基幹情報システム開発のための
生産技術及び見積技術に関する研究

2008 年 1 月

津 田　道 夫

基幹情報システム開発のための
生産技術及び見積技術に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2008 年 1 月

津 田　道 夫

内容梗概

現代社会の活動を支えているのは，あらゆる分野で構築された基幹情報システムである．基幹情報システムとは，企業や行政機関などの活動を支える経営情報システムのことである．銀行オンラインシステムや鉄道座席予約システムなど日常生活に密着している情報システムも多い．

基幹情報システムの開発には常に課題がある．業務ソフトウェア(エンタプライズ系ソフトウェア)の規模は増加し，納期は厳しく，安定した品質が要求される．

通常，基幹情報システムは大規模システムが多く，SI ベンダと呼ばれるソフトウェア会社が顧客から発注を受けて開発する．開発プロジェクトが混乱した場合，品質の悪化や納期遅延による開発コストの増加が SI ベンダの業績に大きな影響を与える．開発プロジェクトが混乱する要因としては，技術面の要因と管理面での要因がある．技術面の要因では，ソフトウェア生産技術力の不足，技術力のある要員の不足，見積の失敗，性能や信頼性など非機能要件の設計力不足などがある．管理面の要因では，プロジェクト管理不足，不明確な業務仕様のままの開発，発注者(顧客)と開発者(SI ベンダ)の契約不備などがある．

本論文では，開発プロジェクト混乱の技術要因の中からソフトウェア生産技術と見積技術の解決に取組み，以下の目標を設定した．

（1） ソフトウェア開発方式の標準化による開発力向上

（2）既存ソフトウェアの再利用による生産性向上

（3）開発コストの早期見積によるリスク管理

本研究の目的は，これらの目標実現に寄与することである．

（1）では，情報システム開発方法論と統合開発支援ツールを開発した．開発方法論では，標準開発プロセスを定義し，開発作業を標準 WBS で詳細化する．標準 WBS により，開発する要員の役割別に作業内容と責任範囲，作業基準が明確になり大規模プロジェクトの運営が容易になる．開発プロセスはフェーズと呼ぶ単位に分け，フェーズ単位に進捗を管理する．

統合開発支援ツールは，開発方法論と一体になったツールであり，ソフトウェア部品によるプログラム自動生成を特長にしている．ソフトウェア部品は，スケルトンと処理部品がある．スケルトンは，プログラムの制御構造を形態別にパターン化したものである．処理部品は，共通機能を部品化した機能部品と，データ固有の処理を持つデータ処理部品を開発した．

統合開発支援ツールは，1981 年からソフトウェア部品再利用技術を中心にプログラム自動生成

機能を拡張してきた．本研究では，統合開発支援ツールの評価を 1980 年代から 2000 年代まで長期間収集した実績データで分析した．生産性指標としてプログラム生成率，品質指標として不良密度を用いて分析した．また，プログラマの習熟率を分析した．習熟率の結果は，大規模プロジェクトの開発組織の生産力確保に，教育・訓練と反復作業の必要性を示している．

（2）の既存ソフトウェアの再利用は，基幹情報システムの一部に既存ソフトウェアを流用するのではなく，既存システムから業務仕様を抽出・理解して部品化などに活用するのが目標である．具体的には，プログラムに格納されている業務データ(データ項目)に着目してソースコードを分解，変換，整理するデータ中心型リバースエンジニアリングを開発した．ソースコードの各ステートメントを，個々のデータ要素に分解し，一対のデータ要素に変換して論理を表現する．表現された論理をカプセル化して業務ルールを抽出する．抽出した業務ルールの正当性と仕様抽出率を実プロジェクトのデータと比較して，高い一致率を得た．

（3）では，最初の見積である試算見積の手法として協調フィルタリング法による予測モデルと UCP(Use Case Point)法による自動計測方式を開発した．

協調フィルタリング法は，検索データに欠損値が含まれることを前提とした事例ベース類推法のひとつである．類似プロジェクト検索に用いる変数を 6 種類設定した予測モデルを開発した．また，膨大な検索アルゴリズムから規模見積に最適なアルゴリズムを探し出すツールを開発した．

UCP 法は，オブジェクト指向開発で作成したユースケースモデルをもとに，規模や開発工数を見積る手法である．計測で必要なアクタ分類とユースケース分類の自動化手法を開発した．また，この手法によるユースケースポイント計測支援ツールを開発した．

両者とも評価の結果，高い精度での予測結果を得た．協調フィルタリングでは，欠陥を含んだデータでも実用的な精度で予測できることを確認した．UCP 法による自動計測方式では，支援ツールの自動計測値と熟練者の手動計測値が近い結果を得た．

研究業績一覧

（1）学術論文誌

1．Michio Tsuda, Sadahiro Ishikawa, Osamu Ohno, Akira Harada, Mayumi Takahashi, Shinji Kusumoto, Katsuro Inoue, “Effectiveness of an Integrated Case Tool for Productivity and Quality of Software Developments”, IEICE Transactions on Information and Systems, Vol.E89-D, No.4, pp.1470-1479, April 2006.

2．津田道夫, 楠本真二, 松川文一, 山村知弘, 井上克郎, 英繁雄, 前川祐介, “ユースケースポイント計測におけるアクタとユースケースの自動分類の試みと支援ツールの試作”, 電子情報通信学会論文誌D（採録決定）.

（2）国際会議(査読あり)

1．O. Ohno, H. Matumoto, M. Tsuda, “Development and Evaluation of Structured Software Development System “EAGLE/P(CANDO)””, Proc. of COMPINT'85, pp. 114-119, 1985.

2．Y. Morioka, H. Matumoto, O. Ohno, M. Tsuda, H. Maezawa, ““EAGLE/P”: A Program Synthesizer System Using Original Components” , Proc. of the 11th Annual International Computer Software & Applications Conference, pp.306-310, 1987.

3．M. Tsuda, Y. Morioka, M. Takadachi, and M. Takahashi, “Productivity analysis of software development with an integrated CASE tool”, Proc. of the 14th International Conference on Software Engineering, pp.49-58, 1992.

4．I. Nagaoka, K. Sanou, D. Ikeo, T. Nagashima, S. Akiba, M. Tsuda, “A Reverse Engineering Method and Experiences for Industrial COBOL System” , Proc. of the 4th Asia Pacific Software Engineering and International Computer Science Conference(APSEC97/ICSC97), pp.220-228, 1997.

5．S. Kusumoto, K. Inoue, T. Kashimoto, A. Suzuki, K. Yuura, M. Tsuda, “Function Point Measurement for Object-Oriented Requirements Specification”, Proc. of International Computer Software and Applications Conference, pp.543-548, 2000.

6．S. Kusumoto, M. Imagawa, K. Inoue, S. Morimoto, K. Matsushita, M. Tsuda, “Function Point Measurement from Java Program”, Proc. of 24th International Conference on Software Engineering (ICSE2002), pp.576-582, 2002.

（3）全国大会・研究会等

1．永岡郁代, 津田道夫, 団野博文, 佃軍治, “既存ソフトウェアの再利用とオブジェクト指向によるシステム開発における考察”, 情報処理学会ワークショップ論文集, Vol.95, No.1, pp.139-144, 1995.

2．津田道夫, 大野治, 小林正樹, 中野恭秀,“2000 年問題”, 情報処理学会ワークショップ論文集, Vol.97, No.1, pp.73-74, 1997.

3．米田豊満, 津田道夫, 湯浦克彦, 勝瑞雅也, 宇川裕行, 小堺弘光,“未経験者によるオブジェクト指向設計の適用と評価”, 情報処理学会シンポジウムシリーズ, Vol.99, No.9, pp.87-90, 1999.

4．団野博文, 湯浦克彦, 岩渕史彦, 津田道夫, “コンポーネント指向業務設計技法（HIPACE/AGORA）の開発と改良”, 情報処理学会シンポジウムシリーズ, Vol.99, No.9, pp.67-74, 1999.

5．鈴木文音, 大坪稔房, 勝瑞雅也, 湯浦克彦, 津田道夫, 宮崎肇之, 降旗由香理, 小室彦三, 大野治, “JAVA によるシステムの開発設計と COBOL による現行系との比較評価”, 情報処理学会シンポジウムシリーズ, Vol.99, No.9, pp.87-90, 1999.

6．川村透, 今城哲二, 津田道夫,“EDP 部門向け開発支援ツール“EAGLE””, 情報処理学会第 29 回全国大会, 3R-1, 1984.

7．津田道夫, 葉木洋一, 大野治, 小野功,“システム開発支援 EAGLE の開発”, 情報処理学会第 30 回全国大会, 4S-1, Vol.1, pp.619-620, 1985.

8．宮副英彦, 津田道夫, 大野治, “EAGLE におけるプログラム部品の開発”, 情報処理学会第 30 回全国大会, 4S-6, Vol.1, pp.629-630, 1985.

9．葉木洋一, 津田道夫, 小林正和, 大野治,“システム開発支援 EAGLE―総論―”, 情報処理学会第 31 回全国大会, 2F-1, Vol.1, pp.455-456, 1985.

10. 高橋まゆみ, 津田道夫, 大野治,“EAGLE によるシステム開発の標準化”, 情報処理学会第 31 回全国大会, 2F-2, Vol.1, pp.457-458, 1985.
11. 千吉良英毅, 永井義明, 小林正和, 津田道夫, “EAGLE におけるプログラム部品合成実現方式”, 情報処理学会第 31 回全国大会, 2F-3, Vol.1, pp.459-460, 1985.
12. 永井義明, 千吉良英毅, 小林正和, 津田道夫,“EAGLE における部品体系”, 情報処理学会第 31 回全国大会, 2F-4, Vol.1, pp.461-462, 1985.
13. 内藤一郎, 千吉良英毅, 小林正和, 印東功, 津田道夫, “EAGLE における仕様情報再利用方式”, 情報処理学会第 32 回全国大会, 3F-1, 1986.
14. 安藤潤市, 津田道夫, 穂垣博文,“EAGLE によるシステム開発の適用と効果”, 情報処理学会第 33 回全国大会, 3F-4, 1986.
15. 津田道夫, 曽根原勝, 葉木洋一, “EAGLE 機能改善要求の分析”, 情報処理学会第 33 回全国大会, 3F-5, 1986.
16. 高橋まゆみ, 津田道夫, 前澤裕行, “EAGLE 開発の分散化”, 情報処理学会第 33 回全国大会, 3F-6, 1986.
17. 永井義明, 横山岳浩, 津田道夫, 前澤裕行, “EAGLE 機械処理設計用知的分散システム”, 情報処理学会第 33 回全国大会, 3F-8, 1986.
18. 津田道夫,“HIPACE システム分析技法”, 情報処理学会研究報告「情報システムと社会環境」, Vol.1988, No.30, pp.1-11, 1988.
19. 玉井昌朗, 岡田世志彦, 飯田元, 井上克郎, 鳥居宏次, 長岡渡, 梅本肇, 津田道夫, “プロセスモデルに基づく分散開発支援システムの試作”, 情報処理学会第 44 回全国大会, 1J-6, 1992.
20. 田村和敏, 奥川淳一, 津田道夫,“CSS 統合開発環境(1)-概要-” , 情報処理学会第 45 回全国大会, 1992.
21. 津田道夫, “CAPSDF におけるリエンジニアリング”, 情報処理学会研究報告, Vol.93, No.4, pp.77-85, 1993.
22. 津田道夫, “CAPSDF におけるリエンジニアリング”, 情報処理学会研究報告, Vol.93, No.59, pp.165-172, 1993.
23. 南波則孝, 高橋まゆみ, 屋敷麻理, 津田道夫, “リエンジニアリング技術を用いたソフトウェアの標準化”, 情報処理学会第 48 回全国大会, 4K-7, 1994.

24. 津田道夫, “ソフトウェアプロセスの標準化と管理支援”, 情報処理学会シンポジウム論文集,

Vol.94, No.2, pp.155-156, 1994.

25. 津田道夫, 永岡郁代, 青木一紀, 和栗正一, “ソフトウェア変更作業の分析と支援機能”, 情報処理学会研究報告, Vol.95, No.11, pp.1-6, 1995.
26. 津田道夫, 大野治, 秋庭真一, 内藤一郎, 山川敦夫, 堀内一, “DORE(1)-二項分析による既存プログラムからの業務ルール抽出技術-”, 情報処理学会第 52 回全国大会, 6S-3, 1996.
27. 津田道夫, “業務仕様のコンポーネント化”, 情報処理学会シンポジウムシリーズ, Vol.98, No.1, pp.51-52, 1998.
28. 柏本隆志, 石田直也, 神谷年洋, 楠本真二, 井上克郎, 鈴木文音, 勝瑞雅也, 湯浦克彦, 津田道夫, “要求仕様書に対するファンクションポイント計測ツールの試作”, 電子情報通信学会技術研究報告, Vol.97, No.630, pp.87-93, 1998.
29. 柏本隆志, 楠本真二, 井上克郎, 鈴木文音, 勝瑞雅也, 湯浦克彦, 津田道夫, “要求仕様書に対するファンクションポイント計測ツールの試作と評価”, 電子情報通信学会技術研究報告, Vol.98, No.559, SS98-47, pp.17-23, 1999.
30. 今川勝博, 柏本隆志, 楠本真二, 井上克郎, 鈴木文音, 湯浦克彦, 津田道夫, “要求仕様書からのファンクションポイント計測ツールの改良　－要求分析ツール REQUARIO で作成された要求仕様書を対象として－”, 情報処理学会第 59 回全国大会, pp.301-302, 1999.
31. 今城哲二, 大野治, 津田道夫, 高橋まゆみ, “ビジネスオブジェクト近況”, 情報処理学会シンポジウムシリーズ, Vol.99, No.11, pp.49-50, 1999.
32. 柏本隆志, 楠本真二, 井上克郎, 鈴木文音, 湯浦克彦, 津田道夫, “イベントトレース図に基づく要求仕様書からのファンクションポイント計測手法”, 情報処理学会論文誌, Vol.41, No.6, pp.1895-1904, 2000.
33. 森岡佑, 谷口考治, 楠本真二, 井上克郎, 英繁雄, 前田憲一, 津田道夫, “プログラム実行時情報を用いたトランザクションファンクション抽出手法”, 情報処理学会第68回全国大会, Vol.1, pp.303-304, 2006.
34. 森岡佑, 谷口考治, 楠本真二, 井上克郎, 英繁雄, 前田憲一, 芝元俊久, 津田道夫, “プログラム実行履歴を用いたトランザクションファンクション抽出手法”, 電子情報通信学会技術研究報告, Vol.106, No.523, SS2006-79, PP.1-6, 2007.

（4）雑誌・書籍・企業論文誌

1．角谷一郎, 津田道夫, 葉木洋一, 石坂裕之, “システム・フローを自動生成する開発支援ソフト EAGLE2”, 日経コンピュータ, pp.121-136, 1986.7.7.

2．津田道夫, 葉木洋一, 前澤裕行, 石本信幸, 高橋まゆみ, “ワークステーションによるソフトの分散開発環境 SEWB”, 日経コンピュータ, pp.121-136, 1987.8.17.

3．津田道夫, 米永茂男, “EAGLE/SEWB によるシステム設計支援の機械化”, 事務管理第 27 巻第 5 号, pp.27-33, 1988.

4．津田道夫 共著（監修 ; 原田実）, CASE のすべて, 第 2 章 2.4 節, 第 3 章 3.1 節, 第 5 章, 5.6 節, 第 6 章 6.2 節, オーム社, 1991.

5．大野治, 降旗由香理, 津田道夫, “データ部品方式による実践事例”, Bit 別冊“ソフトウェアのモデル化と再利用”, pp.84-102, 1995.3.

6．津田道夫 共著（監修 ; 江村潤朗）, コンピュータの大百科, 第 5 章 CASE の話(PP.422-448), オーム社, 1996.

7．宮副英彦, 津田道夫, 小林正和, 前澤裕行, 堀内一, “アプリケーションシステムの効率的設計技法“HIPACE” ”, 日立評論, Vol.62, No.12, pp.15-20, 1980.

8．葉木洋一, 今城哲二, 津田道夫, “システム開発支援ソフトウェア“EAGLE” ”, 日立評論, Vol.66, No.3, pp.19-24, 1983.

9．葉木洋一, 北尾修治, 千吉良英毅, 津田道夫, 大野治, 仁平博三, “システム開発支援ソフトウェア“EAGLE”―EAGLE 拡張版 EAGLE2―”, 日立評論, Vol.68, No.5, pp.29-34, 1986.

10. 前澤裕行, 葉木洋一, 津田道夫, 印東功, “ソフトウェア生産技術の展望”, 日立評論,Vol.70, No.2, pp.1-6, 1988.

11. 西尾高典, 今城哲二, 中原俊政, 津田道夫, “日立製作所のアプリケーション開発支援体系“CAPSDF” ”, 日立評論, Vol.75, No.11, pp.9-14, 1993.

12. 湯浦克彦, 津田道夫, 初田賢司, 団野博文, “新企業情報システムを実現するコンポーネント指向業務設計技法”, 日立評論, 1998.5.

13. 井上克郎, 松本健一, 津田道夫, 新海平, “産学官連携によるエンピリカルソフトウェア工学プロジェクト EASE”, 日立システムジャーナル, 第五巻, pp.59-64, 2005.

14. 大杉直樹, 松本健一, 津田道夫, 中屋広樹, 十九川博幸, “協調フィルタリング技術によるソフトウェア規模の予測”, 日立システムジャーナル, 第六巻, pp.59-66, 2006.

謝辞

本研究の全般に関し，常日頃より適切なご指導を賜りました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に，心から深く感謝申し上げます．井上先生には，平成２年の阪大－日立共同研究以来，長年にわたりソフトウェア生産技術に関するご指導をして頂きました．

本論文を執筆するにあたり，適切なご助言とご指導を頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 増澤 利光 教授に心より感謝致します．

本研究の実施，並びに本論文の執筆にあたり，適切なご助言とご指導を頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 楠本 真二 教授に心より感謝致します．

本研究を行うにあたり，長年にわたりソフトウェア工学及びエンピリカルソフトウェア工学のご指導を頂きました奈良先端科学技術大学院大学前学長 鳥居 宏次 特任教授に心から深く感謝申し上げます．鳥居先生からは博士論文執筆のお奨めを頂きました．

本研究の基となった情報システム開発方法論，統合開発支援ツールの開発成果は，日立製作所 大野 治氏，日立ソフトウェアエンジニアリング 前澤 裕行氏，日立システムアンドサービス 高橋 まゆみ氏，東京国際大学(元日立製作所) 故今城 哲二教授，東京国際大学(元日立製作所) 堀内 一 教授との共同作業によるものです．多くの協力を頂き，厚く感謝いたします．

リバースエンジニアリングの研究は，日立コンサルティング(元日立製作所) 秋庭 真一氏に協力頂きました．厚く感謝いたします．

協調フィルタリングによる試算見積の研究では，奈良先端科学技術大学院大学 松本 健一 教授，ＮＴＴデータ(元奈良先端科学技術大学院大学) 大杉 直樹氏にご指導頂きました．厚く感謝いたします．

本研究にあたり，機会を与えて頂いた日立システムアンドサービス 小島 一翁氏に感謝するとともに，研究や論文執筆で協力頂いた井上研究室及び日立システムアンドサービス 生産技術部の皆様に感謝いたします．

目次

第 1 章　まえがき

1.1　本研究の目的

経産省は，ソフトウェアを「現代経済社会の基盤」であり「価値を実現する源泉」であると述べている．「ソフトウェアは，あらゆる産業を支え，かつ製品・サービスの価値を生み出している」[1].

基幹情報システムは，企業では製品や商品を生む生産基盤であり，個人では便利で快適な生活を営む生活基盤である．また，医療，福祉，自治体などからのサービスを受ける基盤でもある．

自動車産業を例にすれば，製造メーカは生産システムや調達システムを活用して自動車を生産している．自動車と社会基盤の関係では，国家プロジェクト「高度道路交通システム(ITS:Intelligent Transport Systems)」がある．交通渋滞 20%削減，死亡事故半減，CO2 の 15%削減などを目標にしている．環境保護の視点からは，自動車のリサイクルがある．「自動車リサイクル法」が制定され 2005 年 1 月より施行されている．引取業者，フロン回収業者，解体業者等の関係者に使用済自動車の処理状況報告を義務付け，適正に処理されているかを監視する．これを運用するのが自動車リサイクルシステムである．

もうひとつの例として電子取引がある．PC，ネットワークなどの技術進歩に加えて，電子決済，IC カードなどの技術も加えて実用化された電子取引(EC；Electronic Commerce)も社会基盤として確立している．我が国の 2004 年電子商取引の市場規模は，事業者向け(B2B)は 102 兆 7 千億円と推計され，電子商取引化率は 15%，1998 年(8 兆 6 千億円)と比較して 12 倍となっている．一般消費者向け(B2C)は 5.6 兆円(対前年比 28%増)であり順調に拡大している[2].

これらを実現しているのが，企業や公官庁の基幹情報システムである．基幹情報システムを動かす業務ソフトウェアは，エンタプライズ系ソフトウェアと呼ばれている．従来は，事務処理ソフトウェア，ビジネスアプリケーションソフトウェアとも呼ばれていた．

エンタプライズ系ソフトウェアの開発は常に課題を抱えている．ブルックスの「銀の弾丸」論文[3]に代表されるようにソフトウェアの生産性と品質の向上は，いつの時代でもソフトウェア工学のテーマである．500 人月以上の大規模プロジェクトの 50%が納期遅延し，50%が予算をオーバし，30%の顧客が品質に不満があるという報告がある[4]．また，ソフトウェアの規模も拡大を続けている．1980 年代では SIS(戦略情報システム)と呼ばれた基幹情報システムの開発が始まり，例えば都市銀行第 3 次オンラインシステムでは 700 万ステップの規模をもつソフトウェアを開発してきた．1990 年代後半にはインタネットを利用した BtoC(Business to Consumer)ビジネスの開始もあり，ソフトウェアの規模は増大している．

ソフトウェア開発プロジェクトが混乱する要因としては，技術面の要因と管理面での要因がある．技術面の要因では，オープンソースなど新技術への対応力不足，ソフトウェア生産技術力の不足，開発コスト見積の失敗，性能や信頼性など非機能要件の設計不足などがある．管理面の要因では，プロジェクト管理不足，不明確な業務仕様のままの開発，発注者と開発者の契約不備などがある．大規模プロジェクトでは，開発にかかわる関与者が多岐になりコミュニケーション不足や，変更管理や統合管理などの管理不足が起こりやすい．

筆者は，日立製作所(以下，日立と言う)及び日立システムアンドサービス(以下，日立 SAS と言う)で基幹情報システムの生産技術の開発に従事してきた．特に 1983 年からは，日立の生産技術開発部門でソフトウェア生産技術の企画・開発と普及に専従してきた．長年におけるソフトウェア生産技術開発の経験から，基幹情報システム開発の生産性と品質向上には，以下に述べる目標が重要であると考えてきた．

目標１：ソフトウェア開発プロセスの標準化と，それを一貫して支援するツール

目標２：既存ソフトウェアの再利用

目標３：ソフトウェア規模の早期見積

本研究の目的は，これら３つの目標の一部を実現することである．

1.2　本研究の概要

本研究では，1.1 節で述べた３つの目標を達成するために，以下の 3 点の研究を行なった．

（１）基幹情報システム開発方法論と統合開発支援ツール

（２）リバースエンジニアリングによる業務仕様理解支援

（３）ソフトウェア規模の試算見積

(1)は目標１，(2)は目標２，(3)は目標３の実現のための研究である．特に，(1)(2)は，データ中心型アプローチ(DOA：Data Oriented Approach)のコンセプトに基づいて研究した．DOA とは，「データ項目は，設計情報の中では一意で，安定しているので，はじめに共有資源として設計すれば安定した情報システムが建設できる」という考えである．データ項目には，名称や属性と共に，企業の業務プロセスや業務ルールが固有処理として埋め込まれている．

（1）基幹情報システム開発方法論と統合開発支援ツール

基幹情報システムの開発では，参加する要員も多く，開発作業の標準化が重要である．開発方法論は，開発プロセス(開発手順)や開発技法，開発基準を標準化して体系化したものである[5].

ソフトウェア開発プロセス[6]は，逐次型モデル[7]，反復型モデル[8]などが提案されている．開発作業と成果物の定義は，内容を階層的に詳細化していく WBS(Work Breakdown Structure)[9][10]手法が提案されている．

開発技法は，構造化技法，オブジェクト指向技法，アジャイル開発技法がある[12].

構造化技法は，構造化分析(SA)，構造化設計(SD)，構造化プログラミング(SP)がある．構造化分析は，データの流れに注目した DFD(Data Flow Diagram)法[13][14][15]，入出力のデータ構造に着目した機能抽出法[16][17]がある．構造化設計は，機能をトップダウンで段階的に詳細化する設計法で，設計の結果は構造化チャート(Structure Char)と呼ぶ図形で表現する[18][19]．構造化プログラミングでは，GO TO レスの構造化チャートや言語仕様が開発されている[5][20][21].

オブジェクト指向技法には，オブジェクト指向分析(OOA)，オブジェクト指向設計(OOD)，オブジェクト指向プログラミング(OOP)がある．オブジェクト指向技法は，構造化分析と同じくグラフ図でオブジェクトの構造，属性，処理などを表現する[22][23][24][25][26]．モデル表記法を標準化したのが UML(Unified Modeling Language)である．

本研究では，基幹情報システム構築で効果的なソフトウェア開発プロセスモデルと開発技法を提案する．標準開発プロセスを定義し，開発作業を WBS で詳細化する．開発プロセスをフェーズと呼ぶ単位に分け，開始と終了をコントロールするフェーズドアプローチ手法を提案する．

開発技法は，基幹情報システムで使われるデータ項目を最初に分析して，その分析後にソフトウェアの構造を設計するデータ中心型アプローチ(DOA)[29][61]を主要技法にする．データ項目の名称，意味や属性を標準化し参加者の誤解や不整合を防止する．データ操作をカプセル化してデータ処理部品にし，ソフトウェア製造に利用する．業務仕様の分析と設計には，構造化分析技法を用いる．DFD に加えて日本語の表記を制限した構造化日本語を提案する．

統合開発支援ツール(統合型 CASE：Computer Aided Design Software Engineering)は，開発方法論と一体になりソフトウェア開発プロセス全体を支援するツールである[30]. 1980 年代から，多くのツールベンダが統合開発支援ツールを開発してきた[31][32]．その機能は，ソフトウェア開発プロセスの一貫支援，DOA によるデータの一貫管理，開発技法を支援する対話型設計支援機能とプログラム自動生成機能，テスト支援機能である[5][30].

ソフトウェア開発プロセスの一貫支援とは，開発方法論で定義した開発手順のメニュー表示と

開発作業をナビゲーション(プロセス管理)することである．

データは，データ辞書(Data Dictionary)やリポジトリ(Repository：情報資源倉庫)で一元管理される．データ辞書は，データ項目に関する名称，属性，関連，操作を格納している．リポジトリは，基幹情報システムの情報も企業においても価値のある資源であるという DOA のコンセプトで作られた情報資源倉庫で，設計仕様データ，ソースコード，各種管理データを格納している．

設計支援機能は，業務仕様モデルの編集を支援する機能である．データモデル，プロセスモデル，UML モデルをサポートしている．プログラムの自動生成機能は，ソフトウェア部品を再利用する生成方式と仕様書やチャート図を生成するビジュアルプログラミング方式がある．

本研究では，開発方法論をサポートする統合開発支援ツールの提案と効果を分析する．統合開発支援ツールは，1981 年から現在までの 26 年間にわたり基幹情報システムを対象にソフトウェア部品再利用技術を中心にしたプログラム自動生成機能を拡張してきた．

本研究では，統合開発支援ツールの機能拡張を 1980 年代から 2000 年代までの期間で３フェーズに分けて述べると共に，データ項目辞書を中心にしたデータ処理部品を提案する．また，それぞれのフェーズの実績データを示して生産性を分析する．長期間にわたり統合開発支援ツールの生産性を分析したレポートは少ない[33]．生産性指標として，スケルトンとソフトウェア部品によるプログラム生成率，テストフェーズでの信頼性，プログラマの習熟曲線の分析結果を述べる．特に最後の習熟曲線は，大規模プロジェクト開発における開発組織の生産力確保に重要な指標であり，教育・訓練と反復作業の必要性を示している．

（２）リバースエンジニアリングによる業務仕様理解支援

基幹情報システムの開発は，新規開発よりも既存システム(レガシーシステム)再構築の場合が多い．基幹情報システムのダウンサイジングや Web システム化などのシステム再構築をレガシーマイグレーションと呼んでいる．レガシーマイグレーションで留意すべきことは，レガシーシステムで実現してきた業務仕様をもれなく反映させることである．換言するとレガシーシステムに埋め込まれている業務仕様は企業の事業形態が大きく変化しない限り再利用すべき重要な情報である．レガシーシステムから業務仕様を理解すると共に，データ項目や業務仕様を抽出して，データ標準化やソフトウェアの部品化に再利用するのは有効である．

従来では，既存の業務仕様の理解や設計情報の抽出にレガシーシステムの仕様書調査や業務有識者へのインタビューなどにより分析･設計作業をしてきた．しかし既存の仕様書は永年の機能拡張や変更の内容が反映されていない場合が多く，有識者のインタビューもスキルに依存するため

に必要な情報を十分に収集できないケースがあった．これらの情報が欠落したまま再構築した結果，トラブルが発生する危険性があった．

レガシーシステムから設計情報を抽出する技術はリバースエンジニアリングの分野で多くの研究がなされてきた[34]．リバースエンジニアリングとはシステム構成要素の解明により，他の形式か高水準の設計仕様を逆生成することである．再文書化(re-documentation)，設計復元(design recovery)，関数抽象化(function abstraction)，業務ルール(business rules)の抽出･表現がある[35][36]．これまで，リバースエンジニアリング技術による設計情報抽出技術や影響波及解析技術が提案され，多くは実用化されてきた[37][38]．主な技術は，相互参照技術，プログラム表現技術，プログラムスライス技術である．相互参照技術は，プログラムと画面・帳表のデータ項目関連情報などのクロスリファレンス作成技術である[39]．表現技術は，ソースリストを字ずらしや制御の流れを矢印などで示して清書したり，木構造のチャートで図式化してプログラムを読み易くする技術である．

本研究では，レガシーシステムから業務ルールを抽出・理解する手法として，プログラムに格納されている業務データ(データ項目)に着目してソースコードを分解，変換，整理して，業務ルールを抽出するリバースエンジニアリング DORE(Data Oriented Approach Reengineering)を提案する．本研究では，「データ項目には，企業の基幹業務のプロセスやルールが埋め込まれている」というデータ中心型アプローチ DOA[29]のコンセプトに基づいている．ソースコードの各ステートメントを，個々のデータ要素に分解し，分解したデータ要素を一対のデータ要素にて表現される２項オブジェクトとして表現し，該当する２項オブジェクトの組み合わせによって，各ステートメントの論理を表現する．次に，表現された論理をカプセル化して業務ルールを抽出する．抽出した業務ルールは，レガシーシステムの理解支援に活用する．また，データ処理部品としてソフトウェアの再利用に活用する．

本研究では，データ中心型リバースエンジニアリング DORE による業務ルール抽出方式を提案すると共に，抽出した業務ルール正当性の評価と設計仕様書に記述している業務ルールに対するカバー率で仕様抽出量を評価した．

（３）ソフトウェア規模の試算見積

基幹情報システムの開発プロジェクトにおいて，品質の悪化や納期遅延による開発コストの増加は情報産業に属する企業では業績に大きな影響を与える場合がある．プロジェクトのトラブルを未然に防止するには，ソフトウェアの規模，開発工数，コストなどを的確に見積ってリスク管

理することが重要である．プロジェクト管理では見積は重要な要件である[40].

見積は，ソフトウェア開発プロセスの上流から下流に逐次，段階的に実施される．企画・計画段階で見積る試算見積，基本設計後に見積る概算見積，詳細設計後に見積る詳細見積がある．

ソフトウェアの見積手法[41]は，仕様，規模や工数の間に存在する特定の数学的な関係を用いて規模や工数を算出するパラメトリック法，プロジェクトのすべての成果物とアクティビティを積算するアクティビティベース法，過去のプロジェクトの実績データからプロジェクトの特性値を比較する事例ベース類推法などがある．現在は，最初に業務要件(業務仕様，要求定義)からソフトウェアの規模を算出し，次に規模にシステム特性(複雑性，品質，性能など)から得たパラメータ値を掛けて開発工数を算出するパラメトリック法が主流になっている．

ソフトウェア規模を測るメトリクスとして，「ソースコードの量」と「機能の量」のメトリクスがある．「ソースコードの量」は，ステップ数(S；Step)，関数数，クラス数，サイクロマティックス(複雑性)などのメトリクスがあるが，一般的にはステップ数がよく使われている．ステップ数は，コード行数(LOC；Lines of Code)とも言われている．「機能の量」では FP(Function Point)，UCP(Use Case Point)のメトリクスがある．FP 法は，ソフトウェアの機能要件を洗い出して，各要素の数に複雑性などで重み付けをして算出する方法である[42]．FP 法の利点は，開発言語や開発環境/実行環境から独立して算出できる，機能から規模を算出するので発注者と SI ベンダが共通の認識が持てる，生産性評価がし易いなどがあり，現在は FP が普及しつつある．FP 法は，複数の手法が提案されているが，現在は，FP 法の国際団体である IFPUG(International Function Point Users Group)の計算法が標準になっている[43].

UCP 法は，FP 法をベースとし，近年増えつつあるオブジェクト指向開発の早期要求分析の段階で作成されるユースケースモデルをもとに，規模や開発工数を見積る手法である[44][53].

企画・計画段階での試算見積は，発注者の予算獲得や発注者と SI ベンダとの最初の契約で使われるので重要な見積である．しかし，重要な見積であるにもかかわらず，この時点では業務要件は確定しておらずソフトウェア規模の算出は困難であった.

試算見積の手法として, FP 試算法, NESMA 法[45], 協調フィルタリング法[46], UCP 法[47][48], FP 要素見積法[49][50]，電中研法[51]などが提案されている.

本研究では，試算見積の手法として協調フィルタリング法による予測モデルの提案と UCP 法による自動計測方式を提案する. いずれも支援ツールを試作して，実プロジェクトで評価をした.

協調フィルタリング法では，過去のプロジェクト実績情報から，予測対象の類似プロジェクトを抽出して，規模(FP 値)を算出する.

協調フィルタリング法は，検索データに欠損値が含まれることを前提とした事例ベース類推法のひとつである．情報検索の分野において，多くの情報からユーザの好みに合う情報を抽出して推薦する技術として研究されてきた[52]．書籍のネット販売では，この技術を用いて顧客の好みに合った推薦図書を提示している．しかし，ユーザの評価を用いて書籍を推薦する場合は，ユーザの評価データの値域が一定なのに対して，ソフトウェア開発では，例えば画面数や設計工数など値域が一定でないメトリクス(以下，変数と言う)を用いる．

本研究では，ソフトウェア規模見積に用いる変数を 6 種類設定した予測モデルを提案する．変数は数値で表現できないカテゴリ変数と数値で表現できる数値変数がある．また，膨大な検索アルゴリズムからソフトウェア規模見積で最適なアルゴリズムを探し出す探索ツールを試作した．

UCP法は, 要求分析で作成したユースケースモデルに基づいて工数見積りをする. しかし, UCP法を実際に適用する場合，計測者の主観に依存する部分があるため，同一ユースケースモデルに対する計測でも，計測者によって誤差が生じるという問題点も存在する. 本研究では，このような計測者による誤差をなくすことを目的として幾つかの制限を設けた上で，UCP計測で行われるアクタとユースケースの自動分類手法を提案する．また，この提案によりユースケースポイント計測支援ツールを試作した．更に，本ツールをソフトウェア開発プロジェクトで作成したユースケースモデルへ適用し，その適用可能性を評価した．

1.3　本論文の構成

本論文では，第 2 章で開発方法論と統合開発支援ツールを，第 3 章でリバースエンジニアリングによる業務仕様理解支援を，第 4 章でソフトウェア規模の試算見積を，第 5 章むすびで まとめと今後の研究方針を述べる．

第2章　開発方法論と統合開発支援ツール

2.1　緒言

基幹情報システム開発では，開発方法論の標準化と統合開発支援ツールの活用が効果的である．開発方法論とは，ソフトウェア開発プロセス(開発手順)，開発作業，成果物，開発技法を標準化，体系化したものである[5]．統合開発支援ツール(統合型CASE)は，開発方法論と一体になりソフトウェア開発プロセス全体を支援するツールである．

ソフトウェア開発プロセス[6]は，逐次型モデルではウオータフォールモデル[7]が提案されており，反復型モデルではスパイラルモデル[8]，イタラティブモデル，インクリメンタルモデル， アジャイルモデルなどが提案されている．国産ソフトウェアベンダの開発プロセスは， ウオータフォールモデルが72%を占めているが， スパイラルモデルなどの反復型モデルの適用も増えている[11]．開発作業と成果物の定義は，WBSと呼ぶ体系化手法が提案されている[9][10]．プロジェクトで実施されるすべての作業を拾い出して，その内容を階層的に明確化していく手法である．本来はプラント建設などのエンジニアリングのプロジェクト管理で用いられた技法であるが情報システム開発でも広く使われている．

また，ソフトウェア開発プロセスだけでなく，情報システム開発及び取引の明確化を目的に，情報システムの構想からその廃棄にいたるまでのソフトウェアライフサイクルプロセスを可視化し，共通の枠組みを規定した SLCP-JCF(Software Life Cycle Process - Japan Common Frame)がある．国際標準である ISO/IEC12207 と整合をとりながら，国内ソフトウェア市場の特性を加えている．最新版として 2007 年に，企画や要求定義などユーザ主導作業を強化した SLCP-JCF2007 が発行されている[54].

開発技法は，構造化技法，オブジェクト指向技法，アジャイル開発技法が提案されている．

構造化技法は，構造化分析(SA)，構造化設計(SD)，構造化プログラミング(SP)に分かれる．構造化分析は，データの流れに注目した DFD を基本とする Yourdon 法[13]や DeMarco 法[14]，Gane/Sarson 法[15]，入出力のデータ構造から機能を抽出する Jackson 法[16]，Warnier/Orr 法[17]がある．

構造化設計は，機能をトップダウンで段階的詳細化してプログラム構造を決める設計法で，設計の結果は構造化チャートと呼ぶ図形で表現する．Constantine[18]により主要概念が開発され，その後に Myers が複合設計(composite design)として改良している[19].

構造化プログラミングでは，Dikstra の「GO TO 命令は有害である」と言う有名な記事[20]を出発点にして，GO TO レスの構造化チャートや言語仕様が開発された．構造化チャートは，構造化

プログラミングの 3 基本要素(連接，分岐，反復)を木構造図で表現した記法である．PAD 法[21]，HCP 法，SPD 法，YAC 法，Nassi & Shnederman の NS 法がある[5].

オブジェクト指向技法には，オブジェクト指向分析(OOA)，オブジェクト指向設計(OOD)，オブジェクト指向プログラミング(OOP)がある．オブジェクトとは，現実世界のデータを抽出して，属性と操作をカプセルにしたものである．データの抽象化(data abstraction)とも呼ばれている．

オブジェクト指向分析/設計技法は，複数のグラフ図を用いてオブジェクトの構造，属性，処理などを表現する．Cord & Yourdon 法[22]，Shlaer & Mellor 法[23]，Booch 法[24]，OMT 法[25]がある．この中から OMT 法，Booch 法に OOSE/Objectory 法[26]を統合してモデル表記法を標準化したのが UML(Unified Modeling Language)[27][28]である．UML は 1997 年に OMT(Object Management Group)で標準化されている．2006 年にはバージョン 2.0 がリリースされている．

基幹情報システム開発は，開発期間が長く，参加者が多いのが特徴である．本番運用後に備えて品質と保守性の高い情報システムの構築が要求される．開発では，発注者と開発ベンダでプロジェクト体制を組むが，多様な関与者が参加する．発注者側では，プロジェクトオーナー，情報システム開発部門の技術者，エンドユーザが参加する．開発ベンダ側では，プロジェクト管理者，業務ソフトウェア開発者，業務運用設計者，テスト担当者，共通技術開発者，インフラ担当者などが参加する．また業務ソフトウェア開発は，複数の外部協力会社が参加することが多く，階層的な下請け構造になっている場合もある．近年，中国やインドなどへの国際調達が増えている．

このような背景で，開発方法論は，適用の容易性があること，参加者のコミュニケーションが図れること，客観的な成果の評価が出来ること，開発手法を統一して生産性と信頼性を確保することが要求される．

統合開発支援ツールは，開発方法論と一体になりソフトウェア開発プロセス全体を支援するツールである．以下に特徴を示す．

・標準開発方法論(標準開発プロセス，標準開発技法)のサポート

・ソフトウェア開発ライフサイクル全般のサポートとデータ一貫管理

・ソフトウェア部品再利用技術によるプログラム自動生成

開発プロセスのサポートとは，標準開発手順のメニュー表示と開発作業をナビゲーションすることである．開発技法の記法に従って編集と生成，実行を対話処理で行い，標準フォーマットの形式で各種ドキュメントを自動生成する．編集作業などで作成した仕様データは，次作業の入力情報として引き継がれていく．

データは，データ辞書やリポジトリで一元管理される．データ辞書は，データ項目に関する名

称，属性，関連，操作が格納されている．リポジトリには，プログラム，設計仕様，レコードやデータベースの定義情報，画面/帳票定義情報などが格納されている．リポジトリは，「基幹情報システムの情報も企業においても価値のある資源である」という情報資源管理の考えから登場した情報資源倉庫である．

設計支援機能は，データモデルや業務仕様モデルの編集を支援する機能である．データモデルでは Chen の E-R(Entity-Relationship)モデル[55]，プロセスモデルでは DFD モデル，オブジェクト指向モデルでは UML をサポートしたものが多い．プログラムの自動生成機能は，ソフトウェア部品を再利用してプログラムを生成する方式と仕様書やチャート図の編集処理から自動生成するビジュアルプログラミング方式がある．ソフトウェア部品再利用方式は，標準パターン(スケルトン，フレームワーク)と機能部品から半完成部品を生成して固有部分を追加作成する方式である．ビジュアルプログラミング方式はアクション図や木構造チャートなどからプログラムを自動生成する方式である．テスト支援は，テストデータやスタブの生成，ソースコードを表示した対話型テスト支援，テストカバレージの測定などの機能がある．

統合開発支援ツールが，開発プロセスの一貫支援とデータの一貫管理を実現するには統合プラットフォームが必要になる．統合プラットフォームのリファレンスとして PCTE(Portable Common Tool Environment)[56]のトースタモデルがある．支援ツールの搭載と制御はタスク管理サービスが行い，データはリポジトリで管理して，データ統合サービスが支援ツールにデータを渡す．これらをあわせてオブジェクト管理と呼ぶ.

本章では，ソフトウェア開発プロセスモデルと開発技法を標準化した開発方法論HIPACE (Hitachi High-pace)と統合開発支援ツールEAGLE(Effective Approach to Achieving High Level Software Productivity)を提案する．また，統合開発支援ツールの効果を分析する．

開発方法論では，ソフトウェア開発プロセスの作業をWBSで定義して，プロジェクトの特性に合わせて選択できるようにする．名称，設計フォーマット，開発作業内容を統一する．プロジェクト参加者の担当作業と責任範囲を明確にする．開発プロセスをフェーズと呼ぶ単位に分け，開始と終了をコントロールするフェーズドアプローチ手法を提案する．

開発技法として，データ項目の分析後にソフトウェアの構造を設計するデータ中心型アプローチ DOA を提案する．データ項目の名称，意味や属性を標準化データ辞書に登録する．正規化したデータ構造はデータベース設計に反映する．データ項目は，名称，意味，属性に加えて，操作をカプセル化してデータ処理部品にする．この考え方はオブジェクト指向の考え方に似ている．

業務仕様の分析と設計には，構造化設計技法を用いる．構造化設計技法は，DFD による業務仕

様の記述に加えて日本語の表記を制限した構造化日本語を提案する．

統合開発支援ツールは，1981 年から現在までの 26 年間にわたり基幹情報システムを対象にソフトウェア部品再利用技術を中心にしたソフトウェア自動生成機能を拡張してきた．プログラム開発の生産性向上にソフトウェアの再利用は有効な戦略である[57]．本研究では，統合開発支援ツールの機能拡張を 1980 年代から 2000 年代までの 30 年を 3 フェーズに区切り，データ辞書を中心にしたデータ処理部品の拡充を述べる．また，それぞれのフェーズの実績データを示してソフトウェアの生産性を分析する. 生産性指標として, ソフトウェア部品によるプログラム生成率, 部品利用による品質，プログラマの習熟曲線の分析結果を述べる．習熟曲線は，大規模プロジェクト開発における開発組織の生産力確保に重要な指標であり，統合開発支援ツールの教育・訓練と反復作業の重要性を示している．

以降，2.2 節で基幹情報システム開発方法論 HIPACE を述べ，2.3 節で統合開発支援ツール EAGLE を述べ，その評価を 2.4 節で述べる．

2.2　基幹情報システム開発方法論 HIPACE

開発方法論 HIPACE は，1980 年に第 1 版(V1)が開発され，1990 年代に V2，2000 年代に V3 に拡充している．開発方法論は，主な IT ベンダでも開発されてきた．方法論のさきがけとしては J.Martin の IE(Information Engineering)がある．企業の情報戦略，ビジネスエリア分析など上流工程を重視しているのが特長である[58]．国産主要 IT ベンダは，STEPS/SDUP(NEC)，SDEM(富士通)，ADSG(日本 IBM)，TERASOLUNA(ＮＴＴデータ)を開発している．

開発方法論 V1 を開発した背景には，1970 年代後半に急激に増大したソフトウェア開発規模とソフトウェア工学の研究成果の実用化がある．

B.Boehm は 1973 年に「1970 年代にはソフトウェアコストがハードウェアコストを上回り，1980 年代には大部分がソフトウェアコストになる」という予測をした[59]．この予測を立証するように企業における情報システムの規模は 1970 年代後半から増大し，ソフトウェア開発の生産性向上が課題となった．図 2.1 は，主要企業における情報システムの規模の拡大例である[60]．いずれも日本を代表する企業の例であるが，1970 年代後半から規模が増大しており，市場からソフトウェア生産技術の提供が IT ベンダに強く求められるようになった．

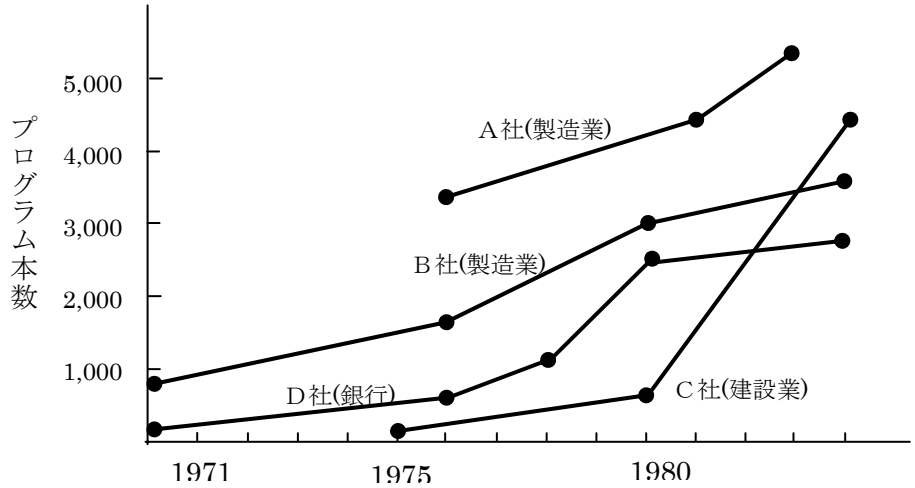


図 2.1　企業における情報システム拡大の例

2.2.1　開発方法論 V1 の開発と拡充

開発方法論 V1 は, フェーズドアプローチ, データ中心型アプローチ(DOA)を特徴にしている.

フェーズドアプローチとは，作業の手戻りを防止する目的でプロセスを「フェーズ」と呼ぶ単位分解して，それぞれ作業の前後にイニシエーション(計画)とターミネーション(レビューと承認)を反復する方法である．イニシエーションは，計画重視のアプローチで，作業の漏れと無駄を排除する効果がある．ターミネーションは，作業の完成度や品質を評価して手戻りを防止する効果がある．フェーズは，さらに「ステップ」「作業」に分解される．開発作業は WBS で詳細化していく．成果物対応に必要な作業を詳細化して階層構造として体系化する．ソフトウェア開発で必要な WBS を標準テンプレートとして揃えておき, プロジェクトの特性に合わせて選択できるようにしている．図 2.2 に開発方法論 V1 の作業構造を示す.

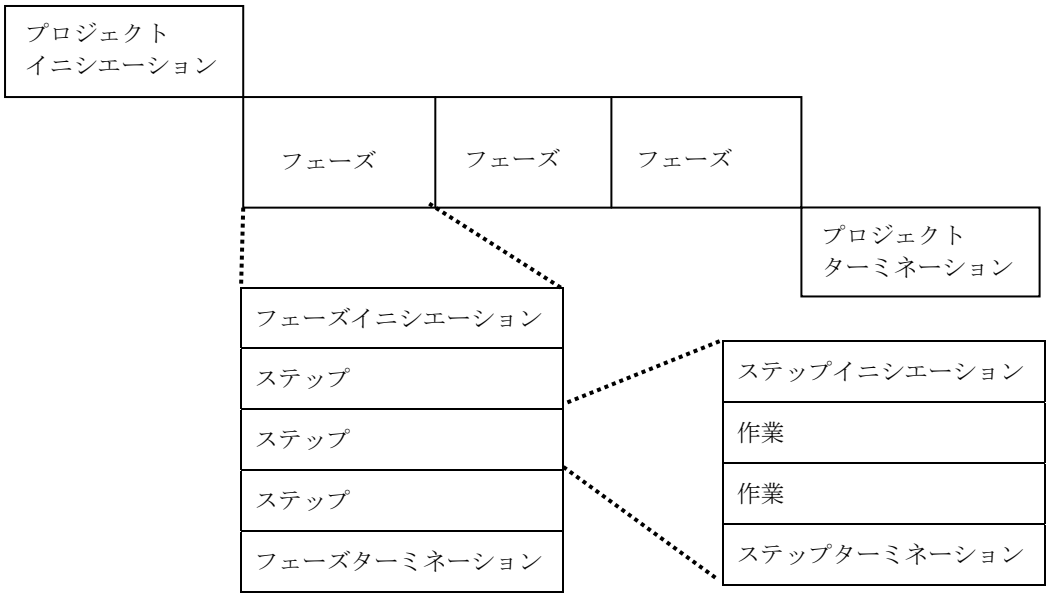


図 2.2　開発方法論 V1 の作業構造

データ中心型アプローチ DOA とは，データを基幹情報システムの重要な資産であると考え，共有資源として扱う開発技法である．DOA は「企業活動で用いているデータは基本的に安定しており, データを中心にした設計により保守性の高い情報システムが開発できる」という概念がベースになっている．データは，プログラムや DB などの情報資産と比較して安定した情報資産であり，その中に業務規則(業務ルール)が格納されている.

（1）開発方法論 V1

開発方法論 V1 は，標準プロセスとして標準開発手順 SPDS(Standard Procedure to Develop System)とプロジェクト管理手順がある．開発技法は，構造化技法(分析 SA，設計 SD，プログラミング SP)とデータ分析技法が中心である．また，日本語の仕様記述を標準化する構造化日本語を開発した．

標準開発手順は，ウオータフォールプロセスモデルによるソフトウェア開発標準プロセスである．ソフトウェアライフサイクルコストの低減，エンドユーザ参加型開発の実現，プロジェクト管理の効率化，DOA による情報資源管理の実現を目標にしている．開発作業は，「フェーズ」の単位に分割され，開発プロセスの上流から「システム分析」「システム計画」「システム設計」「プログラム設計」「プログラム作成」「テスト」「移行」「運用・評価」の 8 フェーズがある．図 2.4 に標準開発手順を示す．フェーズは，更に「ステップ」にブレークダウンされる．図 2.4 では，「システム分析」フェーズは A0 から A4 まで 5 ステップから構成されているのを示している．標準手順マニュアルには，作業の内容，書き方，入力情報とワークシートが書かれている．標準手順は，上流工程を重視したプロセスになっており，エンドユーザなどプロジェクト関与者の役割と責任範囲を明確にしている．

構造化分析技法は，Gane/Sarson の DFD 記法を採用した．図 2.3 に DFD の記述例を示す．

DFD により以下の効果が期待できる．

・業務仕様を図形表現できるので容易に記述できる．

・エンドユーザとエンジニアのコミュニケーションの円滑化が実現できる．

・トップダウンによる階層構造で分析するので，全体仕様と詳細仕様が把握できる．

・「もの」(データやファイル)と「できごと」(処理)を区分したモデル表現ができる．

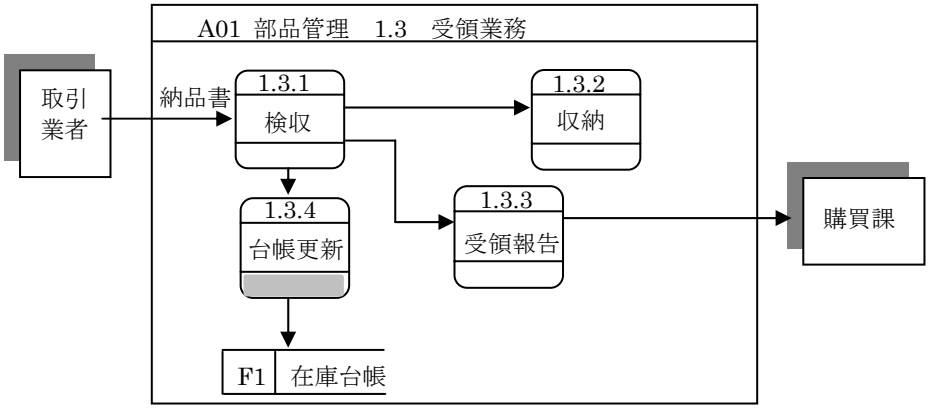


図 2.3　DFD の記述例

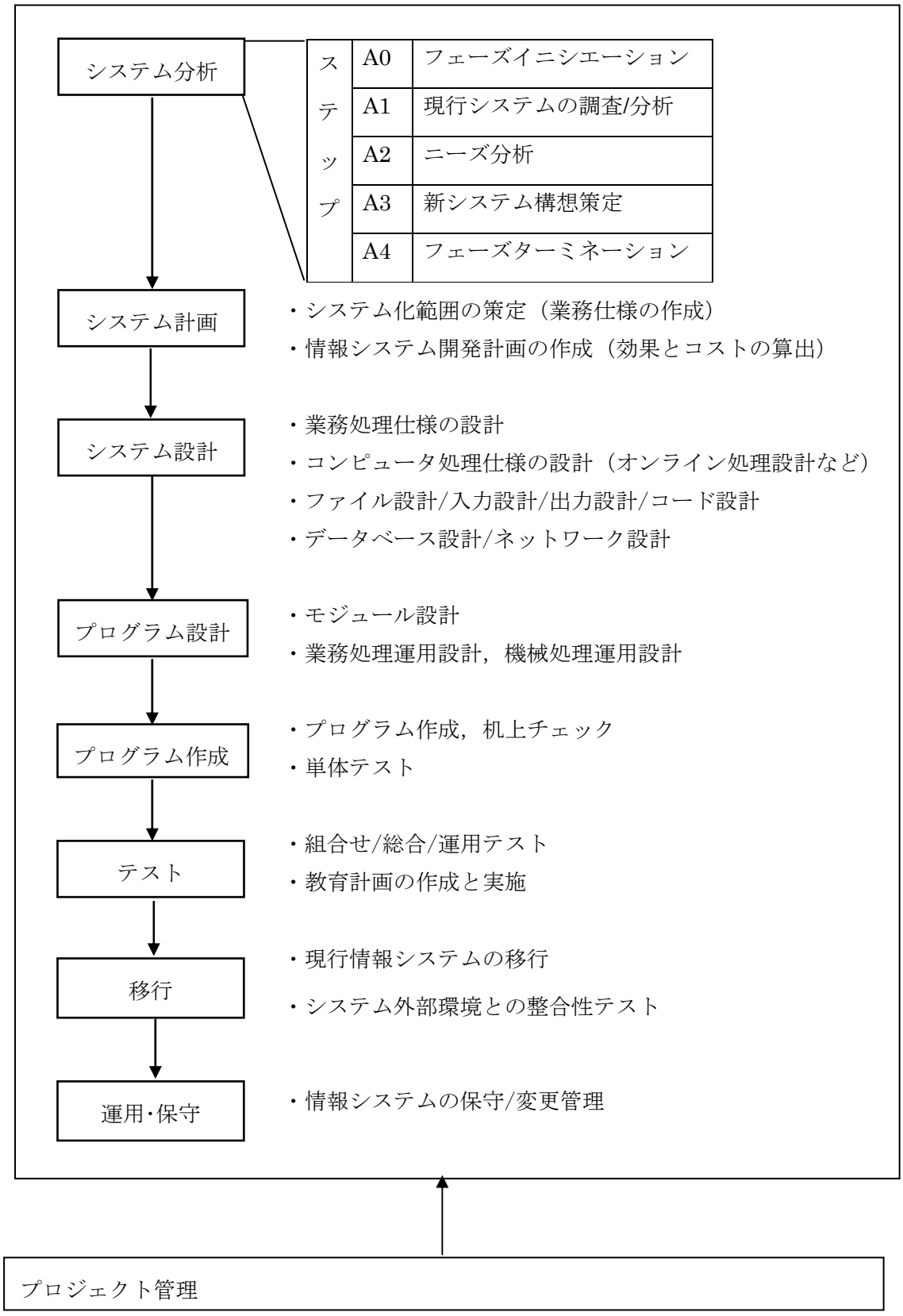


図 2.4　開発方法論 V1 の標準手順

構造化分析技法は，DFD による業務仕様の図形記述に加えて文章による補足説明書がある．日本語の記述技法として，曖昧な表現の防止と段落付けによる論理構造の明確化を図る目的で構造化日本語を開発した．構造化日本語は，「曖昧さのない記述」「段落付けによる文章の構造化」を基本的な考え方にしている．曖昧さのない記述では，記述してよい文章の構造を「連続」「選択」「繰返し」の３構造に制限し，記述してよい言葉を決めた．記述してよい言葉とは，指示内容の明確な他動詞，データ分析で名称が標準化された名詞，定められた修飾語，接続詞である．例えば，記述禁止の曖昧な表現は，動詞では「処理する」「修正する」，修飾語では「大部分を～」「大半を~」「速いときは」などである．

図 2.5 に構造化日本語の記述例を示す．従来の記述では，「支払報告書を作成する」の次に何をするのか曖昧である．また残高が 1000 万円以下なら何をすればいいのか，どの条件の場合に「残高を定期預金合計に加える」のかが不明確である．構造化日本語の記述では，これらの曖昧性が解消されている．

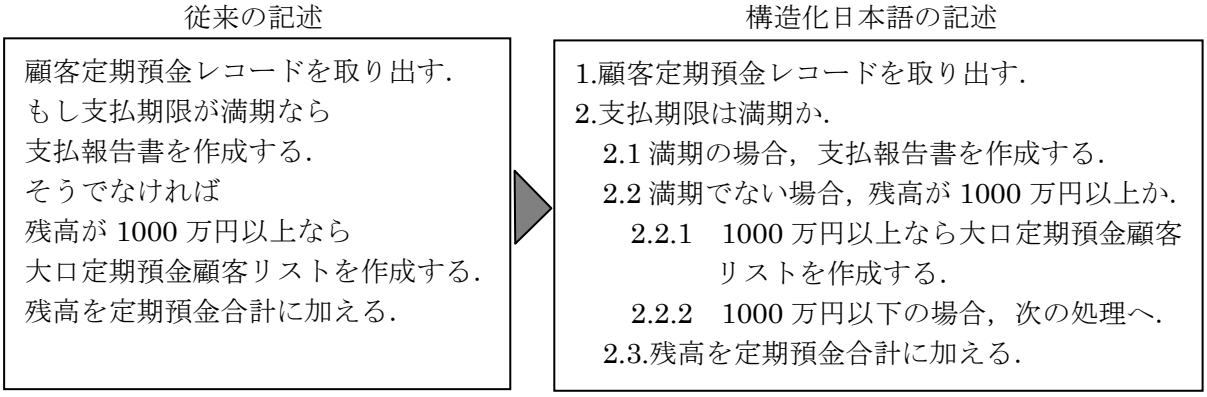


図 2.5　構造化日本語の記述例

（２）開発方法論の拡充

開発方法論は，1980 年に V1 が開発された以降も新しい開発プロセスや生産技術の発展と共に継続して改良してきた．開発方法論の拡充を表 2.1 に示す.

V2 では，ソフトウェア開発プロセスの国際規格 SLCP-JCF94 及び SLCP－JCF98 に対応して作業項目・用語を見直して標準手順基本作業フレームワークを設定した．この他に，「ステップ」の作業を，開発を担当する技術者の種類に対応させて３つのカテゴリに開発作業を分類した.

・ビジネス：利用者の業務要求から，システム化した業務運用方法を決定し，稼動させる作業
（アプリケーションエンジニア及び利用者）
・情報システム：稼働環境を構築し，稼働後のシステム運用を決定する作業
（テクニカルエンジニア，アプリケーションエンジニア）
・アプリケーションプログラム：業務処理ソフトウェアの開発
（プロダクションエンジニア）

V3 では, プロジェクトの関与者を顧客･ソリューションベンダ･ソフトウェアベンダに分けて, V2 で設定したカテゴリ別に並行作業が出来る標準手順を開発した．「ビジネス」「情報システム」「アプリケーションプログラム」のステップ作業を並行で開発するが，アプリケーション設計の結果をシステム基盤の設計や業務詳細設計にフィードバック出来るように，開発プロセスのモデルを W 字プロセスモデルにしている．V3 の標準手順を図 2.6 に示す．V3 では，テストフェーズを拡充している．テストフェーズにおける責任範囲(顧客とベンダ)，テスト環境(開発環境と顧客環境)，顧客との契約方式(請負契約と準委任契約)を明記している．テスト環境の顧客環境とは，顧客から提供される本番に準ずるテスト環境で，本番環境の場合もある．通常，連動テストまではベンダ主体で開発環境を使ってテストするが，システムテストからは，顧客主体の作業で顧客のテスト環境でテストする．ベンダは顧客作業を支援する．

表 2.1　開発方法論の拡充

フェーズ	V1	V2	V3
年代	1980 年代	1990 年代	2000 年代
ソフトウェア開発プロセス	フェーズドアプローチ WBS による体系化 データ中心型アプローチ(DOA)	スパイラルアプローチ 共通フレームワーク (SLCP-JCF94/98 対応)	W 字モデル
開発技法	構造化技法 データ分析技法 部品開発/利用技法	オブジェクト指向技法 リポジトリ利用技法 リエンジニアリング技法	コンポーネント指向技法 UML FP 見積技法

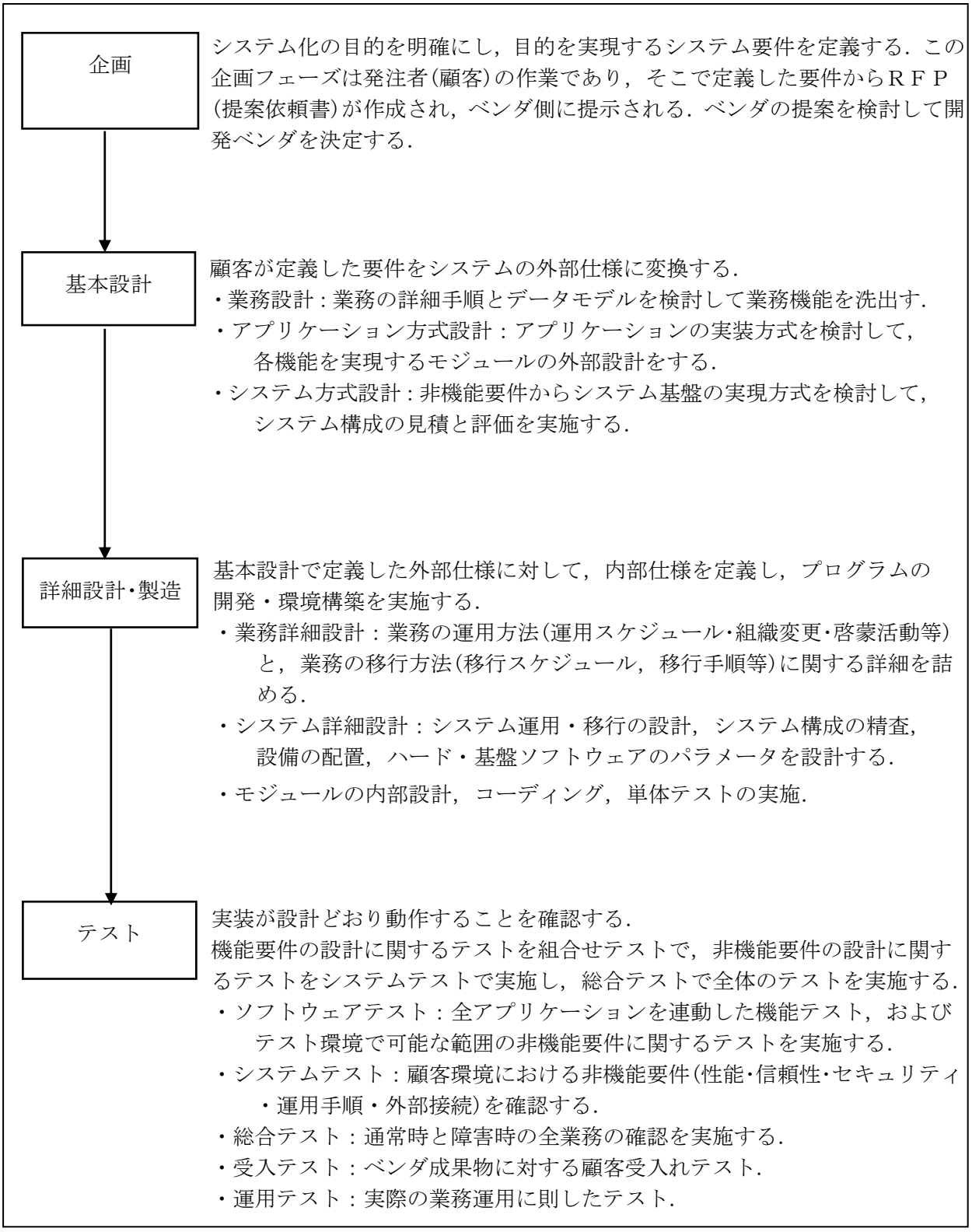

企画
システム化の目的を明確にし，目的を実現するシステム要件を定義する．この企画フェーズは発注者(顧客)の作業であり，そこで定義した要件からＲＦＰ(提案依頼書)が作成され，ベンダ側に提示される．ベンダの提案を検討して開発ベンダを決定する．
基本設計
顧客が定義した要件をシステムの外部仕様に変換する．
・業務設計：業務の詳細手順とデータモデルを検討して業務機能を洗出す．
・アプリケーション方式設計：アプリケーションの実装方式を検討して，各機能を実現するモジュールの外部設計をする．
・システム方式設計：非機能要件からシステム基盤の実現方式を検討して，システム構成の見積と評価を実施する．
詳細設計･製造
基本設計で定義した外部仕様に対して，内部仕様を定義し，プログラムの開発・環境構築を実施する．
・業務詳細設計：業務の運用方法(運用スケジュール･組織変更･啓蒙活動等)と，業務の移行方法(移行スケジュール，移行手順等)に関する詳細を詰める．
・システム詳細設計：システム運用・移行の設計，システム構成の精査，設備の配置，ハード・基盤ソフトウェアのパラメータを設計する．
・モジュールの内部設計，コーディング，単体テストの実施．
テスト
実装が設計どおり動作することを確認する．
機能要件の設計に関するテストを組合せテストで，非機能要件の設計に関するテストをシステムテストで実施し，総合テストで全体のテストを実施する．
・ソフトウェアテスト：全アプリケーションを連動した機能テスト，およびテスト環境で可能な範囲の非機能要件に関するテストを実施する．
・システムテスト：顧客環境における非機能要件(性能･信頼性･セキュリティ・運用手順・外部接続)を確認する．
・総合テスト：通常時と障害時の全業務の確認を実施する．
・受入テスト：ベンダ成果物に対する顧客受入れテスト．
・運用テスト：実際の業務運用に則したテスト．

図 2.6　V3 の標準手順

2.2.2　データ中心型アプローチ DOA

データ中心型アプローチとは，データ項目を共有資源として扱う開発技法である．データ項目とは，「社員番号」「給与」「氏名」など業務や基幹情報システムで使われているデータで，画面，帳票，ファイルやレコードを構成する最小の単位である．データ項目は，設計情報の中では一意に決まり，しかも安定しているので，はじめに共有資源として管理してシステム開発をすれば安定した基幹系情報システムが建設できる．

DOAの目的は，企業/組織におけるデータの共有化とソフトウェアの重複排除である．

データの共有化とは，データを共有資源として共通の名前で企業/組織全体で共有し，活用することである．

ソフトウェアの重複排除とは，データが固有の処理を持つという特性に着目して，データとデータ固有の処理をカプセル化して，共通プログラムやデータ処理部品にすることでプログラムの重複を排除することである．同じデータを処理するプログラムが複数存在すると，保守作業を増大させ，品質悪化の要因にもなる．

データ分析の手順を図2.7に示す．

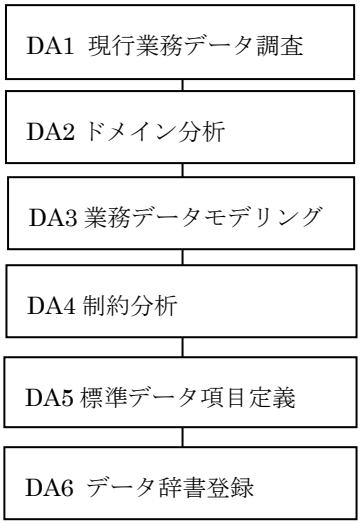


図 2.7　データ分析の手順

（1）DA1 現行業務データ調査

現行業務/現行情報システムの帳票，画面，DB，ファイルのデータ項目を抽出して，名称，形式，意味を調査する．抽出したデータ項目の名称を統一する．永年の企業活動の間に，「異名同値」である通称，略称などが存在する場合が多い．また反対に同じ名称だが部門で意味の異なる「同名異値」の名称がある場合もある．名称を統一して意味も統一する．

（2）DA2 ドメイン分析

データの表現方式に着目して，意味の持つ制約条件を持つ値の集合(標準定義域)をデータ分析ではドメインと呼んでいる．ドメインは，データ項目間の共通の制約条件を持ち，プロジェクト開発の関与者が共通認識できるものである．例えば，「郵便番号」「電話番号」などはドメインになる．ドメイン分析の目的は，共通の制約条件の部品化によるソフトウェア開発と保守の効率化である．ドメイン分析では，名称，カテゴリ，制約条件を洗い出す．

ドメインの例を，以下に示す.

・ドメイン名称： 出荷数量

・カテゴリ　　： 数字２桁

・値制約　　： ０～５０

カテゴリは，業務カテゴリと共通の基本カテゴリがある．基本カテゴリを表2.2に示す.

表 2.2　基本カテゴリ

No	カテゴリ	説明文
1	識別コード	管理対象を識別するための文字列(値)であり，必ず個々の値の指し示す対象が決められている.
2	識別番号	個々の値と管理対象の対応があらかじめ決められていない．おもに時々刻々発生する出来事を識別するために使用する.
3	区分	コードの一種であるが，ある基準による物事の分類を明らかにする目的を持つ.
4	フラグ	ある特定の条件を満たすか否かを示す.
5	日付	ある特定の時点の年月日を表わす.
6	時刻	ある特定の時点の時刻を表わす.
7	期間	ある特定の時点から次の時点までの期間を表わす.
8	時間	ある特定の時点から次の時点までの時間を表わす.
9	金額	円，ドルなどの通貨の額.
10	数量	個数，重量などを表わす数値.
11	名称	物事の名称を表わす名詞または短い句.
12	記述	名称より長い文字列であり，句あるいは文に相当する.
13	その他	上記以外のその他の項目.

（3）DA3 業務データモデリング

業務データモデリングでは，リレーションの正規化とエンティティを定義してデータモデルを作成する．帳票や画面などのデータからエンティティ(実体)を抽出する．エンティティとは，データが属する集合体である．エンティティをデータ項目の組合せに分解し正規化して第３正規形のリレーションにする．正規化したリレーションから，業務データモデルを作成する．併せてリレーションを統合し，エンティティを定義してデータモデル図を作成する.

（4）DA4 制約分析

データの制約は, データ項目に固有のデータ制約(固有制約)とデータ項目関連のデータ制約(関連制約）がある．固有制約は，データの属性や，ドメインを規定する形式制約，データの取り得る値を規定する値制約, 導出元データから値を決定される導出制約がある. 導出制約は, 例えば「給与」では,「給与＝基本給＋残業単価×残業時間」となり, 業務ルールを意味している. 関連制約は, 複数のデータ項目の値により規定される制約で，例えば「顧客レベルがAクラスの顧客は，100万円までクレジットが利用できる」といった制約である．

制約分析では，ライフサイクル(LCP : Life Cycle Process)も抽出する．LCPはデータの発生から更新，消滅までのイベントである．「社員番号」の例では，「入社」で発生し，「退社」「退職」で消滅する．通常は「社員番号」の更新は，ありえないが，会社の合併などにより「社員番号」の更新があるかもしれない．

（5）DA5 標準データ項目定義

ドメイン, エンティティから標準データ項目を定義する. データ項目は,「実体名」-「属性名」-「ドメイン名」の構造で名称をつける．例えば，社員誕生日は「社員」-「誕生」-「日付」の名称になり，これをデータ項目辞書に登録する．

データ項目は, 属性(名称, 長さ, タイプなど), 制約(形式制約, 値制約, 導出制約など), LCP(生成，更新，消滅)によりにカプセル化される．カプセル化した標準データ項目は統合開発支援ツールのデータ辞書に登録する．

データ中心型アプローチの目的は，システム保守に柔軟に対応できる安定した基幹情報システムの構築とデータ処理部品によるソフトウェアの重複排除である．データ処理部品の開発については次節で述べる．

2.3　統合開発支援ツール EAGLE

統合開発支援ツールは，ソフトウェア開発プロセス全体を支援するツールである．標準開発プロセスに対応した支援機能がある．またデータ分析で標準化したデータ処理部品によりプログラムを自動生成する[61].

統合開発支援ツールは，1980 年代に開発され，開発対象や開発環境を変えながら，機能追加を図り 2000 年代まで拡充してきた．1980 年代に開発した初期の統合開発支援ツールを EAGLE1，1990 年代のツールを EAGLE2，2000 年代のツールを EAGLE3 と呼び，それぞれの機能を述べる．表 2.3 は統合開発支援ツールの推移である．開発対象は，EAGLE1 ではホストコンピュータの基幹情報システムであったが，EAGLE2 では C/S(クライアントサーバシステム)が追加され，EAGLE3 では Web システムになっている．EAGLE3 では，アプリケーションアーキテクチャーを，画面処理(P : Presentation)層，ビジネスロジック(F : Function)層，データベースアクセス(D : DataAccess)層の 3 層にした実装モデルの開発支援をサポートしている． 3 層モデルにより，スケーラビリティと可用性(Availability)を確保できる．

EAGLE3 開発環境は，EAGLE1，EAGLE2 ではホストコンピュータによる集中開発環境であったが EAGLE3 では Web システムによる分散開発環境に移っている．なお，1990 年代の終わりには WS(ワークステーション)の図形処理機能による設計支援機能を開発している．

表 2.3　統合開発支援ツールの推移

項目	EAGLE1	EAGLE２	EAGLE３
時期	1980 年代	1990 年代	2000 年代
開発環境	ホストコンピュータ	ホストコンピュータ ＋WS(ﾜｰｸｽﾃｰｼｮﾝ)	PC
開発対象モデル	1 層モデル	1 層/2 層モデル	1 層/2 層/3 層モデル
開発言語	COBOL，PL/I	COBOL，C，VB，PB	COBOL，Java，C，C++，VB，.Net(C#，VB，COBOL)
開発対象	ｵﾝﾗｲﾝｼｽﾃﾑ，ﾊﾞｯﾁｼｽﾃﾑ	ｵﾝﾗｲﾝｼｽﾃﾑ，ﾊﾞｯﾁｼｽﾃﾑ，C/S(ｸﾗｲｱﾝﾄｻｰﾊﾞｼｽﾃﾑ)	ｵﾝﾗｲﾝｼｽﾃﾑ，ﾊﾞｯﾁｼｽﾃﾑ，C/S(ｸﾗｲｱﾝﾄｻｰﾊﾞｼｽﾃﾑ)，Web ｼｽﾃﾑ

2.3.1 ソフトウェアライフサイクル一貫支援サポート

統合開発支援ツール EAGLE は，分析からテストまでの工程を支援している．表 2.4 に EAGLE の機能エンハンスを示す．EAGLE2 と EAGLE3 の機能は，それぞれ新たに追加された機能のみを示している．

表 2.4　統合開発支援ツールの機能

機能	EAGLE1	EAGLE２	EAGLE３
データ分析	データ辞書 ・名称，属性	データ辞書 ・制約	業務ルール辞書
詳細設計支援	・ファイル/レコード ・画面 ・帳票 ・画面遷移	・データフロー ・システムフロー ・ＤＢスキーマ ・コード	・データモデル ・オブジェクト指向 分析/設計 ・エンドユーザ向け 帳票作成
プログラム設計支援	プログラム定義		
プログラム作成支援	（次項で説明する）		
単体テスト支援	・テストコマンド生成 ・ソーステスタ	・テストケース生成 ・構造図(PAD)テスタ	・テストデータ生成

（1）データ分析

基幹系情報システムで使うデータ項目をデータ辞書に登録する．データの要素や相互関係を分析し，その名称，構造，意味などをデータ辞書に登録する．EAGLE2 ではデータ名称，属性に加えて制約の登録を実現した．制約とは，表現形式や正当性の判断条件，導出ルール，値の制限などである．この定義情報がデータ処理部品としてプログラム自動生成で使われる．

（2）詳細設計支援

DB，画面，帳票，ファイル/レコードの設計を支援する．定義情報は統合開発支援ツールのリポジトリで管理する．EAGLE2 では，データフローやシステムフローの設計支援機能をサポートしている．データフローからシステムフローを生成する機能がある[62]．EAGLE3 では，DB 情報のアクセス条件から DB アクセス部品(D 層プログラム)を自動生成する機能がある．

（3）プログラム設計/作成支援

プログラム設計では，プログラム構造とプログラム間インタフェースを決める．設計はできるだけ多くの再利用可能なソフトウェア部品を利用する．

統合開発支援ツールはプログラマが定義したプログラム仕様から，スケルトン，入出力部品，データ処理部品を使ってプログラムを生成する．プログラマは，プログラム固有の処理部分を追加コーディングしてプログラムを完成させる．追加コーディングは，木構造のチャートによる構造図エディタを利用する．構造図エディタは，ソースから構造図表示の逆変換もできる．

（4）単体テスト支援

統合開発支援ツールはソースコードを分析してテストツールが必要なテストコマンドを生成する．例えば，ファイル I/O シミュレーション，ブレイクポイントセット，演算結果の表示などである．プログラマは生成したテストコマンドに従って会話処理でテストデータを入力して単体テストを実行する．EAGLE2 では，テストケース生成機能を実現した．プログラマは，標準スケルトンに対応した標準テストケーススケルトンにテストケースを追加して完成させる．テストの網羅性を評価するテストカバレージ(C0/C1 メジャー)自動取得機能がある．

2.3.2 ソフトウェア部品再利用技術によるプログラム自動生成

ソフトウェア部品再利用技術の目的は，既存のソフトウェアを再利用してプログラム開発量を抑制することである[63].

ソフトウェア部品は，スケルトンと処理部品に分けられる．スケルトンとは，プログラムの制御構造を形態別に分類･整理してパターン化したものである．処理部品は，共通機能を部品化した機能部品とデータの制約条件から作成したデータ処理部品がある．機能部品は，情報システム一般の汎用部品(年齢計算など)と業務固有の業務部品(勤務時間計算など)に分けられる．ファイルや DB のアクセス処理やオンラインシステムのトランザクション制御を共通化した入出力部品も機能部品になる．図 2.8 にソースプログラム生成の流れを示す．ソフトウェア部品の開発では，EAGLE1 では標準スケルトンと機能部品を開発した．EAGLE2 では，データ項目辞書からのデータ処理部品を実現した．EAGLE3 では，業務ルール部品のサポートと 3 層モデルのプログラムを生成する機能を実現した．表 2.5 にソフトウェア部品再利用技術の推移を示す．

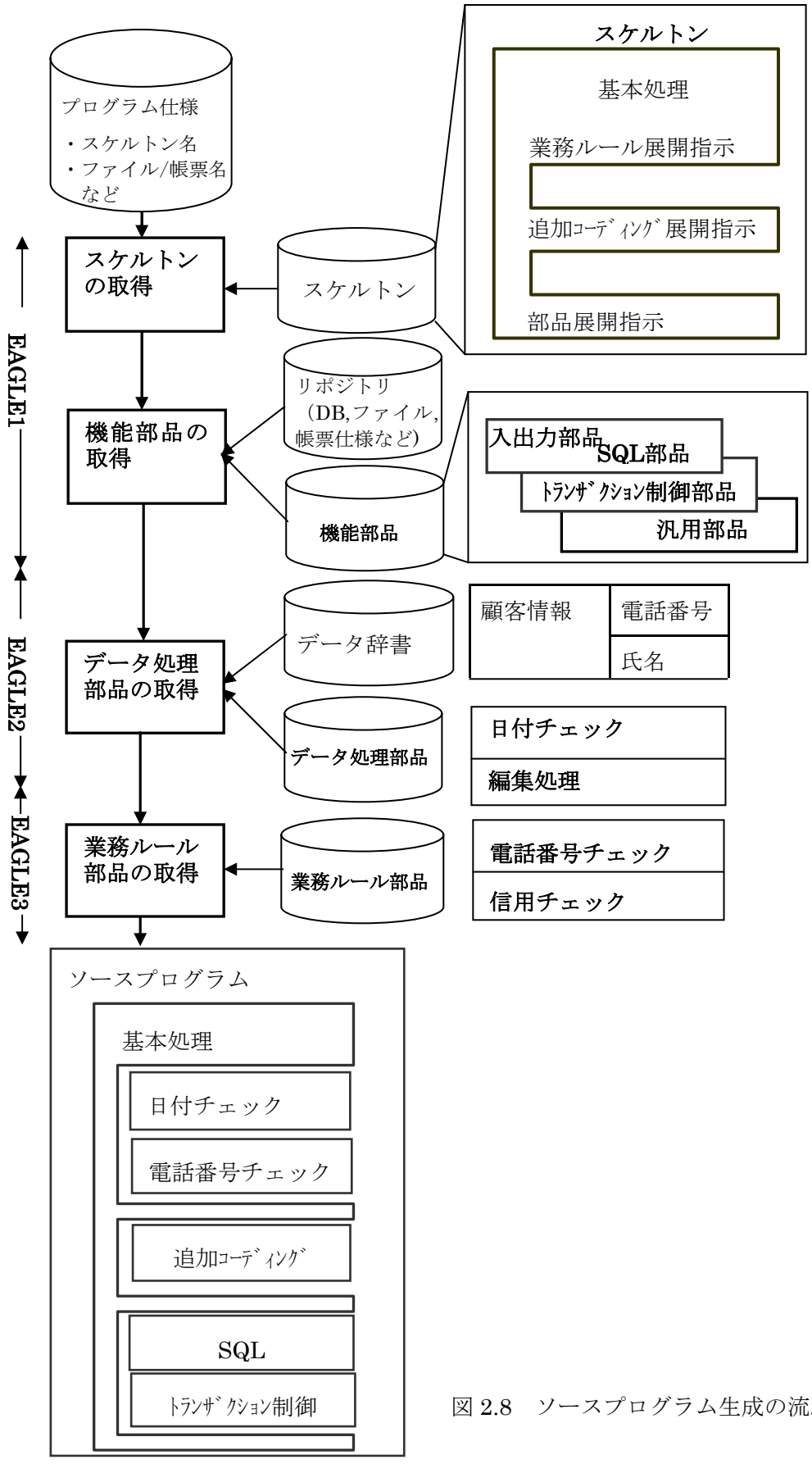


図 2.8　ソースプログラム生成の流れ

表 2.5　ソフトウェア部品再利用技術の推移

項目	EAGLE1	EAGLE２	EAGLE３
プログラム自動生成	スケルトンと機能部品の自動取込	データ処理部品自動取込	業務ルール部品自動取込 DB 設計仕様から DB アクセスプログラムの生成
標準部品ライブラリ	標準スケルトン（バッチ/オンライン）汎用部品	標準スケルトン(C/S 用) 標準データ処理部品	標準業務ルール部品
テストケース	なし	標準テストケース	—

（1）標準スケルトンの開発

EAGLE1 では，標準スケルトンの開発にあたり，プログラム構造をデータ構造の見地から体系化した．データ構造を，ファイル，レコード，データ項目の 3 レベルに分けて，それぞれの制御と処理を調査した．その結果，プログラムの構造を決める要因として，入出力ファイルの数とアクセス方式，レコードの処理条件が大きいことが分かった．その結果を反映してバッチプログラムでは 22 個の標準スケルトンを開発した[64].

ファイル変換･編集では「レコードの抽出」「照合」など 8 種類，ファイル更新では，マスタファイル数，アクセス方式別に 9 種類，帳票作成ではブレークキー数別に 5 種類である．オンラインプログラムの標準スケルトンは 9 種類である．スケルトンは完全なソースコードではない．プログラマはスケルトンに追加コードを挿入する．例えば，ファイル更新スケルトンは，ファイルオープンコード，レコード検索コード，レコード更新コードやファイルクローズコードが含まれている．このプログラムは大部分のファイル更新プログラムで適用が可能な構造になっている．プログラマはプログラムを完成させるためにスケルトンにレコード検索や更新のコードを挿入する．これらのスケルトンは最大 500-1000 行の長さである．

表 2.6 に標準スケルトンの数を示す．EAGLE2 では画面遷移プロセスなどオンラインプログラムのスケルトンをサポートした．

表 2.6　標準スケルトン

タイプ	スケルトンの数		標準スケルトンの例
バッチプログラム	EAGLE1	22	ファイル処理(変換，更新，併合)
	EAGLE2	31	帳票作成
	EAGLE3	37	DB(検索，更新)
オンラインプログラム	EAGLE1	9	問合せ，データ入力，DB 更新
	EAGLE2	45	画面遷移，画面処理，帳票処理，テーブル処理，DB 処理
	EAGLE3	21	３層モデルスケルトン

標準スケルトンは，プログラムの制御構造を標準化しているので，プログラマの能力に依存しない均質のプログラムが完成する．ユーザ追加コーディング箇所を局所化しているので，不良作り込みを防止できる．構造化プログラミングにより高い保守効率を実現している．

（2）標準データ処理部品の開発

EAGLE1 では, 日付計算, コード変換, エラー処理などの機能部品(235 個)を開発した. EAGLE2 では，データ項目のドメインを分析して，「日付」や「金額」などの共通のドメインに対する制約を標準データ処理部品として作成した[65]．データ処理部品には，入力編集･変換・チェック･出力編集などの処理がある．データ入力プログラムでは，データ処理部分の平均 40%がチェック処理と入出力処理であり，これらの処理の部品化が生産性向上には非常に重要であると考えた．
表 2.7 にデータ項目「社員誕生年月日」の例を示す．

表 2.7　データ項目「社員誕生年月日」

データ名称		社員誕生年月日
属性	データタイプ	整数
	長さ	6
	別名	社員誕生日
データ処理部品	データチェック	LX03：実在日チェック(西暦)
	入力編集	LX11：年月日変換(和暦→西暦)
	出力編集	LX21：年月日変換(西暦→和暦)

ドメインは「年月日」になり，データ処理は数字チェック，範囲チェック，妥当性チェックや暦変換（和暦⇔西暦）処理などがある．日本の企業では和暦を使用している企業が多く，入力データの和暦を, プログラム処理のために西暦に変換して, 再度帳表などの出力で和暦に戻す例が多い．
入力編集部品 LX11 によるソースコード生成は以下のようになる．

```
MOVE BIRTHDAY TO D06-I-YMDWA
CALL `LX06' USING D06-TBL
MOVE D06-YMD TO BIRTHDAY
```

更に，企業固有の制約条件がある場合は，必要なデータ処理部品を追加する．社員の年齢制限がある場合は，年齢計算と年齢制限チェック(例：社員は 65 歳以下)の部品を追加する．標準データ処理部品として，「名前」「時間」「日付」「金額」など 235 種類の部品を整備した．表 2.8 に日付チェック部品と編集部品の例を示す．

表 2.8　日付チェック部品と編集部品

日付チェック部品
実在日チェック（西暦）
実在日チェック（和暦）
過去日チェック（西暦）
未来日チェック（西暦）
年月日範囲内チェック（西暦）

日付編集部品
年月日変換（西暦→和暦）
年月日変換（和暦→西暦）
年月日算出（西暦年月日±日数）
期間算出（西暦年月日間日数）
月末日算出（西暦年月日）
通算週算出（年始～西暦年月日）

（3）業務ルール部品のサポートと 3 層モデルのプログラムの生成

業務ルールを部品として業務ルール辞書に登録して再利用する．業務ルールは，例えば「販売金額の計算式」「顧客の電話番号の実在チェック」である．

EAGLE3 では，3 層モデルアーキテクチャのプログラム開発を支援している．3 層モデルでは，画面処理 P 層，ビジネスロジック F 層，データベースアクセス D 層に分けて開発する．データとビジネスロジックの集中管理により再利用が促進される．また，疎結合なので変更影響範囲が局所化して保守性が高まる．図 2.9 は，データ分析で抽出した制約が，どの層に対応するかを示した図である．D層にある存在制約と参照制約は，データベースの追加･更新･削除に関連を持つエンティティ間での整合性を保持するための制約である．

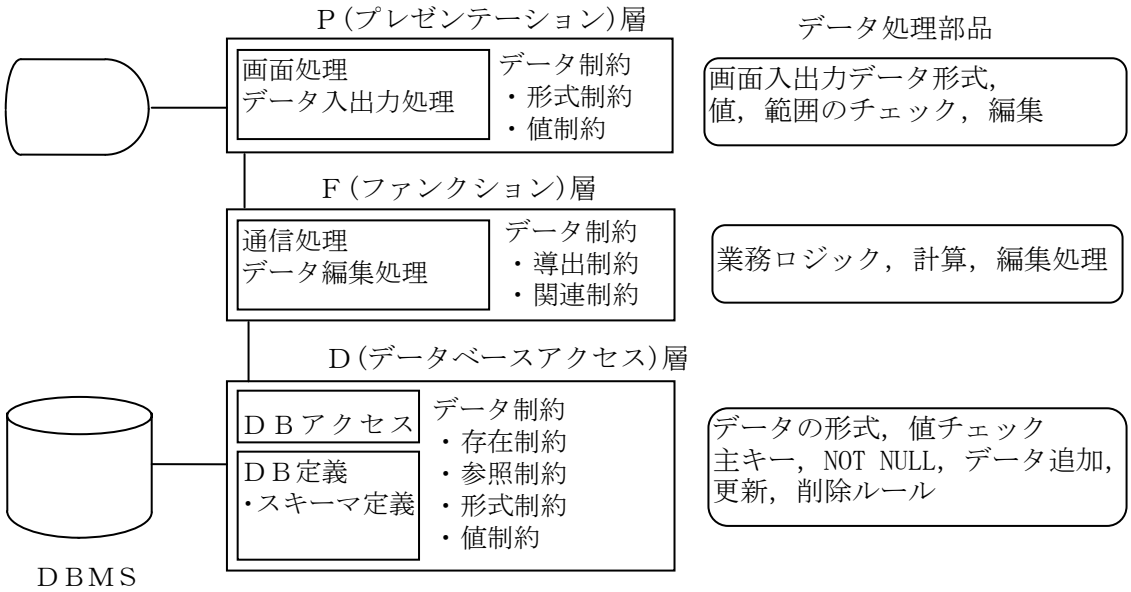


図 2.9　3 層モデルのデータ処理部品

2.4　統合開発支援ツールの評価

統合開発支援ツール EAGLE1，EAGLE2，EAGLE3 の各フェーズの実績データでソフトウェアの生産性と品質を分析する．ソフトウェア部品によるプログラム生成率，テストでの不良率，プログラマの習熟曲線の分析結果を述べる．

2.4.1　生産性評価

ソフトウェア開発の生産性とプログラム生成率は連動している．各フェーズにおける代表的な５プロジェクトを抽出して統合開発支援ツールの機能拡張と自動生成率の関連性を分析した．プロジェクトの概要を表 2.9 に示す．

表 2.9　プロジェクトの概要

フェーズ	プロジェクト	規模	開発言語	備考
EAGLE１	A	小	COBOL	
EAGLE２	B	小	COBOL	
EAGLE３	C	大	COBOL，HTML	実行基盤：Linux
	D	大	バッチ:COBOL, オンライン:Java，JSP	3 層モデルの開発
	E	大	COBOL	3 層モデルの開発

プログラムの自動生成の評価として生成率(%)を使う．生成率の定義を以下に示す．

$$\text{プログラム生成率} = \frac{\text{Lg}}{\text{Lt}}$$

Lg：統合開発支援ツールが生成したプログラムの量(ステップ数)

Lt：プログラム全体の規模(コメントを含むステップ数)

データ処理部品の効果をプロジェクト A とプロジェクト B で評価する．開発規模は，それぞれ 43 本，44 本の小規模開発である．両プロジェクトとも公共向けシステムで，相互のプログラムの流用はないが業務の類似性はある．次に EAGLE3 での生成率を評価する．プロジェクト C は流通業向けシステム(プログラム 400 本)である．プロジェクト D とプロジェクト E は，3 層モデルのアーキテクチャーで開発している．両プロジェクト共，公共向けの大規模開発(プログラム 2000 本以上)であるが，業務の類似性はない．

プログラム生成率をバッチプログラムとオンラインプログラムに分けて評価した．3 層モデルの開発では，3 層(P 層/F 層/D 層)別の生成率を計測した．表 2.10 にプログラム生成率の結果を示す．

表 2.10　プログラム生成率

<table>
<tr><td colspan="2">フェーズ</td><td>EAGLE 1</td><td>EAGLE 2</td><td colspan="3">EAGLE 3</td></tr>
<tr><td colspan="2">プロジェクト</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td></tr>
<tr><td rowspan="3">バッチ</td><td>帳表作成</td><td>69%</td><td>98%</td><td>62%</td><td rowspan="3">86%</td><td rowspan="3">72%</td></tr>
<tr><td>ファイル処理</td><td>78%</td><td>97%</td><td>74%</td></tr>
<tr><td>データチェック</td><td>50%</td><td>73%</td><td>77%</td></tr>
<tr><td rowspan="3">オンライン</td><td>問合せ，更新</td><td>74%</td><td>77%</td><td>72%</td><td>P 層： 0%</td><td>46%</td></tr>
<tr><td rowspan="2">データ入力</td><td rowspan="2">50%</td><td rowspan="2">72%</td><td rowspan="2">52%</td><td>F 層：44%</td><td>43%</td></tr>
<tr><td>D 層：:98%</td><td>99%</td></tr>
<tr><td colspan="2">平均生成率</td><td>68%</td><td>81%</td><td colspan="3">71%</td></tr>
</table>

プロジェクト A では平均生成率は 68%，プロジェクト B では平均生成率は 81%であった．このことによりデータ項目辞書からのデータ処理部品のプログラム自動生成による効果が生成率で 13%向上したといえる．過去において，処理部品は作成したプログラムから再利用可能部分を抜き出して作られた．設計者個人の視点で作成されたので品質や性能に課題があった．EAGLE 2 では，データ項目単位の処理を部品化してデータ項目辞書に格納した．データ処理部品は全てのプロジェクトで再利用できる．

オンラインはバッチに比べて生成率が低い．これはバッチが 22 個のスケルトンのうちどれかを改造なしで適用できるのに対して，オンラインは画面操作がプロジェクトで異なるのでプログラマの追加コーディングが入るためである．

EAGLE3 では，バッチプログラムの生成率が EAGLE2 に比べて低い傾向がある．これはバッチジョブの形態が変化していることによる．EAGLE2 までのバッチ処理はファイル処理が主体で，例えばシーケンシャルファイルの更新処理では，トランザクションファイルの入力→データチェック→ソート処理→マスタファイルのマッチング処理→エラー処理のようなジョブフローになり，それぞれの処理のプログラムを作成していた．EAGLE3 では，従来のバッチ処理に加えて DB 処理が増えてきている．ジョブフローの冗長性がなくなり大量データ処理の性能問題が改善する反面，プログラム制御のパターン化は困難になる．このため DB 突合せチェックスケルトンと DB フリースケルトンを開発した．DB フリースケルトンでは，D 層モジュールの呼び出しや，リラン処理の制御はスケルトンで用意されているがプログラムの全体制御は設計者が記述する．

プロジェクト C は，実行基盤がオープンソフト(OS：Linux，Web ブラウザ：Mozilla)による

Web システムの開発プロジェクトである．バッチプログラムの帳表作成とオンラインプログラムのデータ入力の生成率が低い．プロジェクト D とプロジェクト E は，オンラインプログラムを 3 層モデルで作成した例である．P 層と F 層のプログラム生成が低い．Web システムでは P 層プログラムは，画面ブラウザや画面遷移で多くの実現手段があり，P 層開発の標準化が課題である．プロジェクト D では，顧客指定の画面仕様のためプロジェクト固有の雛形を作成してプログラムを作成した．F 層プログラムの開発には，まだ業務ルール辞書の活用が不十分で今後の課題である．P 層と F 層のプログラム生成が低いのに対して D 層プログラムは，ほぼ完成に近い生成率である．

EAGLE3 における 3 プロジェクトの平均生成率は 71%である．Jones は，ソースコードの理論的再利用率は 75%と述べている[66]．この値と比較しても統合開発支援ツールの生産性は高い．

EAGLE によるプログラム生成率の報告がある．1 件は，SI ベンダによる地方自治体水道料金システムの例で，EAGLE1 を利用した生成率は 51%である[67]．もう 1 件は，EAGLE2 を利用した顧客による医薬品販売管理システムの開発例で，生成率は 73%である[68]．

統合開発支援ツールの生産性をプログラム自動生成率で述べたが，ここでは他のアプローチとの比較をする．COBOL や Java などの言語による開発に対して，古くは第 4 世代言語(4GL)や簡易言語があり，最近では Web システム開発用スクリプト言語などがある．

4GL とは，事務処理でよく使われる集計処理，照合処理などのプログラムコード群をマクロ化した言語である[5]．これらは COBOL などの言語に対して記述量が数分の一になるので，２倍から５倍の生産性があると言っているが，統合開発支援ツールでは表 2.10 で示したように 70%以上の生成率を達成している．これは換算すると 4GL などの生産性に相当するレベルである．4GL などは性能を要求される基幹系情報システムへの適用には限界がある．また，基幹系情報システムは，長期間の稼動が期待されている．その間には企業戦略や業務拡張への対応や情報新技術への対応などでシステム保守作業が継続する．この面においても 4GL などには限界がある．

2.4.2 品質評価

テストは，単体テスト，組合せテスト，システムテストがある．単体テストはプログラマが，プログラムがプログラム仕様を反映しているか検証する．組合せテストとシステムは設計者が，機能要件及び非機能要件(性能，信頼性など)を満たしていることを検証する．

対費用効果の面で後工程でのトラブルを抑えるために開発工程の初期段階で不良を発見するのは重要である．この観点から，プログラム作成工程での十分な単体テストを強く進めている．

品質評価では，EAGLE2 でサポートした単体テスト用テストケース作成支援機能による品質を分析した.テストケース生成機能の効果を 2 プロジェクトで比較した.2 プロジェクト共 EAGLE2 で開発したが，プロジェクト F は標準テストケースを使っていない．プログラマはテストケースを手作業で作成している．プロジェクト G は EAGLE2 を利用して標準テストケースを生成している．プログラマは半完成の標準テストケースにテストケースを追加して完成させる．両者とも地方自治体向けシステムで，同じ開発グループで開発している．規模は同じく 10KS(KS＝1,000Steps)である．表 2.11 に，2 プロジェクトの不良密度を示す．不良密度は開発工程で発見された不良数をプログラムサイズで割った値(件数/KS)である．ここでは単体テストと組合せテストでの不良密度を述べる．

$$\text{不良密度} = \frac{\text{Ft}}{\text{Lt}}$$

Ft：不良件数

Lt：プログラム全体の規模(コメントを含む KS(1,000Steps))

PCL(Program Check List)密度(件数/KS)は，プログラム規模 1 KS あたりのテストケース(PCL)の割合である．

表 2.11　不良密度

フェーズ	EAGLE2			
プロジェクト	F		G	
	PCL 密度	不良密度	PCL 密度	不良密度
単体テスト	37	3.7	53	4.8
組合せテスト	15	2.9	14	2.2
合計	52	6.6	66	7.0

プロジェクト G では単体テストでは不良が多いが，組合せテストでは不良が少ない．この結果から，標準テストケースの効果として不良の検出が，組合せテストから単体テストにシフトしたといえる．

この仮説を前提に，組合せテストでの不良分析をする．不良を 3 タイプに分類する．タイプ 1 は，単体テストで検出すべきレベルの不良である．例えば，入出力関連処理，ループ処理などである．タイプ 2 は組合せテストで検出する不良である．プログラムインタフェースや業務仕様などの不良である．タイプ 3 はその他の不良である．図 2.10 に組合せテストのプロジェクト F，G の不良分類を示す．

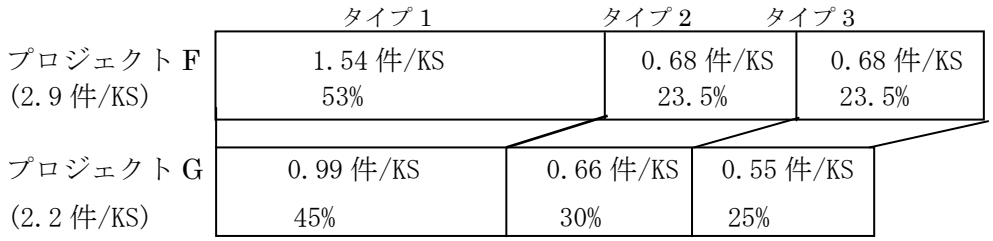


図 2.10　組合せテストの不良分布

プロジェクト G はタイプ 1 の不良が少ない(F が 53%で G は 45%)．タイプ 2 の不良密度は同じである(F：0.68 件/KS→G：0.66 件/KS)．プロジェクト G の結果から標準テストケースの効果が立証されている．単体テストで不良を検出するには最適なテストケース数が重要である．EAGLE では完全なソースコードを生成せず，プログラマが追加コーディングすると述べた．事前に十分にテストされたコードを使って生成した部分よりも手作業で追加した部分の不良は多い．

品質評価では，EAGLE2 の単体テスト用テストケース作成支援機能による効果を分析したが，プログラム生成率と品質の関連を報告した例がある[61]．EAGLE1 を利用した生成率 68%のプロジェクトと生成率 81%のプロジェクトを比較すると，生成率の高いプロジェクトの不良密度が，単体テストで 13%，組合せテストで 27%，総合テストで 90%向上した例が報告されている．プログラム自動生成の効果は，特に単体テストの不良密度に顕著に現れている．生成率 50%あたりを変異点にして不良密度は急激に下がっている．

2.4.3　習熟性評価

生産性と品質の向上には，統合開発支援ツールの機能拡充に加えて，開発者の教育も非常に重要であり，EAGLE1 の完成と同時に教育コースを設置した．教育の効果を測定する目的で，EAGLE1 と EAGLE2 の教育コースにおける受講者のプログラム作成時間を測定した．

（1）習熟曲線と成長率

習熟曲線とは，あるプロダクトを繰り返し生産するとき，作業者の技能習熟効果により，そのプロダクトの単位生産工数が漸減する現象を表現したもので，「生産量を倍増していけば，単位当たり加工時間は一定の比率で減少していく」という仮説に基づいている．例えば，あるプロダクト 1 個の生産時間を 100 時間とする．生産量を倍増するごとに 1 個当りの生産時間(単位加工時間)が 90%の比率で減少すると仮定すると，生産量を倍増して 2 個生産した場合の単位加工時間は 90

時間となる. 以下同様に, 4 個生産する場合は 81 時間となり, 8 個生産する場合は 73 時間となる. 生産量が倍になるごとに減少する単位加工時間の一定の漸減率を習熟率と呼ぶ. 習熟率 90%で生産量が倍増するごとに単位加工時間が減少する曲線を 90%習熟曲線と呼ぶ. ソフトウェアの場合も同じで, 同じようなプログラムを開発する場合, 経験と共に作成時間は減少する.

習熟曲線は, 表現モデルが提案されている[69]. このモデルをベースにしてプログラム開発習熟曲線モデルを設定した. 但し, 習熟率の替わりにプログラム開発経験回数をパラメータとした.

プログラム開発習熟曲線モデル(Y)

$$Y=aX^{-n}$$

Y : X 回目のプログラム開発平均時間（時間/KS）

a : 1 回目のプログラム開発平均時間（時間/KS）

X : プログラム開発経験回数

n : 効果係数

プログラム開発習熟曲線モデル(Y)から, プログラム開発経験回数の効果を示すモデルとして成長率モデル(E)を設定した. Y_Xは$Y_X=aX^{-n}$から, Y_{2X}は$Y_{2X}=a(2X)^{-n}$から導出した値である.

成長率モデル(E)

$$E=\frac{Y_{2X}}{Y_X}$$

E : X 回目と 2X 回目のプログラム開発平均時間の比率

Y_{2X} : (2X)回目のプログラム開発平均時間（時間/KS）

Y_X : X回目のプログラム開発平均時間（時間/KS）

成長率は, ふたつの開発時間の割合を表現している. 例えば, X=3 の成長率 E が 80%なら, 6 回目(X=6)の開発は 3 回目の 80%の時間で開発できる.

（2）データの収集

プログラム開発習熟曲線と成長率を計算するため，EAGLE の教育コースからデータを収集した. 受講者は, 開発経験のない新人である. EAGLE1 の教育コース(以下, 教育 1 と言う)と EAGLE 2 の教育コース(以下，教育 2 と言う)のデータを比較した．2 グループ間ではプログラム作成能力の大きな差はなかった．4 週間で EAGLE を使って 400 から 600 ステップの COBOL プログラムを作成した．プログラムはバッチプログラムで相互に類似している．教育コースの期間と受講者数を以下に示す.

・教育 1：期間 1982 年 6 月―1983 年 6 月，受講者 36 名

・教育 2：期間 1990 年 9 月―1991 年 3 月，受講者 90 名

収集したデータを以下に示す.

①プログラム開発の各作業の時間

・ プログラム仕様書作成からコンパイル完了まで

・ テストケース作成とテスト

・ ドキュメンテーション

②プログラムステップ数/テストケース数

③プログラム開発経験回数

（3）習熟曲線の評価

図 2.11 に教育 1 と教育 2 のプログラム開発習熟曲線(Y)を示す. 1 回目のプログラム開発時間を比較すると教育 2 では教育 1 と比べて 77%になっている．2 つの曲線の違いは EAGLE 1 と EAGLE 2 の機能差による. このことから教育 2 における未経験者の一定期間でのコード製造量では生産性は教育 1 に比べて 1.3 倍(77%)になっているのが分かる. 教育 2 のカーブの落ち方が急であり，すぐに低いレベルに到達している．10 回目の開発では開発時間の差は 2.3 倍まで増加している(44%).

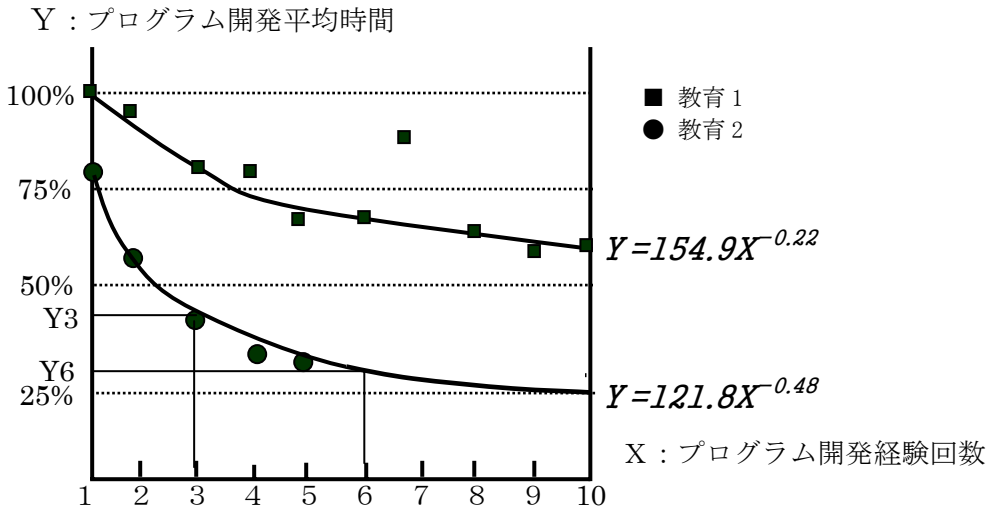


図 2.11　プログラム開発習熟曲線

表2.12に教育1と教育2のプログラム開発作業の3回目(X)と6回目(2X)を比較した成長率(E)を示す．成長率Eは，Y_6/Y_3の値である．プログラム開発作業の合計成長率は，教育1で86%，教育2で72%である．各作業では，教育１と教育２では9%～17%の成長率の差が出ている．

表 2.12　成長率(X=3)

プログラム開発作業		教育１	教育２
成長率(E)	プログラムコーディング	86%	69%
	テストケース設計＆単体テスト	83%	74%
	単体テストのみ	-	83%
	ドキュメンテーション	86%	72%
	合計	86%	72%

図 2.11 から，教育 2 ではプログラム開発習熟曲線の落ち込みが急で，早く低レベルに到達して安定するのが分かる．教育２では 4 回と 5 回のプログラム開発の後に安定レベルに達している．しかし教育１では 8 回と 9 回目の後にも開発時間の減少が見られる．これは大部分のプログラマが EAGLE1 に比べて EAGLE2 の方が開発経験は少なくてもプログラム開発技術を習得できることを意味している．

図 2.12 は，教育 2 でのテストケース設計と単体テストの平均時間の習熟曲線である．教育１の単体テストデータがないので両者を比較することはできないが，テストケース設計作業と単体テスト作業の習熟の傾向が分かる．図 2.12 から，単体テスト単独の作業は習熟効果が小さいので，テストケース作成時間の習熟効果が大きいのが分かる．テスト機能の拡充が今後の課題である．

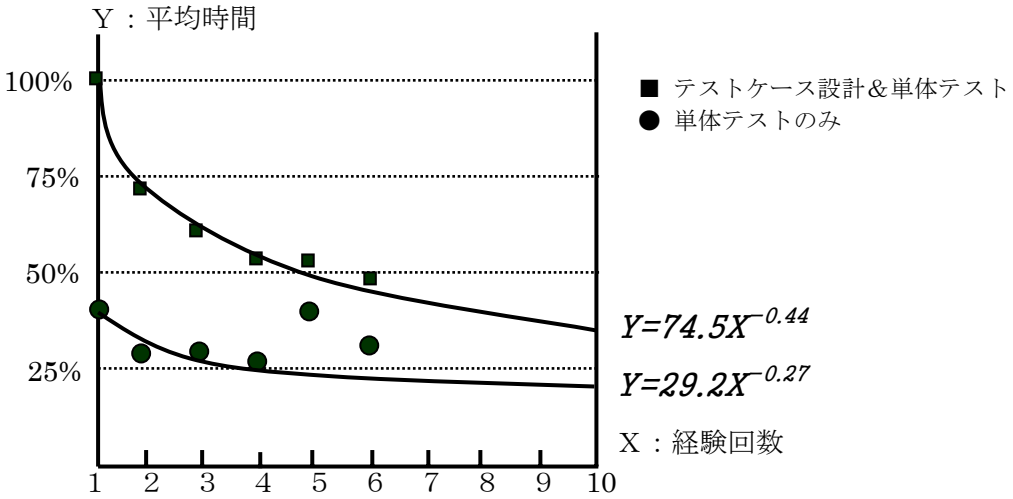


図 2.12　EAGLE2 テストケース設計と単体テスト習熟曲線

EAGLE による教育では習熟度を上げることができた．その要因を以下に述べる．

・図 2.11 に示すようにプログラム生成率の増加に比例してプログラム開発時間は減少している．DB アクセス部品などにより，プログラマは DB スキーマの詳細を知る必要がない．その結果，残った追加コーディングのプログラムはシンプルになっている．

・プログラムの品質は，データ処理部品などによるプログラム生成により確保できる．教育の目的は，個人の技術に依存したプログラム開発でなく，常に開発組織が一定の品質を確保し続けることである．EAGLE 教育コースにより，この目的は達成していると考えている．

・EAGLE2 の対話処理機能拡充の効果．例えば，開発作業で必要な作業操作はメニューから選択するので，受講者は覚えることなく早く習得できる．

・教育 2 ではコースカリキュラムとテキストを改善している．例えばサンプルプログラムが教育 2 のテキストには載っている．教育は専任の教育部門で担当しており，教育経験やノウハウは教育部門に蓄積されている．

2.5 結論

本章では，ソフトウェア開発プロセスモデルと開発技法を標準化した情報システム開発方法論と，それをサポートする統合開発支援ツールを提案した．開発方法論は，開発手順，設計ドキュメント，開発基準，開発技法を体系化しており多数の開発要員が参加する統合開発支援ツールの開発では有効である．SIベンダだけでなく，関連会社(SIベンダの子会社)，協力会社，発注者である顧客にも展開できる．また，統一した開発方法論を維持することで，ソフトウェアライフサイクルプロセス(SLCP)の標準化やUMLなど新しい開発技法の標準化に追随していくことが可能である．

統合開発支援ツールは，プログラム生成率，不良密度，教育における習熟率から効果を分析した．標準スケルトンと処理部品のエンハンスによりプログラム生成率が 20%向上した．EAGLE2 の標準テストケース開発により単体テストでより多くの不良が検出できた(標準テストケース適用で 4.8 件/KS，手作業で作成したテストケース適用で 3.7 件/KS)．教育効果では，EAGLE のエンハンスによりプログラム開発習得効果がでた．例えば，成長率が 86%から 72%になった．またプログラム開発経験回数が 4 回か 5 回で習熟率が上がることが分かった．

今後の課題としては，情報システム開発方法論ではアジャイル開発など反復型開発プロセスの基幹情報システム開発への適用研究，超上流と呼ばれる発注者の要求定義フェーズのサポートと開発要件の抽出･整理技法の研究，テスト技法の研究がある．

統合開発支援ツールでは，オブジェクト指向コンポネントによるプログラム生成，テストの自動化，信頼性や性能など非機能要件の設計支援機能の拡充がある．複数のスケルトンと部品を組合わせて，ある特定分野のコンポネントを開発して基幹情報システム開発に再利用する．テストでは，プログラム静的解析によるテストの上流化や組合せテスト以降のテストデータ自動生成，機密保護などからのセキュリティテストの自動化などの研究がある．

統合開発支援ツールの効果分析では，1980 年代から長期間にわたり実績データを蓄積してきた．継続的にデータを収集するのは重要である．

第 3 章　リバースエンジニアリングによる業務仕様理解支援

3.1　緒言

基幹系情報システムの構築に際し，事業形態が変らない限り現行業務仕様の継承は重要であり，既存ソフトウェア(レガシーシステム)を調査して業務仕様を理解する作業に多くの工数をかけてきた．この業務仕様理解を支援する技術として，従来からリバースエンジニアリングの研究が行われてきた．レガシーシステムの再文書化，設計復元，業務ルールの抽出などにより業務仕様を理解するが，それを支えるのは，相互参照技術，プログラム表現技術，プログラムスライス技術である[35].

相互参照技術は，影響波及解析でも使われる技術で，プログラムと設計仕様(画面，帳票，DB，ファイル)のデータ項目の依存関係，プログラムの特定データ項目の依存関係，プログラム間やプログラムと副プログラムとの依存関係を相互参照表(cross reference)で表現する．ソースコードを解析して，Web 画面にソースコード表示をすると共にプログラム関連図や GOTO 文・ルーチン呼び出し文の呼び出し先のソースコードを同時に表示して影響範囲の調査を支援する[39].

この他にレガシーシステムでは，永年の保守作業によりソースコードのカット&ペーストによるコードクローンが存在する場合が多い．コードクローンとは同一または類似したコードの断片で，多くは他のプログラムから「コピー」されて組み込まれている．このコードクローンを検出・計測して，クローンの統合化/部品化によりプログラム規模の最適化を図る研究が報告されている[70].

プログラム表現技術は，ソースコードの表現を改良してプログラムの理解を支援する技術で，ソースコードのレベルで表現する機能と，図形など表現方法を変える機能がある．ソースコードレベルの表現機能は，見出しや字下げ，制御の流れの表示，実行しないコードやデータ項目の表示などがある．図形レベルの表現機能は，ファイル/レコード仕様書，プログラム処理概要図，木構造などの構造化チャート，データモデル図，プログラム構造図などに変換して表示する．プログラムとジョブ制御情報からプログラムの流れであるジョブフロー図を表現する[71].

プログラム表現だけでなく非構造のプログラム自体を構造化する再構造化技術も提案されている[72]．また切り出したソースコードを処理機能は変えることなく仕様理解や保守作業を効率化するためにプログラムの内部構造を変化させるリファクタリング技術がある．

プログラムスライス技術は，プログラム内の命令間の依存関係を明らかにする技術である[73][74][75]．テストやデバッグ，コード最適化などの目的の他にプログラム理解や影響波及解析で使われる．プログラムスライスは，命令間のデータ依存関係と制御依存関係を逆向きにたどる

ことにより，特定の変数を計算するステートメントのサブセットを抽出する技術である．有効なソースコードの集合を抽出するため，ループ文や配列処理などを含むすべてのソースコードの解析が必要になる．プログラムスライスでデータ依存関係と制御依存関係を解析して，その結果を言語に依存しないデータモデルに変換する手法が提案されている[76].

構造化されていないプログラム(スパゲティプログラム)を対象にして手続き読み出し命令と手続き本体に関するプログラム依存グラフからスライスを求める手法が提案されている[77].

多くのリバースエンジニアリングの研究は，データフローの解析やデータ/制御の依存関係に着目して解析しているが，本章では，データ中心型アプローチの考え方に基づいたリバースエンジニアリング DORE(Data Oriented Approach Reengineering)を提案する．プログラムのデータ項目に着目して，ソースコードのステートメントを２項オブジェクトに変換・整理して，業務ルールを抽出する[78][79][80]．２項オブジェクトとは，ステートメントを一対の表現形式まで分解したものである．データ項目には，「属性」「ライフサイクル(LCP)」「制約」の３要素からなる情報がカプセルとして格納されている．制約とは，データの存在と条件を規定するものであり，これが業務ルールとなる．DORE による業務ルールの抽出により，業務仕様の理解支援,設計仕様書の生成,データの標準化と部品化の支援，オブジェクトの抽出を支援する．

以降，3.2 節で２項オブジェクトの考え方を述べ，3.3 節で業務ルールの抽出方式を述べる．3.4 節で実プロジェクトでの評価･分析を述べる．

3.2　２項オブジェクト

3.2.1　データに存在する業務ルール

既存のソースコード(言語：COBOL)から，データ項目を中心に設計情報を抽出する．これらのデータ項目の制約をプログラムから抽出する．「2.2.2 データ中心型アプローチ DOA」でのべたように，制約とは，データの存在と条件を規定するものでデータ項目固有の制約，データ項目間の関連による制約，データ項目の集合であるグループ単位の制約がある．制約には，形式制約，値制約，導出制約，関連制約がある．このうち業務ルールに該当するのは値制約，導出制約，関連制約である．例えば，「残業時間≦100」という値制約の場合は「残業時間は 100 時間以内である」という業務ルールになる．「残業時間＝通常残業時間＋深夜残業時間」という導出制約は，残業時間算出式の業務ルールになる．

プログラムの特定データ項目の処理(プロセスオブジェクト)は，基本操作(２項オブジェクト)と

制御操作(制御オブジェクト)に分けられる．制御オブジェクトは，条件式で表現された２項オブジェクトと判定結果による操作を表現した２項オブジェクトの集約で表現される．その役割は，制約の表現と制約の検証と基本操作の指示である．図 3.1 にデータ項目，処理プロセスと２項オブジェクト/制御オブジェクトの関係を示す．ソースコードに定義されているデータ項目は，DB/ファイル定義データ，画面定義データ，トランザクション定義データとプログラム内独自定義データに分類される．データ項目は複数のプログラムで使われている．プログラム内プロセスとは特定データ項目の処理の集合である．制御オブジェクトは，一般的には IF 文で表現されている．

既存のプログラムからコードで表現されている基本操作と制御操作を抽出し，特定のオブジェクトへと変換・整理する．即ち，プログラム中にコード(IF 文など)として埋めこまれている判定手段やチェック処理などの制約を分解して，データごとに固有の制約としてカプセル化する．複数のオブジェクトに対して固有の制約は，複数のオブジェクトによって構成する集約オブジェクトの制約として定義する．カプセル化によりデータ更新情報処理の重複排除，修正影響検出の容易さ，網羅性の保証ができる．

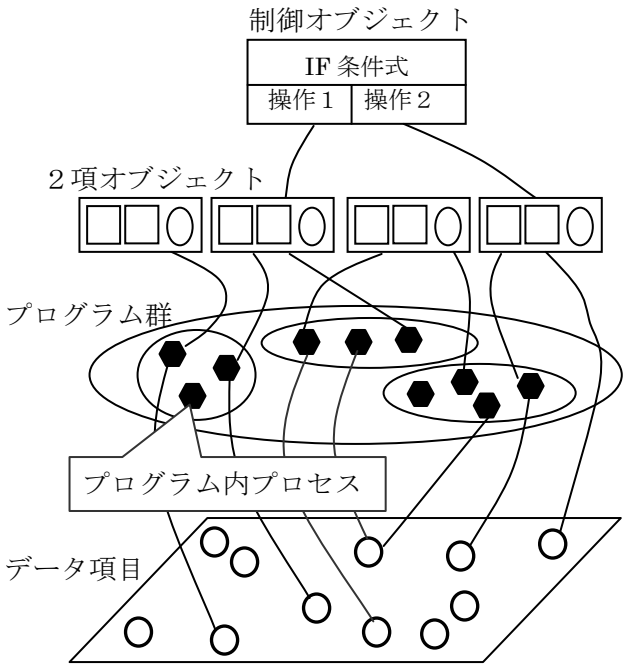


図 3.1　オブジェクト間の関連

3.2.2　２項オブジェクトによる表現

プログラムの基本操作(データ移送，演算など)を，オブジェクト(構成要素)とオブジェクトを関連付ける操作(メソッド)で表現する[79]. すなわちソースコードは基本的に２項関係のオブジェクトとして表現できる．例えば ***MOVE　A　TO　B.***と記述された基本操作は CPｎ[(A,B) MOVE]と表現される．CPｎはカプセル識別子，A と B は構成要素，MOVE はメソッドである．これを２項分析手法により抽出する．２項分析手法とは，データ操作を１対のデータに対する集約オブジェクとして捉えて, これを基本単位(２項オブジェクトと言う)にしてソースプログラムを２項関係まで分解する方法である．演算などに使用される変数は通常２つ以上あるので，この場合は，細分化して２項オブジェクトへの変換動作を繰り返す.

例えば ***COMPUTE　A　=　B + C*** では,

CP02[(A,CP03)COMPUTE]

CP03[(B,C)＋]

と表現され，更に以下のカプセルに変換できる.

CP01[CP02]

CP02[(A) COMPUTE]

CP02[(CP03) COMPUTE]

CP03[(B)＋]

CP03[(C)＋]

このように COBOL ソースコードを以下のように２項オブジェクトに変換してカプセル化する.

（１）演算式

逆ポーランド記法を用いて優先順位の高い項目から２項オブジェクトを抽出する.

COMPUTE　A　=　B + C　*　D を変換した結果を以下に示す.

CP01	C	＊
CP01	D	＊
CP02	B	＋
CP02	CP01	＋
CP03	A	COMPUTE
CP03	CP02	COMPUTE

演算式の優先順位を表 3.1 に示す．IF 文における優先順位も同じである．

表 3.1　演算式の優先順位

1	2	3	4	5
＊ /	＋ ―	＝ NOT ＝ ＜　＞	AND	OR

（2）条件式

条件式（IF 文，EVALUATE 文）は［命題部，真の時の操作，偽の時の操作］の構造であり，命題が真または偽の時の操作は複数の演算式などが記述されているのが多い．条件式のカプセルは，これらの複数カプセルを集約した構造で表現する．IF 文のネストは，命題部を AND 条件で整理する．その結果，上位の命題に影響を受けない独立した IF 文が整理される．

（3）PERFORM 文

COBOL の場合，IF 文などの操作で PERFORM 文を起動してセクションやパラグラフに制御を移すケースがある．PERFORM 文では，セクション名やパラグラフ名を構成要素として PERFORM をメソッドにしたカプセルで表現する．

このようにデータ操作に関するソースコードは２項オブジェクトによるカプセル化ができるが，プログラム制御に関するソースコードは２項オブジェクトに変換出来ない．

- プログラム制御文（*GOTO,GOBACK,CONTINUE*）
- *CALL* 文の *USING* 句

3.3　業務ルールの抽出

業務ルール抽出の流れを図 3.2 に示す．レガシーシステムでは，データ名称が不統一の場合がある．この場合は，事前準備として，異名同値のデータ項目を支援ツールで同値解析して標準データ名称に統一する．レコードの同じ位置を参照している名称や転送している名称，再定義している名称などが同値になる．

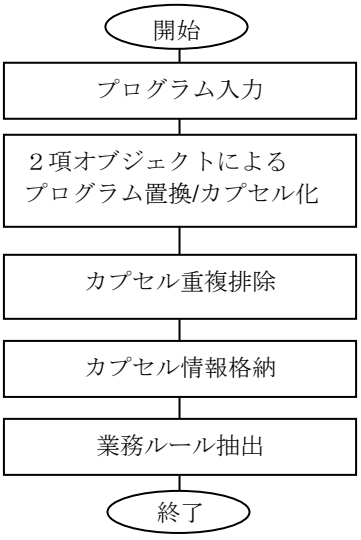


図 3.2　業務ルール抽出の流れ

既存プログラムを入力し，プログラムの各ステートメントを，個々のデータ要素に分解する．分解したデータ要素を一対のデータ要素である２項オブジェクで表現して，その固有のプロセスより構成するカプセルに置換する．次にカプセル化した情報の重複排除を行い，再利用が可能なデータ蓄積構造へ格納し，カプセル化した情報より業務ルールを抽出する．

（1） ２項オブジェクトの抽出とカプセル化

抽出対象の既存プログラムを入力し制約条件部とデータ操作部に分け，２項オブジェクトとその固有のプロセスより構成されるカプセルに置換する．図 3.3 に，抽出対象のプログラム例と，これらのプログラムの各要素に対する２項オブジェクトとその固有のプロセスよりカプセル化した情報を示す．プログラムの最初の処理単位（トランザクション）は，

*COMPUTE KINGAKU = TANKA * SURYOU*

で５個の要素からなるデータ操作である．これらの要素をカプセルに置換すると，例えば「TANKA」は，識別子 CP01 のカプセルに登録され，演算子「＊」はメソッドに登録される．「SURYOU」も同じくカプセル CP01 として登録される．更に，データ「KINGAKU」は，識別子 CP02 の構成要素に，もう一方の構成要素は既に作成したカプセル CP01 を構成要素とし，「COMPUTE」をメソッドとして，カプセル CP02 に登録される．このように最初のデータ操作部（１つのトランザクション）は，CP01 と CP02 の２つのカプセルに置換することができる．また，次の *IF USER = 1* で示される制約条件部では，「USER」と「１」を構成要素とし，そのメソッドを「＝」としたカプセル CP03 に登録される．

次に，制約条件部のカプセルとデータ操作部のカプセルを１つにまとめたカプセルの集約（制御オブジェクト）を作成し，業務ルールが抽出できるように整理する．図 3.3 のプログラムの

```
IF  KINGAKU > 100000
        THEN  COMPUTE  URIAGE = KINGAKU * 0.8
        ELSE  COMPUTE  URIAGE = KINGAKU * 0.9
        END-IF
```

の例では CP10 に登録される．CP10 は CP04，CP06，P08 のカプセルで構成されている．プログラムの他の IF 文は，それぞれ CP11,CP12,CP14 に変換される．

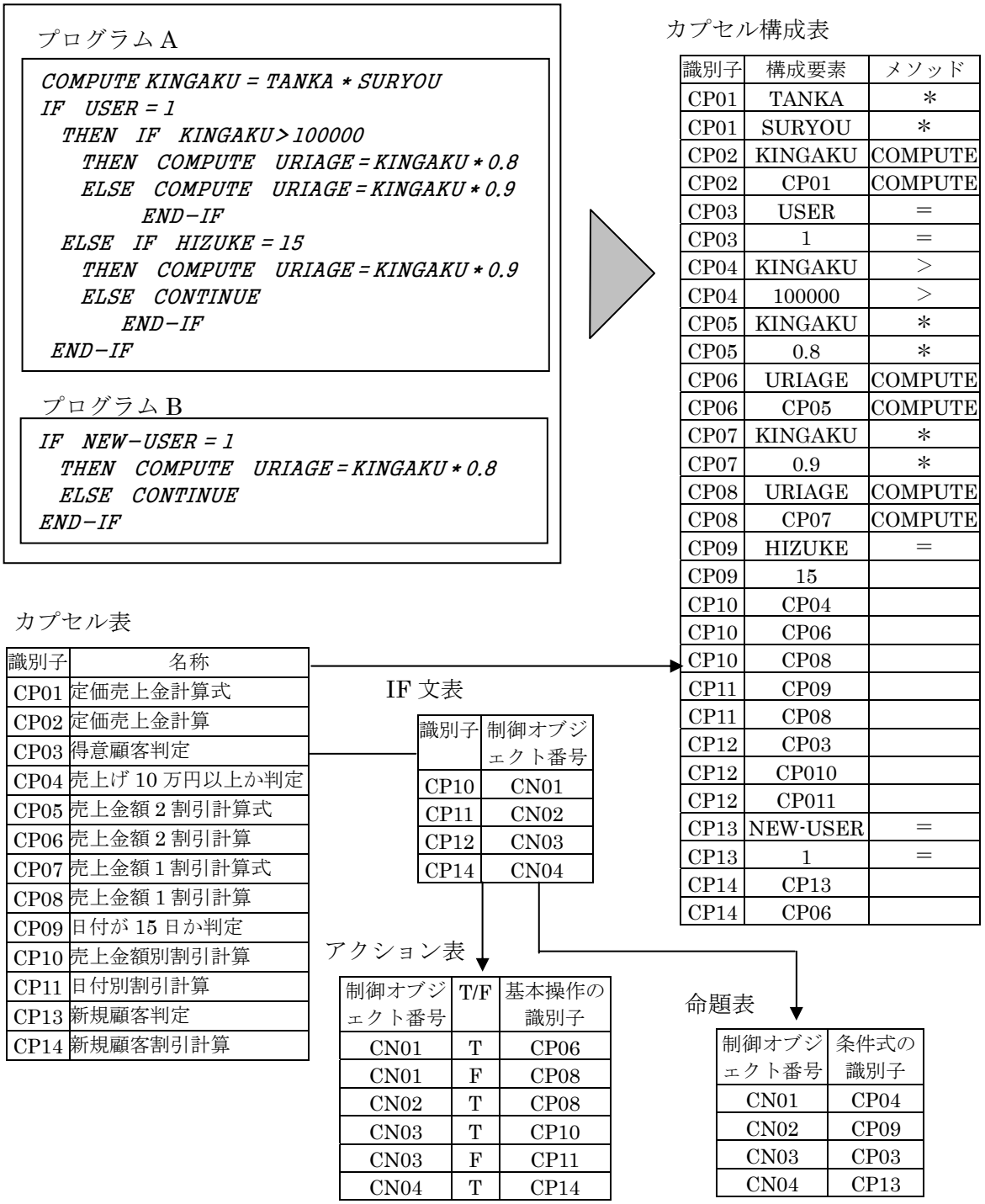


図 3.3　2 項オブジェクトの抽出とカプセル化の例

（2）カプセルの重複排除

プログラムを上記カプセルに置換した後，カプセルの重複排除を行う．プログラム A にあるデータ操作 *COMPUTE　URIAGE = KINGAKU * 0.9* は 2 箇所あり，全く同じ処理を行っており，その結果は，どちらも「KINGAKU」」とカプセル CP07 を構成要素，「COMPUTE」をメソッドとしたカプセル CP08 に置換される．またプログラム B では，プログラム A によってカプセルに置換されたデータ操作 *COMPUTE　URIAGE = KINGAKU * 0.8* と全く同じデータ操作を行っており，その結果は，同様にカプセル CP06 に置換される．本置換／整理方式では，重複したデータ操作や制約条件を２項オブジェクトとそのメソッドとして１つのカプセルにまとめる．即ち，同じ構成要素，メソッドの場合は１つのカプセル番号に整理する．

（3）カプセル情報の格納

抽出したカプセルをプログラム論理抽出のために，カプセル表，カプセル構成表，IF 文表，命題表とアクション表の５テーブルに格納して整理する(図 3.3)．カプセル表は，カプセルを一元管理するテーブルでありソースコードの全プロセスに対するカプセルを格納したテーブルである．それらを識別するためのカプセル識別子とカプセル名称を持つ．カプセル構成表は，カプセルの構成内容を表すもので，各カプセルを構成する構成要素及びそれに対するメソッドを表す．IF 文表は，カプセルと制御オブジェクトの対応を取るテーブルであり，例えば CP10 は，プログラム例の制御オブジェクトで，CN01 は，その制御オブジェクトを識別するデータである．命題表は，制御オブジェクトとその中で使用される制約条件カプセルの対応を格納するテーブルである．CP04 は，上記制御オブジェクト CN01 で使用される制約条件カプセルを示している．アクション表は，制御オブジェクトとデータ操作カプセルの対応を格納するテーブルである．CN01 の制御オブジェクトの制約条件が真である「T」の時には CP06，偽である「F」の時には CP08 のデータ操作カプセルが起動されることを示している．

（4）業務ルールの抽出

プログラムの置換／整理までを自動的に行い，次に業務ルールを抽出する．入力した既存プログラムは，２項オブジェクトとメソッドのカプセルに置換されているため，置換したカプセルを解釈していくことで業務ルールの抽出することが可能となる．しかしながらプログラム内のデータを自動的にカプセル化するので業務データの分別が必要になる．プログラム内のデータは，実世界を表現する業務データ，プログラム処理を制御する制御データ，画面表示属性などの実行環

境データ，日付やデータ件数の画面表示などのデータに分類される．この中から業務データに着目して整理する必要がある．業務データは準備作業で明確になっているが，制御データは業務ルールとして使われている場合があり一意に排除することは出来ない. 例えば, アベンドルーチン，エラー処理ルーチンでの操作，DB アクセスにおけるリターンコードの扱い，スイッチ変数の扱いなどである．スイッチ変数とは，データそのものは業務データではないが，どこかで業務データの意味を引き継いでいるデータである．

カプセル化した情報は Web のブラウザに表示して設計者が編集する．図 3.3 では業務ルールが理解できるように設計者がカプセル表の名称に日本語名称を与えている．抽出した業務ルールは Web ブラウザで指定データの業務ルール，指定データの導出の流れ，指定データと定数との関連(メソッド)などを表示したり，設計仕様書の形式で印刷出力する．

図 3.4 は DORE で業務ルールを表示した例である．

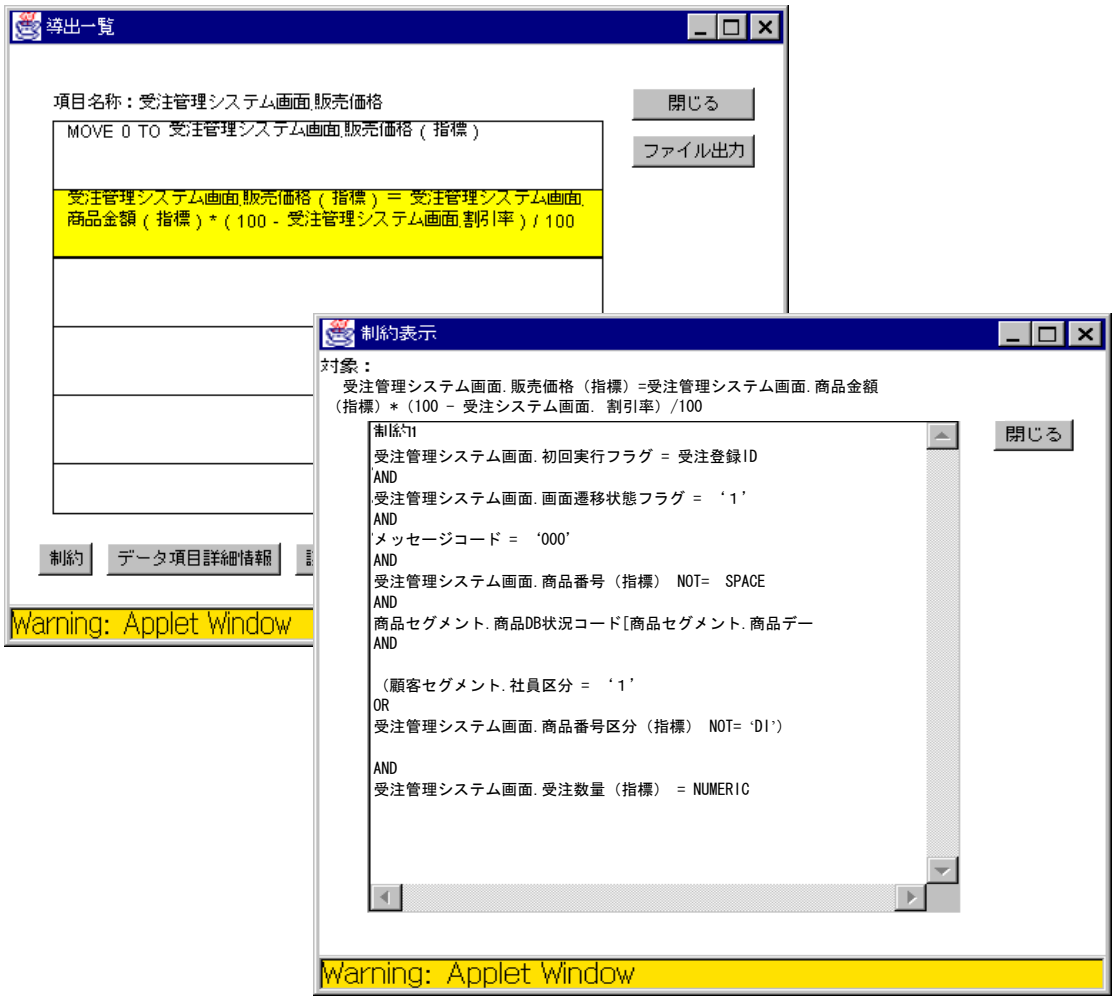


図 3.4　業務ルールの表示例

3.4 評価

抽出した業務ルールの正当性と仕様抽出率を実プロジェクトで評価して，本研究の有用性を検証した．業務ルールの正当性は，レガシーシステムの仕様書に記述している業務ルールと DORE により抽出した業務ルールを比較して評価した．業務ルールの仕様抽出率は，設計仕様書に記述している業務ルールに対する抽出率で評価した．

3.4.1 業務ルールの抽出評価

抽出した業務ルールの正当性を評価するために，製造業 A 社の物流システムの一部(プログラム 55 本，5 万ステップ)をサンプリングして評価作業をした．具体的には，レガシーシステムの仕様書に記述されている業務ルールと抽出した業務ルールを比較した．評価作業の結果，抽出した業務ルールは以下の 4 種類に分類された．表 3.2 に業務ルール評価結果を示す．

①仕様書一致ルール：仕様書の記述と一致した業務ルール

②デッドルール：制約内に論理矛盾を含む業務ルール

③暗黙のルール：プログラムの実装には必要であるが仕様書への記述は不要の業務ルール

④仕様書にないルール：２項オブジェクトにより抽出したが仕様書には記述されていない業務ルール

表 3.2　業務ルール評価結果

データ項目名称	仕様書に記述されたルール	DORE で抽出した業務ルール				
		仕様書一致ルール	デッドルール	暗黙ルール	仕様書にないルール	計
車番号	15	31	12	1	3	47
受渡条件コード	10	117	0	2	0	119
輸送業者コード	22	28	2	2	10	42
出庫時受場コード	11	11	5	0	7	23
合計	58	187	19	5	20	231

抽出した業務ルールを分析した結果，データのチェック仕様に関しては 100%抽出できた．これはデータチェックのソースコード(形式制約や値制約に関する処理)がプログラムの中で局所的に記述されていたのが理由である．データ項目「車番号」「受渡条件コード」では，仕様書一致ルールが仕様書記述のルールよりも多くなっていた．特に「受渡条件コード」では約 12 倍も増加している．これは

「車番号」「受渡条件コード」のデータ処理に直接関係のない分岐(IF 文による記述)や，制約の記述形式が統一されていないために業務ルールとしては同一制約にもかかわらず重複排除できずに抽出されたことによる．レガシーシステムでは，標準化されていないプログラムを対象にするので，このようなケースはよく発生する．制約内に論理矛盾を含むデッドルールは，実行時に通過しないパスを含んだルールである．これは機能拡張の時にプログラマにより作りこまれたのが原因である．この問題に対しては論理矛盾を検出してデッドルールを除去する機能を追加した．また暗黙のルールと仕様書にない業務ルールはレガシーシステムの業務有識者の判断により次期開発の基幹情報システムの業務仕様に反映するか決まるルールである．業務処理で当然のこととして当初から仕様書に記述されない場合と永年の機能拡張の内容が仕様書に反映されていない場合がある．特に仕様書にないルールでは重要な業務ルールが含まれていることが多い．評価の結果，仕様書記述の業務ルール抽出が確認でき，また次期システムへの仕様もれ防止を支援する仕様書にないルールの抽出(20 件仕様書記述業務ルールの 34%)により本研究の有効性を検証できた．

3.4.2 業務ルールの仕様抽出率評価

仕様抽出率を設計仕様書に記述している業務ルールに対する抽出率で評価した．DORE は，業務ルールの抽出作業の過程で，レガシーシステムを分析して各種設計情報を抽出するので，これらの情報も設計仕様書生成に反映させている．抽出する設計情報は，基本情報(ソース情報，JCL 情報，データセット/ファイル情報),クロスリファレンス情報(COPY/プログラムやデータ/ファイル/プログラムなどの関連情報)，プログラム構造情報(プログラムのセクション構造)，データ情報(データの属性，LCP，制約と同値情報)である．

評価は，製造業 A 社と公共関連 B 社の 2 プロジェクトで実施した．A 社では前項のサンプリングプログラム(規模：50KS)を対象に測定した．プログラムに対応する設計仕様書を用意して，その記述内容を抽出できた部分とできていない部分に選別して，それぞれの記述量を測定した．抽出率は，設計仕様書別抽出率とシステム特性別抽出率の 2 種類を測定した．システム特性では，オンラインプログラムをリアル処理とデータ伝送処理に分類し，バッチプログラムをロジック主体と帳票主体に分類した．この測定結果を物流システム(15 万ステップ)にあてはめて仕様書量と抽出率を算出したのが表 3.3 である．

表 3.3　A 社の仕様書抽出率

設計仕様書	設計仕様書の頁数				設計仕様書抽出率
	オンライン		バッチ		
	リアル処理	データ伝送処理	ロジック主体	帳票主体	
詳細説明書	195 頁	49 頁	160 頁	50 頁	55%
入力設計書	390 頁	14 頁	0 頁	36 頁	82%
出力設計書	385 頁	157 頁	12 頁	285 頁	78%
ﾚｺｰﾄﾞ設計書	511 頁	10 頁	69 頁	135 頁	78%
合計	1481 頁	230 頁	241 頁	506 頁	74%
特性別抽出率	76%	74%	63%	76%	74%

仕様書別の抽出率では，入力設計書，出力設計書，レコード設計書の 3 設計書は仕様をデータ項目別に記述していたため 80%前後の高い結果になった．複雑な関連制約などの業務ルールやデータベースへのアクセス方式を記述した詳細設計書では，抽出率が 55%であった．またシステム特性別抽出率では同じくロジック主体のバッチプログラムが他のプログラムと比較して 10%生成率が低かった．全体では生成率 74%(抽出仕様 1827 頁／全体仕様書量 2458 頁)である．

次に公共関連 B 社の共済システム(規模：962 本，900KS)の評価結果を述べる．B 社では，レガシーシステムのダウンサイジングに際して抽出情報から設計仕様書を生成した．この理由はレガシーシステムの設計仕様書がなく，しかもプログラムは標準化されていないためプログラムから直接設計文書を作成(再文書化)するのが非現実的であったからである．抽出情報に加えてデータ構造および「FILLER」に関する VALUE 句情報など２項オブジェクトに変換できない情報を抽出するプログラム情報補完ツールとフォーマットを変換して印刷する設計書作成ツールを開発した．生成対象の仕様書は，７種類の設計書と，サブルーチンに関するインターフェース仕様書である．

①プログラム処理概要図：プログラムのオンライン，バッチの種別，入出力の概要図，
入出力ファイル，ステップ数．

②プログラム機能説明図：プログラムの入力，出力と処理の関係(ＨＩＰＯ図形式)．

③ファイル/レコード仕様書：ファイル／レコード単位に，データ項目名と記号名，属性等．

④入力編集条件表：バッチ処理の入力となるデータ項目の元のデータと制約．

⑤画面帳票仕様書：画面または，オンライン帳票のデータ項目名毎の属性，制約．

⑥出力編集条件表：オンライン画面への出力となるデータ項目の元のデータと制約．

⑦補足説明書：出力するデータの制約(処理条件マトリクス形式)．

⑧インターフェース仕様書：サブルーチンのインターフェースエリアの，データ項目名，属性等．

③④はバッチプログラム用，⑤⑥はオンラインプログラム用の仕様書である．設計書作成ツールによる設計書は，プログラム単位に出力される．バッチ，オンライン，サブルーチンのプログラム毎に出力された設計書数から，仕様書生成率として集計した結果を表 3.4 に示す．

表 3.4　B 社の仕様書生成率

プログラム形態		バッチ	オンライン
対象プログラム本数		751 本	211 本
仕様抽出率	①プログラム処理概要図	91%	95%
	②プログラム機能説明図	91%	94%
	③ファイル/レコード仕様書	89%	—
	④入力編集条件表	91%	—
	⑤画面帳票仕様書	—	88%
	⑥出力編集条件表	—	88%
	⑦補足説明書	89%	88%
	⑧インターフェース仕様書	93%	

生成率は，生成対象としたプログラム本数と実際に設計書が生成できたプログラム数の比である．本研究では，仕様書生成率で 90%を目標として設計仕様書作成ツールの開発を進めた．当初は，COBOL の例外的な使用法や，入れ子 COPY 文に対する PREFIX 指定などがあり 60%程度の仕様抽出率であった．これらの問題について，影響の大きいものからの機能拡張やプログラムの修正などで対応することで，平均として 90%の仕様抽出率を得ることができた．生成した各種の設計仕様書は，レガシーシステムのプログラム設計情報を取得し，プログラム ID を基準に機械的に各種設計書への変換を実施しているため，COBOL プログラムのデータ記号名や命令文，JCL の DD 名を反映している．そのため，設計仕様書の記述項目には漏れが少なく，確度が高いといった特色がある．また，COBOL プログラムのセクション構造を正確に取得でき，作業者がソースを読み解く工数が削減できるため，プログラム機能説明図の生成も効果が高い．さらに，プログラム内の各種の制約を網羅した処理条件マトリックスを生成するので，作業者の仕様理解工数が削減できると共に，各処理条件の正確性が向上する．

3.5 結論

本章では，プログラムのデータに着目して，そこに内在する業務ルールを２項オブジェクトにより抽出するデータ中心型アプローチのリバースエンジニアリング DORE を述べた．DORE の機能と手順を述べ，実プロジェクトで分析した．その結果，DORE による業務ルールの抽出により，業務仕様の理解支援,設計仕様書の生成の有効性を確認した．また，抽出した業務ルールを制約条件として統合開発支援ツールのデータ辞書に登録して，基幹情報システムの開発に利用できることが分かった．今後は，業務仕様理解の他に，特定データの影響波及解析，不要/重複ロジックの抽出によるプログラム規模の圧縮などの活用が期待できる．

本研究の対象は，COBOL 言語で書かれたレガシーシステムであるが，世界のエンタプライズ系ソフトウェアの 70%は COBOL で書かれているという調査結果[81]や我が国における SI ベンダの受託開発ソフトウェアの 18%が COBOL であるという調査結果[11]により，本研究の実用性はあると考えられる．

本研究では，データ項目単位の業務ルールを抽出しており，イベントや時間に着目した業務ルールは抽出できない．今後の課題として，ジョブ制御を統括する運用管理システムやオンライン業務画面遷移情報などを業務処理の流れにマッピングするリバースエンジニアリングと，２項オブジェクトで抽出した業務ルールを併せた業務処理全体の理解支援技術の開発がある．また，基幹情報システム開発で効果的なのはソフトウェア部品の再利用である．抽出した業務ルールからソフトウェア部品を自動生成する技術の研究や，データ項目単位の業務ルールを組み合わせた業務コンポーネントの研究も今後の課題である．新基幹情報システムが採用するソフトウェアアーキテクチャー(MVC モデルや３層モデルなど)に対応したソフトウェア部品の生成技術やオブジェクト指向開発でのクラス生成技術などの適用技術研究も必要である．

第 4 章　ソフトウェア規模の試算見積

4.1　緒言

基幹情報システム開発の見積はプロジェクトの成否を左右する重要な要因のひとつである．見積は，業務要件からソフトウェアの規模を算出し，算出値にシステム特性(複雑性，品質，性能など)のパラメータ値を掛けて開発工数と開発コストを算出するのが一般的な見積方法である．

見積は，発注者と SI ベンダ間の正当な契約のため，またリスク管理のために段階的に実施することが推奨されている[40]．見積は，ソフトウェア開発プロセスの企画・計画フェーズで見積る試算見積，基本設計後に見積る概要見積，詳細設計後に見積る詳細見積がある．特に，企画・計画段階での試算見積は，最初の見積もりであり重要である．しかし，この時点では見積に必要な詳細業務要件は確定しておらずソフトウェア規模の算出は困難であった．

ソフトウェア規模を測るメトリクスとして，最近は「ソースコードの量」であるステップ数でなく，「機能の量」である FP が主流になりつつある．

試算見積で FP を計測する手法として, FP 試算法, NESMA 法[45]，協調フィルタリング法[46], UCP 法[47][48]，FP 要素見積法[49][50]，電中研法[51]などが提案されている．

FP 試算法は，過去のプロジェクト経験から試算式を設定する方式で，日立 SAS では，以下に示す試算法で FP を推定している．見積者は，複数の試算式を併用して総合的に規模を推定する．

・試算 FP1＝(画面数＋帳票数)×a
・試算 FP2＝(更新系ファイル数×35)＋(参照系ファイル数×15)
・試算 FP3＝類似システムステップ数÷b
・試算 FP4＝開発予算÷c

これらの試算法は，いくつかの数値メトリクスによりパラメトリック法で規模を予測する手法であるが誤差が大きい．試算 FP2 は，オランダのソフトウェア計測団体 NESMA (Netherlands Software Metrics Users Association)が提案している NESMA 法と呼ばれている FP 試算法である[45]．NESMA では，「試算 FP 見積(試算 FP2)と詳細 FP 見積は，良い相関(直線的な)を示しているが，いくつかのケースでかなりの差異(50%に到るずれ)が見られる」と報告している[45]．
試算 FP 見積の段階ではソフトウェアの仕様があいまいで，かつプロジェクトの特性が計算式に反映されていないので誤差が大きくなる．

協調フィルタリングによる見積法は，産学官 EASE プロジェクトの一環として奈良先端科学技

術大学院大学(以下，NAIST と言う)が開発した予測手法である[46]．EASE は，信頼性や生産性に課題の多いソフトウェア開発の分野において，観測型のプロジェクト管理手法であるエンピリカルソフトウェア工学(Empirical Software Engineering)[82]の確立を目指している[83]．協調フィルタリング法は，1990 年代に情報検索の分野において，多くの情報からユーザ嗜好情報の抽出，推薦技術として研究されてきた事例ベース類推法のひとつである[52]．

書籍のネット販売では，以下の手順で顧客の好みに合った推薦図書を推薦する．

（1）あらかじめユーザがアイテム(書籍，映画，楽曲など)を評価しておく．

（2）推薦対象ユーザと評価が似ているユーザ(類似ユーザ)を選び出す．

（3）類似ユーザの対象アイテムの評価を用いて推薦対象ユーザの対象アイテムの評価を見積る．

（4）推薦対象ユーザの対象アイテムの評価が高いと見積った場合，推薦対象ユーザに対して対象アイテムを推薦する．

ソフトウェア規模の試算見積も，同じ手順で予測する．協調フィルタリングのソフトウェア工数見積への適用性を評価した報告がある[84]．

UCP 法は，企画･計画フェーズの要求分析で作成されるユースケースモデルをもとに，ソフトウェア規模を見積る手法である[44][47][48]．「機能の量」として，FP をベースにした UCP(Use Case Point)を単位にする．UCP 法の有用性を評価した結果も報告されている[85][86]．UCP 法では，ユースケースモデルに記述される複雑さを考慮するために，アクタおよびユースケースに対して重み付けを行う．特に，ユースケースの重み付けではユースケース中で定義されているトランザクション数を数える．

FP 要素見積法は，ソフトウェアのデータ処理形態(トランザクションファンクション)を更新系，画面出力系，帳票出力系など 16 種類の要素機能に分類して，それぞれの FP 単価を掛けて規模を算出する手法である．電中研法も同じようにソフトウェアが取り扱う画面，帳票，ファイル，電文の数などから FP を算出する手法である．

本章では，試算見積手法の中から協調フィルタリング法に着目して予測モデルを提案し，同じく UCP 法では自動計測方式を提案する．協調フィルタリング法では，複数の変数を設定して類似事例を抽出する．ソフトウェア規模見積に用いる変数を 6 種類設定した予測モデルを提案する．変数は数値で表現できないカテゴリ変数と数値で表現できる数値変数がある．カテゴリ変数は 3 種類の値表現を設定して実測した．また，膨大な検索アルゴリズムからソフトウェア規模見積で最適なアルゴリズムを探し出す探索ツールを試作した．この予測モデルを用いて，過去のプロジ

ェクト実績に対して実評価を実施した．

UCP法は，企画･計画段階で作成したユースケースに基づいて工数見積りを行う．しかしながら，ユースケース中で定義されているトランザクション(原始的な一群のアクティビティ)数のとらえ方によって，同一プロダクトでも，結果に誤差が生じる可能性がある．より正確な見積りを実施するためには，組織的な見積り教育や訓練，実績データの蓄積が重要になる．この問題に対する一つのアプローチとして，UCPの自動化が有効であると考えられる．本章では，見積り経験の浅い者でも見積りが出来ることと，計測者による誤差を無くすことを目的として，UCP計測支援ツールU-ESTを試作した．具体的には，UCP自動計測のために，アクタとユースケースの重み付け方法を提案する．プロジェクトで作成されたユースケースモデルを用いて，ツールによる重み付けと経験者による手動での重み付けを比較して，ツールの有用性を確認した．

以降，4.2節で協調フィルタリング技術による試算見積を述べ，4.3節でユースケースポイント法による試算見積を述べる．

4.2 協調フィルタリング技術による試算見積

4.2.1 協調フィルタリングによる規模の予測

ソフトウェアの規模予測は，以下に示す手順で予測する[46].

(1)見積対象プロジェクトと類似したプロジェクトを算出する類似度計算.

(2)類似プロジェクトの実績値を用いて見積対象プロジェクトの規模を算出する予測値計算.

協調フィルタリングによる予測は，図 4.1 に示すm×n行列で表現されるデータセットを入力とする．p_iはi番目のプロジェクトを表し，m_jはj番目の変数を表す．$\nu_{i,j}$はプロジェクトp_iで計測された変数m_jの値を表す．予測対象プロジェクトp_aの変数値 $\nu_{a,j}$とデータセットの全プロジェクトの変数値 $\nu_{i,j}$の類似度を計算して類似プロジェクトを抽出する．次に予測値計算により予測目標m_nの予測値 $\nu_{a,n}$を算出する．

	m_1	m_2	…	m_j	…	m_b	…	m_n
p_1	$v_{1,1}$	$v_{1,2}$	…	$v_{1,j}$	…	$v_{1,b}$	…	$v_{1,n}$
p_2	$v_{2,1}$	$v_{2,2}$	…	$v_{2,j}$	…	$v_{2,b}$	…	$v_{2,n}$
…	…	…		…		…		…
p_i	$v_{i,1}$	$v_{i,2}$	…	$v_{i,j}$	…	$v_{i,b}$	…	$v_{i,n}$
…	…	…		…		…		…
p_m	$v_{m,1}$	$v_{m,2}$	…	$v_{m,j}$	…	$v_{m,b}$	…	$v_{m,n}$
予測対象プロジェクト								予測目標
p_a	$v_{a,1}$	$v_{a,2}$	…	$v_{a,j}$	…	$v_{a,b}$	…	$v_{a,n}$

図 4.1　予測に用いる m × n 行列

（1）類似度の計算

予測対象プロジェクトp_aと他のプロジェクトp_iとの類似度$sim(p_a, p_i)$を計算する．類似度計算方法は，コサイン計算法，相関係数計算法，ユークリッド距離計算法など 8 種類の計算方法がある．コサイン計算法は，ベクトルを用いた類似度計算アルゴリズムであり，ベクトルのなす角のコサインを用いて類似度を計算する．ユークリッド距離計算法は，プロジェクト間の距離を計算する．図 4.2 に変数が 2 個の場合のコサイン計算法とユークリッド距離計算法の例を示す．この例では，予測対象プロジェクトp_aと 1 番目のプロジェクトp_1の類似度を計算している．α はベクトルのなす角，dは距離であるが，いずれの場合も値が小さいほど類似度は高くなる．

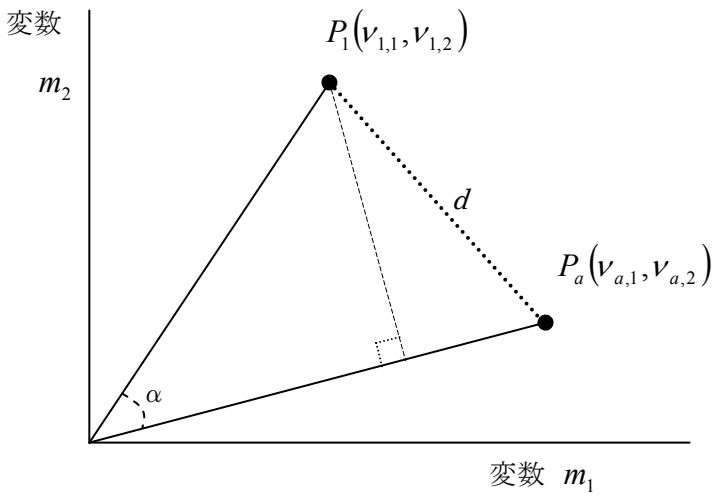


図 4.2　類似度計算の例

コサイン計算法では，次に示す計算式で類似度を計算している．M_aとM_iは，プロジェクトp_aとプロジェクトp_iの両方で計測した変数の集合である．この計算式では，値を持つ変数のみを用いて類似度を計算するので，欠損値の補完をする必要がない．これが協調フィルタリングの大きな特長である．

$$sim(p_a, p_i) = \frac{\sum_{j \in M_a \cap M_i} (v_{a,j} \times v_{i,j})}{\sqrt{\sum_{j \in M_a \cap M_i} (v_{a,j})^2} \times \sqrt{\sum_{j \in M_a \cap M_i} (v_{i,j})^2}}$$

（2）予測値の計算

類似プロジェクトの値と類似度を用いて，予測対象プロジェクトp_aの予測値を計算する．予測値 $v_{a,n}$は，類似度$sim(p_a, p_i)$を重みとして，類似プロジェクトの値 $v_{i,n}$に，プロジェクトの規模を補正する倍率修正$amplifier(p_a, p_i)$を乗じた計算式で算出する．

$$v_{a,n} = \frac{\sum_{i \in k-nearestprojects} (v_{i,n} \times amplifier(p_a, p_i) \times sim(p_a, p_i))}{\sum_{i \in k-nearestprojects} |sim(p_a, p_i)|}$$

k-nearestprojects ；プロジェクト p_a との類似度が高い k 個のプロジェクトの集合を表す．

4.2.2 ソフトウェア規模予測への適用

（1） 予測モデル

プロジェクト実績データ(対象 n=85)を対象に，6 種類の変数でソフトウェア規模(FP 値)を予測した．表 4.1 に変数とデータ欠陥率を示す．変数はカテゴリ変数と数値変数がある．カテゴリ変数とは，数値で表現できない変数である．「対応業種」は，当該システム機能に対応する業種で，17 種に分類した．「開発言語」は，「Java/．Net 系」「VB 系」「その他」の 3 種類に分類した．今回の試行では，異常データを排除した精度の高い実績データを用いて試行したので，欠陥率は 13%から 17%であった．

表 4.1　変数とデータ欠陥率

	変数	種別	欠陥率
1	対応業種	カテゴリ	0%
2	開発言語	カテゴリ(3 種類)	0%
3	画面数	数値	13%
4	帳票数	数値	13%
5	ファイル数	数値	0%
6	一般システム特性(14 種類)	数値	17%

一般システム特性(GSC：General SystemCharacteristics)とは，IFPUG で規定している 14 種の特性で，本来は FP 計測で使われる調整係数(VAF：Value Adjustment Factor)である．個々の特性のシステムへの影響度合い(DI：Degree of Influence)は 0～5 の整数で表すことになっており，数値に対応した影響項目が定義されている[43].

（2）評価指標

本研究では，予測値の評価指標として，次の５つのメトリクスを用いた．個々の予測での誤差分布を示す場合は，箱髭図(box plot)を用いた．図 4.3 に箱髭図を示す．実測値を Y と表し，予測値を $\hat{Y}$ と表す．メトリクスに含まれている Y の個数を t とする．

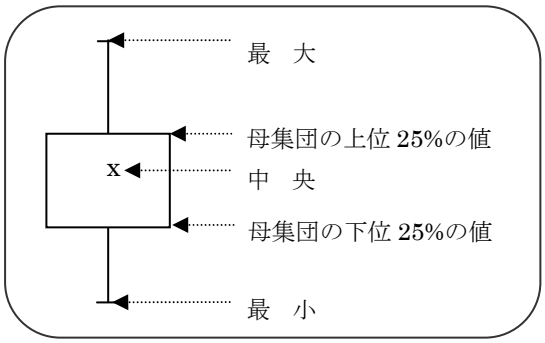


図 4.3 箱髭図

・相対誤差平均(MRE)：予測値が実測値からどれだけ離れていたかを示す．値が小さいほうが誤差が小さく，精度が高いことを示す．

$$MRE=\frac{1}{t}\sum\left|\frac{\hat{Y}-Y}{Y}\right|$$

・相対誤差中央値(MedRE)：相対誤差を昇順に整列したときの中央値．値が小さいと誤差が小さく，精度が高いことを示す．

・相対誤差分散(VRE)：相対誤差の全プロジェクトに関する分散．値が小さいほうが精度のばらつきが小さいことを示す．

$$VRE=\frac{1}{t}\sum\left(\left|\frac{\hat{Y}-Y}{Y}\right|-MRE\right)^2$$

・相関係数(Corr)：予測値と実測値の間の相関係数．値が大きいほど両者の相関が大きく，精度が高いことを示す．

$$Corr = \frac{\frac{1}{t}\sum\left(\hat{Y}-\overline{\hat{Y}}\right)\left(Y-\overline{Y}\right)}{\sigma_{\hat{Y}}\sigma_{Y}}$$

$\overline{\hat{Y}}$，$\overline{Y}$ はそれぞれ $\hat{Y}$，Y の平均値．
$\sigma_{\hat{Y}}$，σ_{Y} はそれぞれ $\hat{Y}$，Y の標準偏差．

・Pred25：相対誤差 0．25 以下で予測できたプロジェクト数の全プロジェクトに対する割合．値が大きいほうが精度が高いことを示す．

$$Pred25 = \frac{\sum isAccurate\left(\frac{\hat{Y}-Y}{Y}\right)}{t}$$

ただし，$isAccurate\left(\frac{\hat{Y}-Y}{Y}\right) = \begin{cases} 1 & \left|\frac{\hat{Y}-Y}{Y}\right| \le 0.25 \\ 0 & \left|\frac{\hat{Y}-Y}{Y}\right| \ge 0.25 \end{cases}$

（3）予測方法

図 4.4 に示すように，最初に予測対象のプロジェクト(P1)を選定する．次に，P1 の変数(「言語」など 6 種類)の値と，P1 を除く全プロジェクトの変数の値を評価して類似プロジェクトを抽出して，P1 の規模(FP 値)を予測計算する．この P1 の予測値と実績値を比較して評価する．この操作を P1~P85 の 85 回実行して，全プロジェクトの FP 値を予測して誤差を測定した．

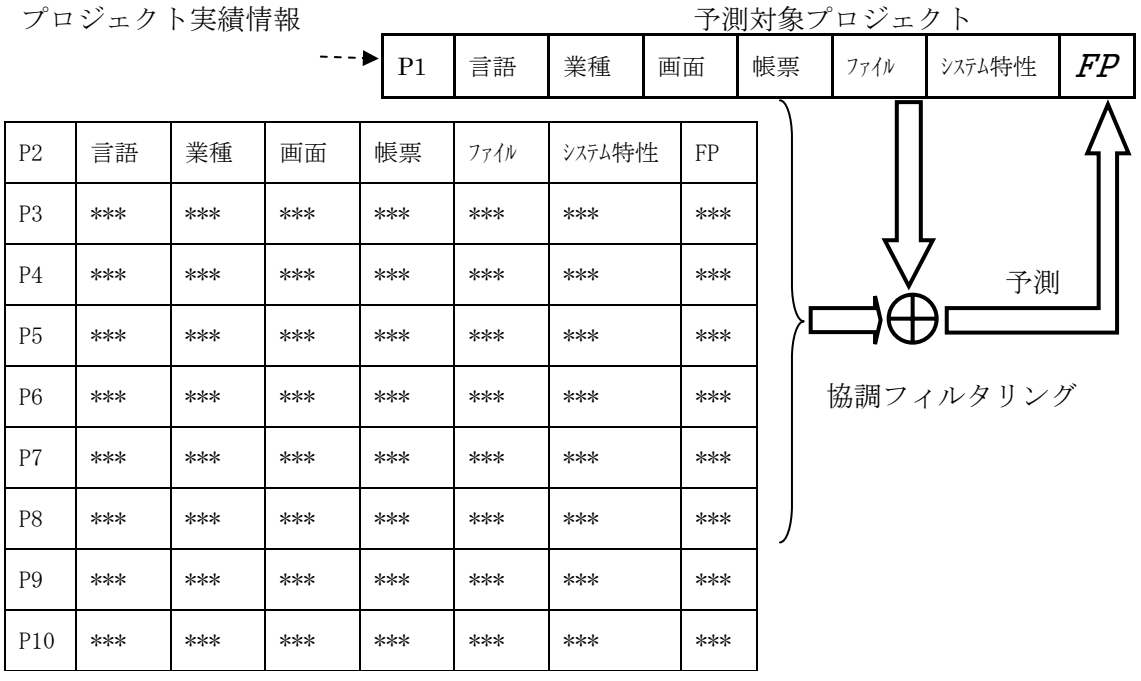


図 4.4　予測方法

（4）最適な検索アルゴリズムの探索

協調フィルタリングによる規模予測では，類似度計算式と予測値計算式の組合せである検索アルゴリズムを指定する必要がある．本研究では，各アルゴリズムでの予測値との誤差を比較して最適な検索アルゴリズムを探した．検索アルゴリズムは，類似度計算式，予測値計算式，変数の正規化，類似プロジェクト数を組合せて指定する．類似度計算式は，コサイン計算，相関係数，ユークリッド距離と各変数の平均値か中央値の組合せになり，予測値計算式は，類似プロジェクトの加重平均値と平均値，中央値，倍率修正値の組合せになる．計算上は 10 万通り以上に及ぶので自動的に有効と思われる組合せを生成し予測を実行・解析する「自動試行・結果解析ツール」を作成した．図 4.5 に本ツールの構成を示す．本ツールは，実行定義ファイルより検索アルゴリズムを自動生成し，NAIST 開発の協調フィルタリングツールを起動する自動実行制御機能と，協調フィルタリングツールより出力された予測データを蓄積し，優良な(誤差が小さい)検索アルゴリズムを探索する予測結果管理機能がある．

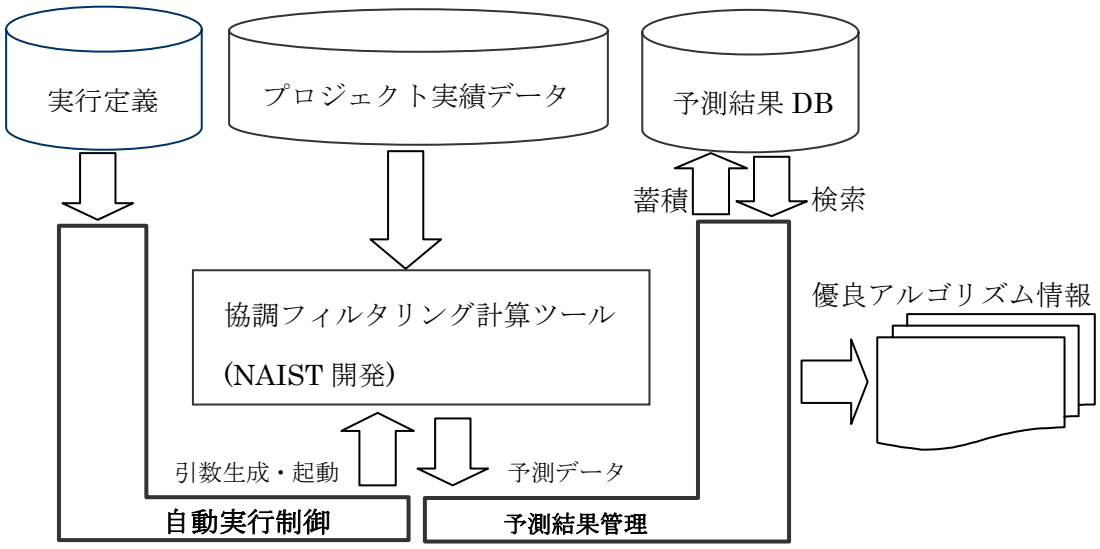


図 4.5　自動試行・結果解析ツール

予測モデルで説明した 6 種類の変数で最適な検索アルゴリズムを探索した．一般システム特性(GSC)は，数値変数(DI=[0，5])のデータと数値変数をカテゴリ変数に変換したデータを用いた．カテゴリ変数は 6 値表現(DI=[VALUE0，VALUE5])と 2 値表現(DI=(LOW，HIGH))に分けた．その結果，図 4.6 に示すように決定アルゴリズム r1 を探し出した．決定アルゴリズム r1 は，類似度計算式は相関係数計算法(各変数の値は平均値)とユークリッド距離計算法(各変数の値は中

央値)による按分計算式，予測値計算式は，類似プロジェクトの規模の加重平均と倍率修正値の中央値を用いたアルゴリズムである．NAIST では，標準アルゴリズム s1 を設定している．コサイン計算(各変数の値は平均値)，予測値計算式は，類似プロジェクトの規模の加重平均と倍率修正値の加重平均値を用いたアルゴリズムである．

4.2.3 評価

NAIST が過去の経験から設定した標準アルゴリズム s1 と，決定アルゴリズム r1 によるソフトウェア規模(FP)の予測結果を図 4.7 に示す．その結果，決定アルゴリズムでは，ケース 6 で相対誤差平均(MRE)が 0．28 という高い予測を出した．類似度計算式では，コサイン計算法単独よりも，計算法を併用した方法の精度が高かった．また GSC 値の設定方法で差異がでた．ケース 6(決定アルゴリズム，GSC:2 値)では，相対誤差平均(MRE)が 28%，相対誤差中央値が 22%の精度であった．同じくケース 4，ケース 5 においても MRE は 43%，33%であった．

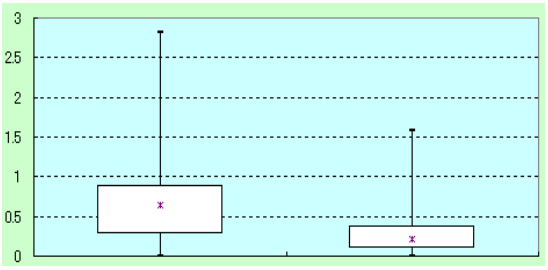
今回の予測では，検索アルゴリズムの探索と協調フィルタリングの有効性を評価するためにプロジェクト実績データは異常だと思われる値を含まないものを用いた．過去のプロジェクト実績データには，失敗プロジェクトなど予測精度を低下させる異常データも多く，欠損も多い．これら異常データを含む実績データを検索し，除去する方式の検討が今後の課題である．

協調フィルタリング(ケース 6)による予測結果と FP 試算法のひとつである NESMA 法による算出結果を比較した．図 4.6 に比較結果を示す．比較の結果，全ての評価指標において協調フィルタリングが優れている結果となったが，MRE の偏差分布を見ると，22 プロジェクト(全体の 26%)においては，NESMA 法の予測結果のほうが優れている結果が出た．

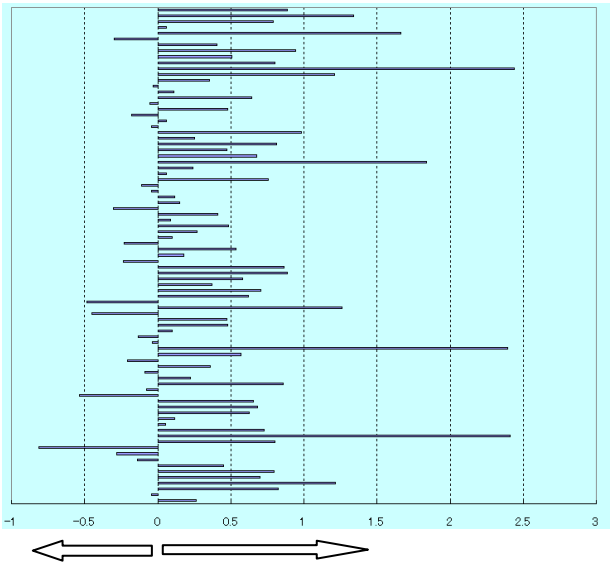
今回は，変数として 6 種類を選択し，その中で一般システム特性(GSC)の値設定を 3 種類に分けて試行したが，予測精度を上げる試みとして，「対応業種」について 17 種に分類されていたデータを，大分類(製造と非製造の２値)，大分類(金融と非金融の２値)，大分類(５種類)，中分類(11 種類)の４つに再分類し，予測を実行した．その結果，各分類方法での差異はほとんど見られなかった．これは対象プロジェクトが少なかったのが理由と思われる．

NESMA 法		協調フィルタリング法
0.72	相対誤差平均(MRE)	0.28
0.65	相対誤差中央値(MedRE)	0.22
0.59	相対誤差分散(VRE)	0.27
0.94	相関係数(Corr)	0.94
23.5%	Pred25	56.5%

相対誤差の分布



各プロジェクトの相対誤差の偏差



NESMA 法の誤差が小さい　協調フィルタリング法の誤差が小さい

図 4.6　NESMA 法との比較

一般システム特性

特性 1	データ通信	アプリケーションがプロセッサと直接通信する度合い
特性 2	分散処理	アプリケーションのコンポーネント間データ転送度合い
特性 3	性能	応答時間やスループットの要求が与える度合い
特性 4	高負荷構成	資源の制約がアプリケーションの開発与える影響度
特性 5	トランザクション量	単位時間当たりのトランザクション量が与える影響度
特性 6	オンラインデータ入力	データが対話的に入力される度合い
特性 7	エンドユーザ利便性	アプリケーションの使いやすさの度合い
特性 8	オンライン更新	内部論理ファイルがオンラインで更新される度合い
特性 9	複雑な処理	処理ロジックの複雑さの程度
特性 10	再利用可能性	コードを再利用可能なように設計・開発すべき度合い
特性 11	インストール容易性	導入や移行が開発にて意識される度合い
特性 12	運用性	起動・バックアップ・回復処理などの運用性
特性 13	複数サイト	アプリケーションが複数のサイトより利用されることを示す
特性 14	変更容易性	ソフトウェアロジック/データフォーマットの変更容易性

GSC：数値表現データ

Pn	特性 1	特性 2	特性 14
P1	3	2	2
P2	2	4	1
P3	3	1	1

GSC： 6 値表現データ

Pn	特性 1	特性 2	特性 14
P1	HIGH	LOW	LOW
P2	LOW	HIGH	LOW
P3	HIGH	LOW	LOW

GSC： 2 値表現データ

Pn	特性 1	特性 2	特性 14
P1	VALUE3	VALUE2	VALUE2
P2	VALUE2	VALUE4	VALUE1
P3	VALUE3	VALUE1	VALUE1

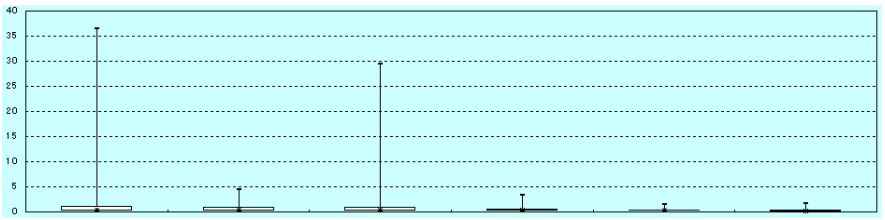
ケース	1	2	3	4	5	6
データ	GSC：数値	GSC： 6 値	GSC： 2 値	GSC：数値	GSC: 6 値	GSC: 2 値
アルゴリズム	標準アルゴリズム s1			決定アルゴリズム r1		
予測結果	⇩	⇩	⇩	⇩	⇩	⇩
相対誤差（MRE）	1.21	0.69	1.03	0.43	0.33	0.28
相対誤差中央値	0.45	0.40	0.42	0.30	0.28	0.22
相対誤差分散(VRE)	3.97	0.77	3.20	0.52	0.28	0.27
相関係数（Corr）	0.93	0.87	0.93	0.94	0.94	0.94
Pred25	24.7%	31.8%	30.6%	37.7%	43.5%	56.5%
相対誤差分布						

図 4.7　一般システム特性データとアルゴリズム選択による予測結果

4.3　ユースケースポイント法による試算見積

4.3.1　ユースケースポイント法による規模の予測

（1）ユースケースポイント法

ユースケースモデルは，ユースケース図とユースケース記述の2つの要素で構成されている．

・ユースケース図:ユースケースの概念をUML(Unified Modeling Language)で視覚化して表現したもの．アクタを人間の形のアイコンで示し，ユースケースを楕円形のアイコンで表現する．アクタとユースケースとの関連は実線で表す．

・ユースケース記述:ユースケース図に記述されたユースケースそれぞれについて，そのユースケースの機能を実現するために必要な詳細情報を記述したもの．

「注文を出す」というユースケースに関し，ユースケース図とユースケース記述をまとめた例を図4.8に示す．

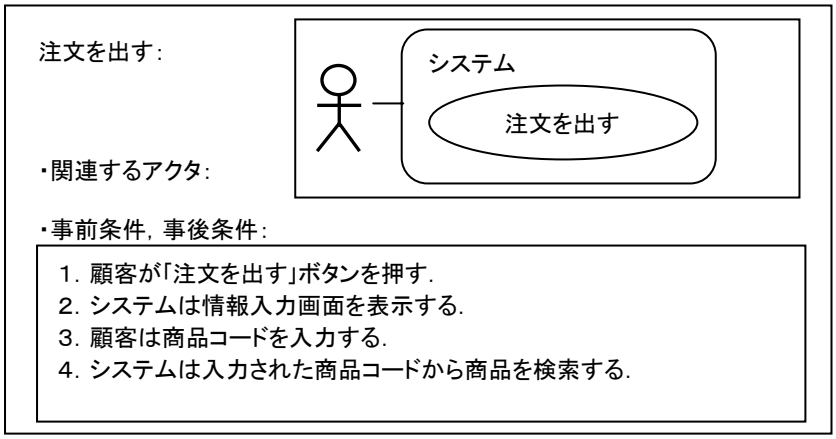


図 4.8　「注文を出す」ユースケース

UCP 計測においては，ユースケース記述の中で，そのユースケースにおけるアクタとシステムとのやり取りを記述した「イベントフロー」に着目する．また，イベントフロー内の 1 つのパスを「イベント」と呼ぶ．

UCP法は，オブジェクト指向開発の早期段階における工数及びコスト見積手法が要求される中で，Karnerによって提案された工数見積手法である[44]．この手法は，FP法をベースとし， ユースケースモデルに基づいて見積りを行う．具体的には，ユースケースモデルに記述されるアクタ，ユースケースを重み付けしてその数を数え上げ，計測対象プロジェクトの技術的な要因や開発環境の要因をもとに調整を行い，工数を見積る．

（2）UCP法の計測手順

UCP法による，ソフトウェアの規模計測は以下に示す手順で計測する．

・Step1：アクタ， ユースケースの重み付け

・Step2：未調整ユースケースポイントの算出

・Step3：技術要因，環境要因による調整

・Step4：ユースケースポイントの算出

・Step5：工数の見積り

直観的には，アクタとユースケースを重み付けして足し合わせたものが未調整ユースケースポイントであり，それを開発における技術要因や環境要因で調整(補正)したものがユースケースポイントとなる．本研究においては，特にStep1，2の部分に着目しているため，Step1，2について説明を行う．Step3以降については，文献[44][53]を参照．

Step1では，まず，ユースケースモデルに記述されているアクタについて，表4.2に基づいてタイプの分類を行う．各タイプには重みが設定されており，各タイプのアクタの個数に重み係数を掛け，合計したものが，アクタの重み(AW：Actor Weight)となる．ここで，単純なアクタというのは，定義済みのAPI(Application Programming Interface)を備えた別システムを指す．

平均的なアクタは，TCP/IPなどのプロトコルを介して計測対象システムと相互作用する別システム，もしくはテキストベースのインタフェース(ASCII端末といったもの)を介して相互作用する人間を指す．複雑なアクタは，GUIを介してシステムと相互作用する人間を指す．

表4.2　アクタのタイプと重み係数

タイプ	説明	重み係数
単純	定義済みAPIを備えた別システム	1
平均的	プロトコル駆動のインタフェース(別システム)， テキストベースのインタフェース(人間)	2
複雑	GUIを介する人間	3

次に，ユースケースについても表4.3に基づいてタイプの分類を行う．ユースケースの場合は，イベントフロー内のトランザクションの数に基づいてその複雑さを決定する．トランザクションは，原始的な一群のアクティビティと定義され，これはすべて完全に実行されるかあるいはまったく実行されないかのどちらか一方となる．このトランザクションの定義は，ファンクションポ

イント法におけるトランザクションファンクションの定義(ソフトウェアに対するデータの出入りを伴う処理，例えば，[画面からのデータ登録]，[帳票出力]，[問い合わせ応答])を基にしている. また, includeするユースケースや拡張ユースケースについては, 計測の対象とされていない. ユースケースについても，各タイプの個数に重み係数を掛け，すべて合計する．これをユースケースの重み(UW：Use Case Weight)と呼ぶ.

表 4.3　ユースケースのタイプと重み係数

タイプ	説明	重み係数
単純	トランザクション数が 3 個以下	５
平均的	トランザクション数が 4 個から 7 個	１０
複雑	トランザクション数が 8 個以上	１５

Step2(未調整ユースケースポイントの算出)では，算出されたアクタの重みと，ユースケースの重みを合計して，未調整ユースケースポイント(UUCP：Unadjusted Use Case Point)を得る.

$$UUCP = AW + UW$$

4.3.2　ユースケースポイントの自動計測

ユースケースポイントの自動計測を行うツールを試作するにあたって, 次の2点が考慮すべき点となった.

(1)アクタの分類に必要な情報の抽出

(2)ユースケースのイベントフローのトランザクション抽出

そこで，なるべく特定の環境に特化しない制限をアクタとイベントフローの記述におき，その上で分類を行うこととした．以降，それぞれの計測方針について述べる.

(1)アクタの分類方法

Karnerの定義では，アクタの分類は，そのアクタがシステムに対してどのようなインタフェースで相互作用するかということを基準にしている．しかし，モデルに記述されるアクタの持つ情報は名前のみであり, インタフェースに関する情報を明示的に持っているわけではない. そこで, 本研究では，アクタの名前とアクタが関連するユースケースのイベントフロー記述に注目し，以

下のStepA1とStepA2で分類を行う．

① StepA1：アクタ名による分類

表4.2より，アクタが人間であるか，もしくは外部のシステムであるかで，タイプの候補を2つに絞ることができる．すなわち，外部システムであれば単純，もしくは平均的であり，人間であれば平均的，もしくは複雑となる．そこで，第1段階として，まず人間か外部システムであるかを判断する．

外部システムであると判断できるキーワードを設定し，そのキーワードを名前の語尾に持つアクタを外部システム，そうでなければ人間とする．キーワードの例としては，「システム」「サーバ」などが挙げられる．また，見積を行う組織， 団体等で，ある特定の命名規則を設定する可能性も考慮し，このキーワードはユーザによる変更・追加も可能とする．

外部システムに関するキーワードとしたのは，人間である場合の名前には無数のパターンが考えられ，外部システムに関するそれよりも数が膨大であると判断したためである．

（ルール A-1）： **アクタ名の語尾がキーワードと一致するアクタは外部システム，それ以外を人間とする．**

② StepA2：キーワードによる分類

アクタ名による第1段階の分類後，絞られた2つの候補から1つに決定する第2段階の分類を行う．ここからは，どのようなインタフェースを持つかに依存するが，先に述べた様にアクタ自身はその情報を陽には持たない．そこで，対象アクタが相互作用するユースケースのイベントフローに着目し，インタフェースに関する情報をそこから得ることとした．図4.9にアクタのタイプ決定プロセスを示す．

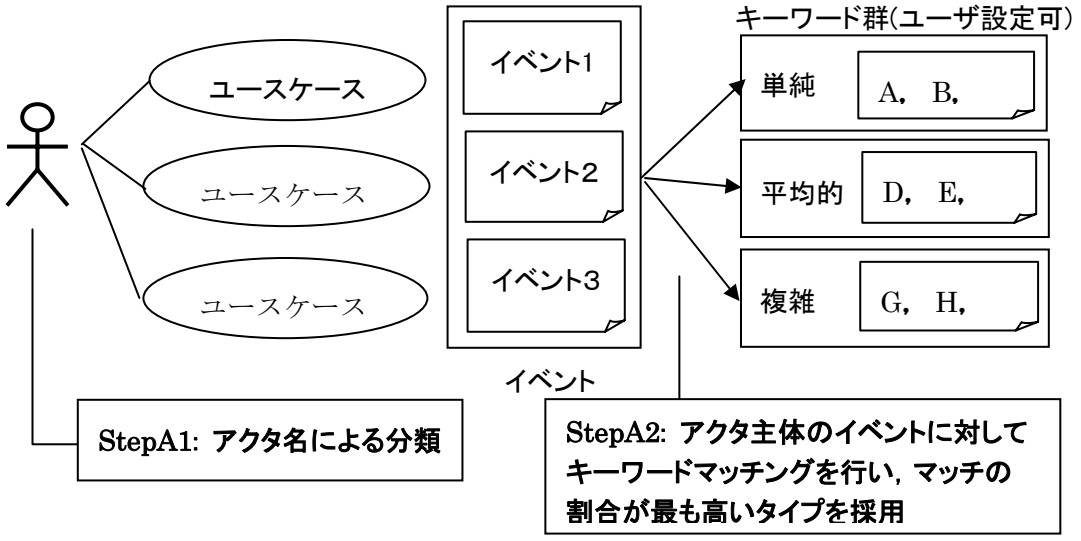


図4.9　アクタのタイプ決定プロセス

具体的には，先ず，インタフェース情報に関するキーワード群を，それぞれのタイプに設定する．キーワードとしては，アクタがシステムに対して行う処理に関するものがある(「入力」，「選択」等)．次に，アクタが関連するユースケースのイベントから，対象アクタが主体となるイベントを取り出し，そのイベント中の単語の中からキーワードと合致するものを探し，当該イベントを合致したキーワードが属するタイプと同じタイプと判定する．しかし，イベント中には，特定のタイプのキーワードだけでなく，異なるタイプに属するキーワードが含まれる可能性がある．この時には，イベント中で最も多く合致したキーワードを含むタイプをイベントのタイプとする．また，このキーワードについてもユーザが変更・追加可能とする．

(ルール A-2)：対象となるアクタが関連しているユースケースのイベントを抽出し，タイプごとのキーワード群に対する合致する割合が最も多いものをそのアクタのタイプとする．

StepA1とStepA2の分類方式を決めた後，次に汎化に対する処理を検討した．

アクタ間には汎化関係が存在するが，Karnerはこの汎化関係については特に言及してはいない．この関係が存在する場合，親アクタの機能を継承した子アクタについては，自分が関連するユースケースを調べるだけではなく，親アクタの関連するユースケースも調べてマッチングをとるべきである．また，もし親アクタに関連が全く存在していない場合は，機能を継承した子アクタが既に存在するため，カウント対象に入れるべきではないとした．このことから，以下のルールを追加する．

(ルール A-3)：汎化関係のあるアクタについては，子アクタは親アクタの関連するユースケースのイベントフローに対してもキーワードの照合を行う．

(ルール A-4)：親アクタに関連するユースケースが存在しない場合，その親アクタはカウントの対象としない．

（2）ユースケースの分類方法

ユースケースの複雑さは，その内部の処理(トランザクション)の数によって決まる．そこで，ユースケース記述内のイベントフローの情報からトランザクションを抽出して分類を行う．トランザクション数をカウントする最も単純な方法としては，イベントフロー内のイベントの数をカウントすることである．しかし，ユースケース記述は定まったフォーマットが存在せず，作成者

が自由に記述できるので，例えばトランザクションが2つ認識できる様な処理が，1つのイベントとして記述される可能性がある．

前述のように，トランザクションは原始的な一群のアクティビティと定義されている．換言するとトランザクションは，ユーザあるいはシステムにとって一つの意味のある処理である．Jacobsonは, ユーザとシステム間の相互作用について代表的な4つの処理として以下を挙げている[87]．

- 主アクタがシステムに要求とデータを送る．
- システムが要求とデータを確認する．
- システムがその内部状態を変更する．
- システムがアクタに結果を返す．

このような一連の処理内容を1つのトランザクションとして認識することは有効である．その場合，イベントを分析して「主アクタとシステム間でデータをやり取りしている可能性があるもの」を抽出することがトランザクション抽出の基本となり，「要求とデータを送る」処理を記述しているイベントがトランザクションの開始となる．

トランザクションの開始部分を識別するための方法として，イベント記述の自然語文章から主語や述語を認識し，アクタやシステムが主語であり，かつ，上述した処理が述語として現れているものを抽出する方法をとる．

そこで，以下のStepU1～U3 でトランザクションを抽出する方法を用いた．

① StepU1：ユースケースシナリオ内のイベントに対して形態素・係受け解析を行い，主語と動詞のセットを，トランザクションを構成するイベントの候補とする．

② StepU2：抽出された候補から，主アクタとシステム間でデータをやり取りしている可能性があるものを取り出す．

③ StepU3：アクタの動作から，システムの応答までの一連の動作をまとめて，1つのトランザクションとする．

① StepU1：形態素・係受け解析によるイベント候補の抽出

イベント記述に対する形態素解析および係受け解析のために，日本語係受け解析器「南瓜」[88]を利用する．「南瓜」は, SVM(Support Vector Machines)に基づく日本語係受け解析器[89]であり，統計的な日本語係受け解析器として最も精度が高い(89．29%)とされているツールである．

南瓜の出力から，イベント中の名詞や動詞など，文の成分の情報が得られる．ただし，主語と

目的語については名詞というカテゴリで分類されているため，さらに助詞などをチェックして主語や目的語であるか判定する必要がある．なお，主語がアクタとシステム以外のイベントについては無視する．

（ルール U-1）：イベントに対する形態素解析と係受け解析を行い，主語がアクタとシステムのものをトランザクションに含まれるイベント候補とする．

② StepU2：イベントの抽出

StepU1で抽出されたイベントに対して動詞の分類を行う．主語と述語となる動詞の対応がとれても，必ずトランザクションになるとは限らない．トランザクションは主アクタとシステム間のデータのやり取りから認識するため，主語がどのような処理を行っているか，動詞を詳しく調べる必要がある．動詞の特徴を調べるために，以下の4つのカテゴリを設けた．

- アクタ入力A
- アクタ入力B
- システム出力A
- システム出力B

「アクタ入力」とは，主アクタからシステムに対するデータの入力要求の可能性がある動詞であり，「入力(する)」や「検索(する)」などの動詞が挙げられる．「システム出力」とは，システムから主アクタに対する応答の可能性がある動詞であり，「出力(する)」や「表示(する)」などの動詞が挙げられる．主語とは独立して，抽出された動詞がそれぞれどのカテゴリに当てはまるかチェックする．「アクタ入力」と「システム出力」に大別されるが，それぞれの項目について，処理の違いによってAとBに更に詳細に分類している．

主アクタの入力操作を表す動詞は大別して2つのパターンがある．操作の段階を表す動詞か，一語で応答までを確定できる動詞(引き続きその入力操作に対するシステムからの出力操作が来ない場合がある)かである．前者をA，後者をBとして定義する．例えば「検索する」という操作は，「(検索キーを)入力する」(操作の段階を表す)と「(システムに)送信する」(一語で応答までを確定できる)という2行のイベントで表現される可能性もある．「削除する」という操作は，「(削除すべき項目を)選択する」と「(削除キー等を)押す」というイベントで表現される可能性もある．このように，システムの応答部分が記述されずに，アクタの動作が2行続けて記述された場合にこれらの

区別をするためにAとBの分類をする.

システム操作を表す動詞にも同様に2つのパターンがある. 主アクタへの応答の動詞と更に別のシステムへの入力を想定させる動詞である. 前者をA, 後者をBと定義する.「表示(する)」などは, 主アクタからのデータによって確認を行い, 処理の結果を返す応答を表す動詞であると考えられる. 他方,「認証(する)」などは, 対象システムを経由して, さらにデータベースやパスワード認証システム等の外部システム(アクタ)へと要求を渡すことが想定される動詞である. これは応答とは別のトランザクションとしてカウントしなければならない.

（ルールU-2）： イベント候補文の動詞を調べ, アクタの入力処理とシステムの出力処理を分類する.

③ StepU3：トランザクションの判定

StepU2で得られた主語と動詞のセットの情報からトランザクションの判定を行う. 基本原則は, 主アクタからシステムに対して要求とデータの送信を行うイベントをカウントし, それ以外の処理を, それに付随したシステムからの応答として纏める.

具体的には, 主アクタからシステム方向のデータの送信が行われたイベントを検出し, システムからの応答までを一つのトランザクションとして認識する. ただし, 人間が書く文章の全てがこれに当てはまるとは限らない. 主アクタからのデータ送信のイベントのみが書かれる可能性もある. この時, 主アクタの動作が, 本当に一つのトランザクションを表しているかを, 確認しなければならない. 前述したとおり, 主アクタからシステムにデータ送信を行う方向の動詞には二種類ある.「入力」という動詞がシステムに対するデータ要求までを意味している場合も,「入力」と「送信」で一つのデータ要求を表している場合もある. これらを区別するために, 対象としている前の行の動詞を調べ, 一つのデータ要求として独立させるか, 複合させるかを決定する. 対象としている行の動詞がトランザクションの開始の可能性がある場合, その前の行の動詞がアクタ入力Aでなければ, 対象としている行はトランザクションの始まりと判断し, アクタ入力Aであればまだユーザからの入力処理の途中であると判断する.

（ルール U-3）: アクタからシステム方向のデータ送信処理のイベントを起点に, システムからの応答までを 1 つのトランザクションとして認識する.

StepU1，StepU2，StepU3 でユースケースの分類方式を決めた後，包含や拡張に関する処理を検討した．Karner の UCP 法[44]では，包含や拡張ユースケースについては考慮していないが，このようなユースケースに，システムの本質的な処理が含まれる場合もあることが指摘されており[85][90]，機能規模計測においては無視できない．

包含や拡張の関係が存在するとき， それをひとつのユースケースとして処理を記述し，その処理を含むユースケースは，自身のイベントフロー内で包含（拡張）ユースケースを参照するイベントとして記述を行う．このことから，我々は以下に示すルールを設定した．

（ルール U-4）：包含または拡張するユースケースは，1 つのユースケースとして複雑さを決定し， 計測対象とする．

（ルール U-5）：それらを参照するユースケースは，イベントにおけるその参照処理をトランザクションに含めない．

（３）アクタ，ユースケース記述の制限と分類不可能時の対策

これまでに述べてきたように，提案する計測手法ではユースケースモデルに対して幾つかの制限を設けた．

・ユースケース記述のイベントの文章構造は単純にする．イベント記述では，主語，修飾語，目的語，述語(少なくとも主語と述語だけでも)をきちんと記述する．これは形態素解析，係り受け解析を行う上で必要な制限である．

・利用組織のコンテキストにあわせて，アクタ名やトランザクション分類のための動詞キーワードを設定する．アクタ分類，ユースケース分類におけるアクタ入力AとB，システム出力AとBに対応するキーワード，動詞の設定を決める．

いずれの制限についても，ある程度のユースケースモデル開発経験のある組織では負担の大きい制限ではない．実際に，4.3.2で述べる評価実験では，特に問題とはならなかった．

しかし，アクタとユースケースの分類に関して，分類不可能な場合が現実には考えられる．その場合には，基本的には過去に計測されたプロジェクトの様々な情報を用いた支援が対策として考えられる．

①アクタ分類不可能時の処理

アクタの分類方法で述べた方法によりアクタのタイプを決定するが，関連するユースケースのイベントフローに，用意したキーワードに全くマッチしない場合もある．この場合，ユーザを支援する手法として，過去の情報を利用することを考える．

具体的には，過去に計測されたプロジェクトの様々な情報をデータベースとして蓄積する．その中のアクタの情報から，分類が不可能なアクタに類似したアクタを検索し，見つかった情報をユーザに提示する．検索はアクタ名の部分一致による検索を行い，アクタ名，そしてそのアクタに対して過去に決定されたタイプ等の情報を表示する．ユーザはそれを受けて， タイプを手動で設定，修正する．

もし，過去の情報からも類似アクタが見つからない場合は，デフォルトのタイプとして，外部システムであれば「単純」，人間であれば「複雑」とする（平均的とは判断しない）．その理由は，前者は，最近のソフトウェア開発の現状を考えると，開発者が全ての処理を記述するより，開発環境のサポートによって，効果的に定義済みAPIを使用して開発することが多いからである．また，後者は，ユーザとシステム間のインタフェースもGUIを介する場合が多いためである．

（ルール A-5）：ルール A-1，A-2 で判定できない場合は，過去のプロジェクトで類似したアクタのタイプを利用する．

（ルール A-6）：ルール A-5 が適用できない場合は，外部システムを「単純」，人間を「複雑」と判定する．

②ユースケース分類不可能時の処理

Karnerの定義では，ユースケースの複雑さを決定するには，イベントフローが記述されていることが前提となる．しかしながら，実際の開発では，ユースケース図が記述されていても，そのイベントフローまでは記述されないこともある．この場合，アクタの場合と同様に，過去の蓄積された情報を利用することを考える．蓄積されたユースケースの情報から，対象となるユースケースと類似したユースケース（例えば，同じ名前を持つ）を検索し，見つかった情報をユーザに提示する．ユーザはそれを受けて，複雑さの設定，修正ができる．

もし，過去の情報からも類似したユースケースが見つからなかった場合は，複雑さを「平均的」とする．

（ルール U-6）：イベントフローを持たないユースケースについては過去の情報を利用する．

（ルール U-7）：過去情報が見つからない場合は，「平均的」複雑さとする．

4.3.3 UCP 計測支援ツール U-EST

UCP 計測支援ツール U-EST(Usecase-based Estimation Supporting Tool)を試作した．U-EST は XMI(XML Metadata Interchange)形式で記述されたユースケースモデルを入力とし，UCP の規模見積を行う(開発言語：Java，規模：5KS)．なお，UML ツールとして Describe[91]を利用した．本ツールのシステム構成図を図 4.10 に示す．システムは 4 つのサブシステムから構成されており，処理はユーザが GUI を通して操作する．

・XMI解析部

XMI形式で記述されたモデルファイルを解析し，記述されたアクタ，ユースケース，イベント等，計測に必要な情報を抽出する．

・シナリオ解析複雑度測定部

XMI解析部で抽出した情報を元に，各アクタ，ユースケースのタイプ分類を行う．その結果から，未調整ユースケースポイントを算出する．各アクタ，ユースケースに対する分類結果，及び未調整ユースケースポイントは，GUIを通してユーザに提示する．

また，表示される結果に対し，ユーザは手動にて分類を再設定することも可能であり，変更による各計測値への更新は随時自動で行われる．

・調整要因評価部

技術要因，環境要因それぞれに対して，ユーザがダイアログ上で評価を行い，評価値を入力する．入力された結果をもとに，各調整係数を算出し，画面に表示する．

・UCP測定部

複雑度測定部，および調整要因評価部の結果を受けて，ユースケースポイント値を算出し，画面に表示する．また，その結果見積られる工数(人時で算出)も同時に画面に表示する．ユースケースポイント値から工数値への変換係数は，ユーザによる変更が可能である．

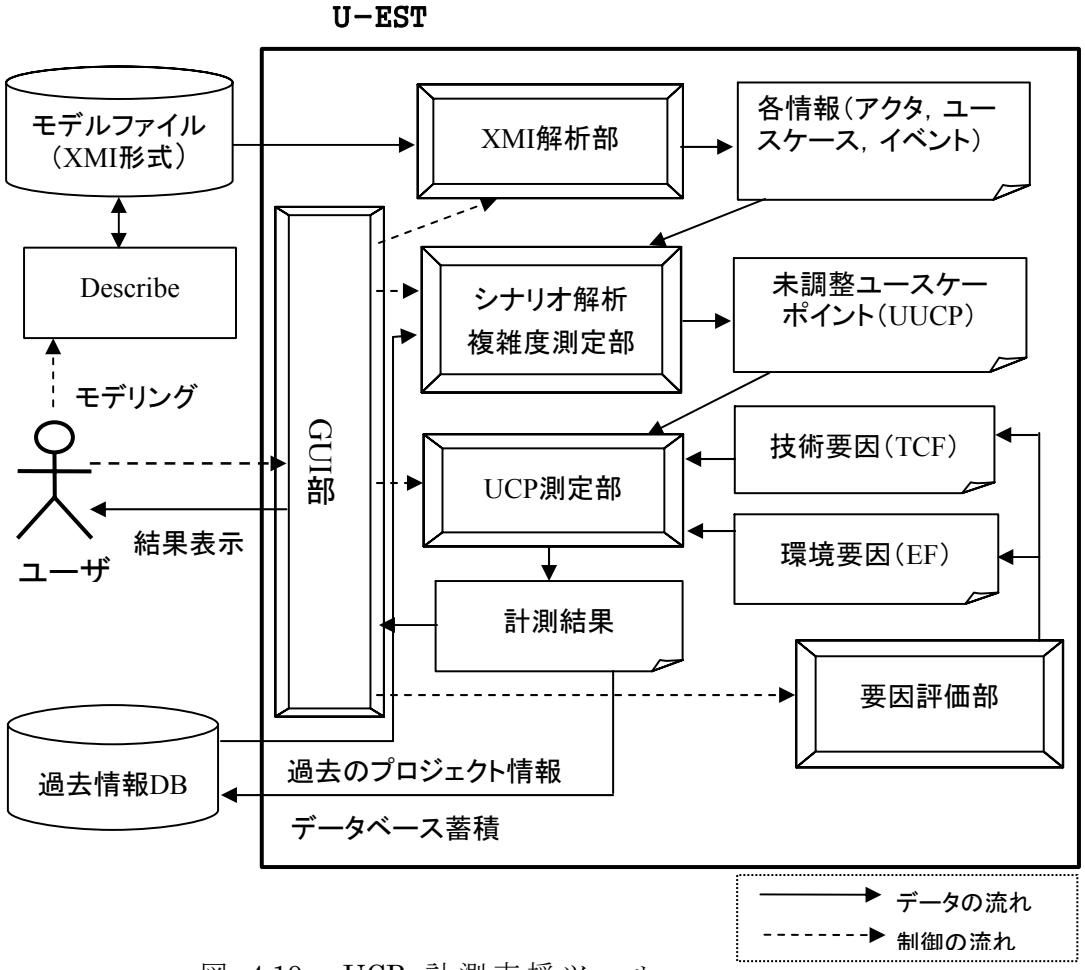


図 4.10　UCP 計測支援ツール

4.3.4　評価

UCP計測支援ツールU-ESTの分類の妥当性評価とツールの適用可能性を評価するため，実プロジェクトで作成したユースケースモデルを用いて，ツールを適用した．評価プロジェクトは，5件で小中規模のWebシステム開発プロジェクトである．表4.4に，その概要を示す．また，表4.5に設定したキーワードやアクタ・システムの動詞のリストを示す．これらのキーワード等の設定は，文献[53]を参考の上，日立SASの技術者と相談して行った．

表 4.4　プロジェクトデータ

プロジェクト	開発言語	アクタ数	ユースケース数
A	Java	5	15
B	Java	5	14
C	Java, VB. NET	2	20
D	Java	5	28
E	Java	8	13

表4.5　キーワードリスト

アクタ名キーワード	システム，　サーバ
単純アクタ用	リクエスト，　要求，　通知，　認証
平均的アクタ(システム)用	メッセージ，　メール，　送信
平均的アクタ(人間)用	コマンド，　テキスト，　入力
複雑アクタ用	ボタン，　押す，　選択
アクタ入力A用	入力，　選択，　指定，　決，　承認，　判定
アクタ入力B用	押す，　押下，　確定，　送信，　要求，　検索，　変更， 修正，　追加，　問い合，　削除，　絞り込
システム出力A用	出力，　表示，　閉じる，　返す，　送信，更新，　求める， 削除，検討，　修正，　参照，　登録，　検索
システム出力B用	認証，　チェック，　記録

計測対象プロジェクトのユースケースモデルにおけるアクタとユースケースの複雑さに関して，U-ESTが計測した複雑さと，UCP計測熟練者の手動で計測した複雑さを比較する．これにより，U-ESTの適用可能性を評価する．アクタ，ユースケースについて，U-ESTと手動での分類結果について表4.6に示す．表4.6は，U-ESTによる分類と，手動での分類の，各複雑さに分類された要素の数を記している．複雑度一致率は，U-ESTで決定された複雑さと手動で決定された複雑さが一致した要素数の，全体数に対する割合である．

表4.6　分類結果

プロジェクト	アクタ							ユースケース						
	手動			U-EST			複雑一致率	手動			U-EST			複雑一致率
	単純	平均	複雑	単純	平均	複雑		単純	平均	複雑	単純	平均	複雑	
A	1	0	4	0	1	4	0.8	13	2	0	13	2	0	1.0
B	3	0	2	2	1	2	0.8	10	4	0	10	4	0	1.0
C	0	0	2	0	0	2	1.0	14	6	0	12	8	0	0.9
D	1	0	4	1	0	4	1.0	27	1	0	27	1	0	1.0
E	0	0	8	0	0	8	1.0	2	8	3	2	8	3	1.0

アクタの複雑度は，一致した割合が0.8～1.0で，高い一致率を達成している．プロジェクトAとBの誤りはアクタ5つ中の1つのずれである．誤りはいずれも外部システムのアクタであった．

技術者の手動による分類では，ユースケース記述の情報には現れない技術者の持っている情報や経験が利用できる．しかし，当然ながら自動分類では，ユースケース記述の情報とキーワード以外の情報は利用できない．キーワードリストを改善して完全一致させる可能性もあるが，他の方法により計測精度を向上する方策の検討が必要である．例えば，プロジェクトでは開発環境，ユーザインタフェイスは一貫したものを使うケースがほとんどであるので，ユーザに「定義済みAPIを使用するか否か」「GUIを介したプログラムであるか」の2点について問うだけで，十分な精度が得られる可能性もある．

ユースケースの複雑度は，高い割合で複雑度が一致している．表4.7に，ユースケース複雑度分類の基になる，トランザクション数を示す．表4.7に示すようにトランザクション数も，ほぼ全てのユースケースについて手動との差が1以内に収まり，熟練した計測者とほぼ同じ結果が得られた．

表4.7　総トランザクション数

プロジェクト	手動での合計	ツールでの合計	合計値の比率
A	85	85	1.00
B	90	90	1.00
C	130	140	1.08
D	145	145	1.00
E	145	145	1.07

ユースケース複雑度分類で差の出たプロジェクトCは，複雑一致率が0．90と他のプロジェクトと比べて低いが，トランザクション数自体は手動と比べてユースケース20個中17個が完全一致し，20個中2個が1のずれ，20個中1個のみが2のずれを出した．他のプロジェクトもトランザクション数の1程度のずれはあるが，通常は3段階の分類に丸め込まれて，複雑度の判定結果では，ずれは検出されていない．なお，プロジェクトCの差の原因は，トランザクション数が「単純(3個以下)」と「平均的(4個から7個)」の境界線を挟むユースケースが2つ検出されたためであった．このように，トランザクション数がタイプの境界値を持つユースケースは，その情報をユーザに呈示して，ユーザに結果の調整をゆだねる方法が現実的である．

また，今回の実プロジェクト評価では，アクタの汎化やユースケースの包含や拡張に関する部分が含まれていなかった．これらについての評価は今後の課題の一つである．

4.4 結論

本章では，ソフトウェア規模の試算見積手法として協調フィルタリング法による予測モデルとUCP 法による自動計測方式を述べた．いずれも支援ツールを試作して，実プロジェクトで評価をした．その結果，高い精度での予測結果を得た．しかしながら，実用化には課題もあり，継続して研究する必要がある．それぞれの成果と課題を述べる．

協調フィルタリングによるソフトウェア規模予測システムの実現には，データの性質を考慮した変数の決定，計算アルゴリズムの選定が重要であり，これらを通して欠陥を含んだデータでも実用的な精度で予測できることを確認した．しかし，実用化には，事業特性やソフトウェア種類別の変数設定や検索アルゴリズムの普遍性評価が必要である．また，組織的に収集するソフトウェア開発データは，開発環境やニーズの変化，組織体制の変更などにより変わるので，収集されなくなったデータ項目は，欠陥値となる可能性がある．本研究ではこのような性質のデータは扱わなかったが，このようなデータ種別の変化への対応も課題である．

UCP 法による自動計測方式では，ユースケースモデルからアクタとユースケースを自動分類する手法と UCP 計測ツール U-EST について述べた．U-EST は，アクタとユースケース分類に幾つかの制限を置いた上で，自動的に複雑度の分類を行う．実プロジェクトでの評価では，U-EST の計測値と熟練者の手動計測値がきわめて近い結果を得た．今回の結果より，当初の目的である，計測のための開発者の負担増を避けること，計測者による誤差を無くすこと，データの統一的な蓄積の基盤の確立が達成できた．今後は，より多くのケーススタディを行い，判定に用いるキーワードリストの再考，アルゴリズムの改良を行い，アクタについて精度の高い複雑度を分類する必要がある．また，ツールによる計測精度を向上させるには，ユーザとのインタラクションは不可欠である．ユーザとツール間のインタラクションに要する工数評価も現場での利用における重要な課題である．

第 5 章　むすび

5.1　まとめ

本研究では，基幹情報システムの生産技術と見積技術のうち，以下の 3 点の研究を行った．特に，（1）（2）は，データ中心型アプローチ(DOA)のコンセプトに基づいて研究した．

（1）基幹情報システム開発方法論と統合開発支援ツール

（2）リバースエンジニアリングによる業務仕様理解支援

（3）ソフトウェア規模の試算見積

（1）では，ソフトウェア開発プロセスモデルと開発技法を標準化した情報システム開発方法論 HIPACE と，それをサポートする統合開発支援ツール EAGLE を開発した．HIPACE は，開発手順, 開発基準, 開発技法を HIPACE マニュアルとして日立グループの技術者に提供している．HIPACE マニュアルのダウンロードは年間 2500 件，Web 閲覧は年間 1 万件を越えている．統一した開発方法論を組織で維持することで，新しい生産技術や国際標準化などに追随することが可能である．

統合開発支援ツールは，実プロジェクトで効果を実測した．プログラム生成率は，従来型のプログラム構造で 81%，3 層アーキテクチャーのプログラムで 71%の効果を確認した．標準スケルトンと処理部品のエンハンスによりプログラム生成率が 20%向上した．品質評価では，EAGLE2 の標準テストケース開発により単体テストで 30%の品質が向上した．教育効果では，EAGLE のエンハンスによりプログラム開発習得効果がでた．例えば，成長率が 20%向上した．またプログラム開発経験回数が 4 回か 5 回で習熟率が上がることが分かった．

（2）では，プログラムのソースコードに記述されているデータ項目に着目して，そこに内在している業務ルールを 2 項オブジェクトにより抽出するデータ中心型アプローチのリバースエンジニアリング DORE を開発した．DORE の評価を顧客の実システムを例題にして分析した．抽出した業務ルールの正当性評価では，仕様書記述の業務ルール抽出が確認でき，また仕様書にない暗黙ルールの抽出(仕様書記述業務ルールの 34%)により本研究の有効性を検証できた．業務ルールの仕様抽出率は，設計仕様書と比較して評価した．その結果，A 社では 74%，B 社ではプログラム設計書で 90%以上の抽出率を確認した. その結果, DORE による業務ルールの抽出により，業務仕様の理解支援，設計仕様書の生成の有効性を確認した．今回の研究では，抽出した業務ルールをソフトウェア部品化する研究はしなかったが，DOA をベースにしているので，業務ルール

を制約条件として統合開発支援ツールのデータ辞書に登録するのは容易であると考えられる.

（3）では，試算見積の手法として協調フィルタリング法による事例ベース類推方式と UCP 法による自動計測方式を開発した.

協調フィルタリングによるソフトウェア規模予測システムでは，6 種類の変数(対応業種，開発言語，画面数，帳票数，ファイル数，一般システム特性)の予測モデルを設定した．またアルゴリズム探索ツールを試作して，最適な検索アルゴリズムを探した．その結果，類似度計算式は相関係数計算法(各変数の値は平均値)とユークリッド距離計算法(各変数の値は中央値)による按分計算式，予測値計算式は，類似プロジェクトの規模の加重平均と倍率修正値の中央値を用いたアルゴリズムを決定アルゴリズム r1 とした．この予測モデルと決定アルゴリズムによりソフトウェア規模を予測した結果，相対誤差平均(MRE)が 0.28 という高い予測を出した．これにより，欠陥を含んだデータでも実用的な精度で予測できることを確認した.

UCP 法による自動計測では，アクタとユースケースのタイプと重み係数の自動分類方式を開発した．アクタの分類ルールを 6 つ，ユースケースの分類ルールを 7 つ設定した．このルールにより UCP を計測する支援ツール U-EST を試作した．評価は，実プロジェクトのユースケースモデルを用いて，U-EST による自動重み付けと経験者による手動重み付けを比較して，ツールの有用性を確認した.その結果，複雑度の一致割合は，アクタで 0.8～1.0，ユースケースで 0.9～1.0 の高い一致率を達成した．トランザクション数についても，手動との差が 1.0～1.08 になり，熟練した計測者とほぼ同じ結果が得られた.

5.2 今後の研究方針

本研究のテーマである開発方法論とソフトウェア部品を中心にした統合開発支援ツール，既存ソフトウェアの活用，ソフトウェア規模の早期見積は，今後も基幹情報システムの開発には重要なキーである．

基幹情報システムは，企業間の電子取引 BtoB(Business to Business)や企業と消費者の電子取引 BtoC(Business to Consumer)と共に，利用者が企業内ユーザから社外ユーザに広がっている．その結果，基幹情報システムは安定した品質や環境の変化に柔軟な構造を持つソフトウェアの開発が要求されている．

今後の研究方針として，開発技法では信頼性，セキュリティや性能など非機能要件の設計支援の研究がある．この分野は，モデル化技法，仕様定義技法，モデルの検証技法など多岐にわたる要素技術の研究が必要である．仕様記述やモデルの検証手法として形式手法の利用もある[92]．試算見積では，本研究で方式研究と試算ツールによる評価をしたが，実用化には，継続した実評価によるモデルとルールの拡充が必要である．これらの研究に取り組んでいきたい．

プロジェクトの実績データを収集・分析して，法則や理論などの知見を発見してプロジェクトへのフィードバックを反復するエンピリカルアプローチ[93]は，ソフトウェア工学の研究には不可欠の要件である．本研究では，3 テーマすべてを実プロジェクトのデータで評価した．今後も実データの蓄積と分析で評価と改善をしていく．

参考文献

[1] 経済産業省が推進するソフトウェア政策について, 経済産業省商務情報政策局, 2005.

[2] 平成 18 年度情報通信白書, 総務省, 2006.

[3] F. P. Brooks, “No Silver Bullet- Essence and Accidents of Software Engineering”, IEEE Computer, Vol.20, No.4, pp.10-19, 1987.

[4] ユーザ企業ソフトウェアメトリクス調査 2007, 日本情報システム・ユーザ協会, 2007.

[5] 原田実, CASE のすべて, オーム社, 1991.

[6] 井上克郎, 松本健一, 飯田元, ソフトウェアプロセス, 共立出版, 2000.

[7] W. W. Royce, “Managing the Development of Large Software Systems：Concepts and Techniques”, Proceedings of WESCON, 1970.

[8] B. Bohem, “A Spiral Model of Software Development and Enhancement”, IEEE Computer, Vol.21, No.5, pp.61-72, 1988.

[9] A Guide to the Project Management Body of Knowledge 2004 edition, Project Management Institute, Inc., Newtown Square, 2004.

[10] Department of Defense handbook Work Breakdown Structure (MIL-HDBK-881), Department of Defense, USA, 1998.

[11] 情報サービス産業における受注ソフトウェア開発の実態アンケート調査結果概要, JISA, 2006.

[12] B. Boehm, “A View of 20th and 21st Century Software Engineering”, ICSE’06, pp.12-29 , 2006

[13] E. Yourdon, L .Constantine, Structured Design, 2nd ed, Yourdon Press, 1978.
（原田実, 久保美沙訳：ソフトウェアの構造化設計,日本コンピュータ協会, 1981）

[14] T. DeMarco, Structured Analisis and System Specification, YOUDON, Inc., 1978.
（黒田訳： 構造化分析とシステム仕様, 日経BP, 1986）

[15] C. Gane, T. Sarson, Structure Analysis Tools and Techniques, Prentice-Hall, 1978.

[16] M. A. Jackson, Principle of Program Design, Acad. Press, 1975.
(鳥居宏次訳：構造化プログラミング設計の原理, 日本コンピュータ協会, 1980).

[17] J. D. Warnier, Logical ConstructioNof Programs, 3rd ed. an Nostrand Reinhold, 1974.

[18] W. P. Stevens, G. J. Meyers, L.L. Constantine, Structured Design, IBM Systems Journal, Vol.13, No.2, pp. 115-139, 1974.

[19] G. J. Myers, Composite/Structured Design, Litton Educational Publishing, 1978.
(國友義久, 伊藤武夫訳：ソフトウェアの複合/構造化設計, 近代科学社, 1979)

[20] E. Dijkstra, Cooperating Sequential Processes, Academic Press, 1968.

[21] 二村良彦, “PAD(Problem Analysis Diagram)によるプログラムの設計及び作成”, 情報処理学会論文誌, 21巻, 4号, pp.259-267, 1980.

[22] P. Coad, E. YoudoNobject Oriented Analysis, 2nd ed, Prentice-Hall, 1991.

[23] S. Shlaer, S. Mellor, Object lifecycles; Modeling the World in States, Prentice-Hall, 1991.

[24] G. Booch, Object Oriented Analysis and Design with Application 2nd ed, Addison-Wesley, 1994.

[25] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. LoresoNobject Oriented Modering and Design, Prentice-Hall, 1991.
（羽生田栄一監訳：オブジェクト指向方法論OMT, トッパン, 1992）

[26] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, Object Oriented Software Engineering: A Use-Case Driven Approach, Addison-Wesley, 1992.
（西野利博, 渡邊克宏, 梶原清彦訳：オブジェクト指向ソフトウェア工学OOSE 復刻版, エスビー・アクセス, 2003）

[27] OMG Unified Modeling Language SpecificatioNoMG, version1.3, 2000.
(OMG Japan SIG翻訳委員会訳, UML仕様書, アスキー, 2001)

[28] UML Resource Page: http://www.UML.org/

[29] 堀内一, データ中心システム設計, オーム社, 1988.

[30] 竹下亨, CASE決定版, 共立出版, 1994.

[31] M.A.CusumaNo, “Japan's Software Factories”, Oxford University Press, New York, 1991.

[32] Y. Matsumoto, “A Noverview of Japanese Software Factories”, Japanese Perspectives in Software Engineering, Addison-Wesley, pp.303-320, 1989.

[33] E. J. Chikofsky, B. L. Rubenstein, “CASE Reliability Engineering for Information System”, IEEE Software, Vol.5, No.2, pp.11-12, 1988.

[34] 竹下亨, ソフトウェアの保守・再開発と再利用, 共立出版, 1992.

[35] 竹下亨, “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”, 情処研報, Vol.93, No.4, pp.1-18, 1993.

[36]C.McClure, “The Three Rs of Software Automation: reengineering, repository, reusability”, Prentice-Hall, 1992.

[37]L.Markosian, “Using an Enabling Technology to Reengineer Legacy Systems”, COMMUNICATION ACM, 37(5), pp.58-70, 1994.

[38] R.S.ArNoldl, Software reengineering, IEEE Computer Society Press, 1993.

[39] 四野見秀明, “保守支援に必要とされるプログラム解析”, 情処ワークショップ論文集, Vol.97, No.1, pp.67-68, 1997.

[40] 情報処理推進機構編, IT ユーザとベンダのための定量的見積りの勧め, オーム社, 2005.

[41] J. McGarry, 古山恒夫, 富野壽監訳, 実践的ソフトウェア測定, 共立出版, 2004.

[42] A. J. Albrech, “Measuring Application Development Productivity”, Proc.of the Joint SHARE GUIDE and IBM Application Development Symposium, pp.83-92, 1979.

[43] International Function Points Users Group (FPUG), Function Point Counting Practices Manual, Release 4.1, 1999.

[44] G. Karner, “Use Case Points - Resource Estimation for Objectory Projects”, Objective Systems SF AB(Rational Software), 1993.

[45] http://www.nesma.nl/japanese/index.htm

[46] 角田雅照, 大杉直樹, 門田暁人, 松本健一, 佐藤慎一, “協調フィルタリングを用いたソフトウェア開発工数予測方法”, 情報処理, Vol.46, No.5, 2005.

[47] S. Kusumoto, F. Matsukawa, K. Inoue, S.Hanabusa, Y.Maegawa, “Estimating effort of use case points: Method, tool and case study”, IEEE Computer Society Proceedings of the 10th International Symposium on Software Metrics, pp.292-299, 2004.

[48] C.E.Walston, J.P .Winters, “Applying Use Cases”, 2nd Edition, Addison Wesley, Winthrop, 2001.

[49] 原田晃, 幕田行雄, 石川貞裕, 大野治, 楠本真二, 井上克郎, “ファンクションポイント法を応用した早期見積技法の提案とそのシステム化”, 電子情報通信学会論文誌D, Vol.J89-D, No.4, pp.755-766, 2006.

[50] 加藤允基, 加藤真一, 珠野知子, 清水健一, “機能整理ファンクションポイント算出ツールの適用による要件定義・見積り精度向上”, 日立システムジャーナル, 第五巻, pp.65-70, 2005.

[51] 高橋光裕, 菱谷淳, ファンクションポイント法ハンドブック, 電力中央研究所研究報告(R04), 電力中央研究所, 2000.

[52] 土方嘉徳, “嗜好抽出と情報推薦技術”, 情報処理, Vol.48, No.9, pp.957-965, 2007.

[53] G.Schneider, J.P.Winters, 羽生田栄一[訳] , ユースケースの適用:実践ガイド, ピアソン・エデュケーション, 2000.

[54] 共通フレーム 2007, IPA/SEC, 2007.

[55] P. P. Chen, “The Entity-Relationship Model: Toward a Unified View of Data”, ACM Trans. Database Systems, Vol.1, No.1, pp.9-36, 1976.

[56] Portable Common Tool Environment (PCTE) Abstract Specification, Standard ECMA-149, 1990.

[57] B. W. Boehm, “Improving Software Productivity”, IEEE Computer, Vol.20, No.9, pp.43-57, 1987.

[58] J.Martin, Strategic Data-Planning Methodologies, Prentice-hall, 1982

[59] B.Boehm, “Software and its impact: a quantitative assessment”, Datamation, pp.48-59, 1973.

[60] 葉木洋一, 北尾修治, 千吉良英毅, 津田道夫, 大野治, 仁平博三, “システム開発支援ソフトウェア, EAGLE”―EAGLE 拡張版 EAGLE2―”, 日立評論, Vol.68, No.5, pp.29-34, 1986.

[61] 大野治, 石田厚子, “データ中心型ソフトウェア開発技法”, 電気学会論文誌 C, Vol.114-C, pp.636-644, 1994.

[62] 角谷一郎, 津田道夫, 葉木洋一, 石坂裕之, “システム・フローを自動生成する開発支援ソフト EAGLE2”, 日経コンピュータ, pp.121-136, 1986.

[63] 松本正雄, ｂｉｔ別冊ソフトウェアモデル化と再利用, pp.5−25, 共立出版, 1995.

[64] 大野治, 旗由香理, 小室彦三, 今城哲二, 古宮誠一, “多次元部品化方式によるソフトウェア開発の自動化ーバッチプログラム用スケルトンの作成とその充分性ー”, 電子情報通信学会論文誌 D-I, Vol.J83-D-I, No.10, pp.1055-1069, 2000.

[65] 大野治, 降旗由香理, 津田道夫, “データ処理部品方式による実践事例”, ソフトウェアのモデル化と再利用, Bit 別冊, pp.84-102, 1995.

[66] C.Jones, Applied Software Measurement, Mcgraw-Hill, 1996.
(鶴保征城, 富野壽訳, ソフトウェア開発の定量化手法, p.245, 共立出版, 1998)

[67] 金根博信, 谷口俊秀, 松井正浩, 松井知之, “部品化の推進による大阪市水道局殿向けシステムの開発”, 第10回ソフトウェア生産における品質管理シンポジウム, 日科技連, pp.95-102, 1990.

[68] 廣田正博, 倉持俊昭, “標準化による生産性と保守の向上―EAGLE2を利用した開発支援システムの確立―”, HITACユーザ会論文誌, pp.477-491, 1992.

[69] 師岡孝次, 習熟性工学, 建帛社, 1985.

[70] 神谷年洋, “コードクローンとは, コードクローンが引き起こす問題, その対策の現状”, 電子情報通信学会誌, Vol.87, No.9, pp.791-797, 2004.

[71] 津田道夫, “CAPSDF におけるリエンジニアリング”, 情処研報, Vol.93, No.4, pp.77-85, 1993.

[72] 四野見秀明, 藤井邦和, “プログラムの自動的な再構造化における限界とその解決法”, 情処研報, Vol.93, No.4, pp.57-65, 1993.

[73] 下村隆夫, プログラムスライシング技術と応用, 共立出版, 1995.

[74] 下村隆夫, “Program Slicing 技術とテスト, デバッグ, 保守への応用”, 情報処理, Vol.33, No.9, pp.1078–1086, 1993.

[75] M.Weiser, “Program Slicing”, IEEE Transactions on Software Engineering, Vol.SE-10, No.4, pp.352–357, 1984.

[76] 四野見秀明, 玉井哲雄, “プログラム解析を提供する API の実現とその適用”, コンピュータソフトウェア, Vol.22, No.1, pp.91-97, 2005.

[77] 友納正裕, 大武和雄, 小泉昌紀, 川崎洋治, 中島震, “COBOL を対象としたプログラムスライス計算方式”, 情処研報, Vol.95, No.11, pp.13-18, 1995.

[78] 山川敦夫, 宮本信夫, 上林高治, 本真希, 秋庭真一, 村信幸, 堀内一, “データ中心分析によるプログラム論理の抽出”, 情処研報, Vol.93, No.42, pp.107-115, 1993.

[79] 津田道夫, 大野治, 秋庭真一, 内藤一郎, 山川敦夫, 堀内一, “DORE(1)－二項分析による既存プログラムからの業務ルール抽出技術―”, 情報処理学会第 52 回全国大会, 6S-3, 1996.

[80] I. Nagaoka, K. SaNou, D. Ikeo, T. Nagashima, M. Tsuda, “A Reverse Engineering Method Experience for Industrial COBOL System”, Proceeding of 4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC’97), pp.220-228, 1997.

[81] Aberdeen Group, Inc., Legacy Applications From Cost Management to Transformation, 2003.

[82] V. Basili, “The Experimental Software Engineering Groupe: A Perspective”, ICSE’00 award presentation, pp.5-10, 2000.

[83] 井上克郎, 松本健一, 津田道夫, 新海平, “産学官連携によるエンピリカルソフトウェア工学プロジェクト EASE”, 日立システムジャーナル, 第五巻, pp.59-64, 2005.

[84] 柿元健, 角田雅照, 杉直樹, 門田暁人, 松本健一, “協調フィルタリングに基づく工数見積もりのロバスト性評価”, ソフトウェア工学の基礎 XI, 日本ソフトウェア科学会 FOSE2004, pp.73-84, 2004.

[85] B.Anda, H.Dreiem, D.I.K.Sjoberg, M.Jorgensen, “Estimating Software Development Effort based on Use Cases - Experiences from Industry”, Fourth International Conference on the UML, pp.487-504, 2001.

[86] J.Smith, “The Estimation of Effort Based on Use Cases”, Rational Software white paper, 1999.

[87] A.Cockburn, 山岸耕二[訳], “ユースケース実践ガイド- 効果的なユースケースの書き方”, 翔泳社, 2001.

[88] CaboCha, Yet Another Japanese Dependency Structure Analyzer, http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/

[89] 工藤拓, 松本裕治, “Support Vector Machine による日本語係り受け解析”, 情報処理学会自然言語処理研究会報告, NL-138, 2000.

[90] K. Ribu, “Estimating Object-Oriented Software Projects with Use Cases”, Master of Science Thesis University of Oslo, 2001.

[91] http://www.embarcadero.com/products/describe/index.html.

[92] 中島震, “ソフトウェア工学の道具としての形式手法”, ソフトウェアエンジニアリング最前線, 近代科学社, pp.27-48, 2007.

[93] A.Endres, D.Rombach, 吉舗紀子[訳], ソフトウェア工学・システム工学ハンドブック, コンピュータ・エージ社, 2005.