可能性からつぎのような要件があげられる。また、これらが満たされたツールがそう遠くない将来に出現することを期待している。

１）　データ項目名（変数名）、名札等の統一化、標準化、業務用語への対応

２）　再利用可能な上位仕様の生成、保存、検索

３）　コンパイラやジェネレータによる解析結果を再エンジニアリングに活用

４）　構造、制御とデータの流れの可視化、マルチビュー、動画化、ハイパーメディア化

５）　関連・影響分析結果の動的表示

６）　上位（下位）の変更を下位（上位）に自動的に反映

７）　設計の意図や決定理由などの復元

8） 究極的には業務規則(business rule)の抽出

9） 成果物をリポジトリに格納して他のツールに利用可能

10） フォワードエンジニアリングに直結

11） 統合化ＣＡＳＥ環境の主要機能

12） ＡＩ（ＫＢＳ）技術の活用

13） 利用者、対象アプリケーション等に合わせてカスタマイズ

14） 異種実行環境からの移行を支援（例．メインフレーム－＞ＷＳ）
異種領域からも