T2R2 東京科学大学 リサーチリポジトリ
Science Tokyo Research Repository

論文 / 著書情報
Article / Book Information

題目(和文)	リエンジニアリングにおける既存システム情報の効率的な活用の研究
Title(English)	
著者(和文)	三部良太
Author(English)	Ryota Mibe
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第10993号, 授与年月日:2018年9月20日, 学位の種別:課程博士, 審査員:小林 隆志,佐伯 元司,横田 治夫,権藤 克彦,西崎 真也
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第10993号, Conferred date:2018/9/20, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Powered by T2R2 (Science Tokyo Research Repository)

TOKYO INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

平成 30 年度 学位論文

リエンジニアリングにおける既存システム情報の効率的な活用の研究

東京工業大学
大学院情報理工学研究科 計算工学専攻

三　部　　良　太

平成 30 年 9 月

要旨

本論文は，「リエンジニアリングにおける既存システム情報の効率的な活用の研究」と題し，以下の７章よりなる．

第１章「序論」では，本論文の背景を説明している．企業において 70% 以上の業務が既にシステム化されている．このため，システム開発の際には既存システムの仕様を考慮し，これに新たな要求を組み入れる必要があり，既存システムの実態調査と，新仕様に反映させるために新要求を正確に獲得することに多くの手間がかかっていた．本論文では，既存エンタープライズシステムの再構築を行う際の仕様復元と，新仕様の定義を効率的に行う手法を提案したことを述べている．

第２章「エンタープライズシステムのリエンジニアリングにおける課題」では，第１章で述べた課題を詳しく述べている．第１章で述べた課題を解決する技術分野としてリエンジニアリングがあり，既存システムから処理仕様の復元を可能としている．しかし，処理レベルの仕様は仕様の詳細検討の段階で活用できるが，ユーザとの仕様定義で活用するには内容が細かすぎるため，業務レベルへの抽象化が必要であることを述べている．また，復元した仕様にユーザの要求を組み入れて新仕様を定義する部分は，ユーザとの合意に手間がかかること，検討漏れやあいまいな決定が，開発後半の大きな手戻りにつながることを述べている．

第３章「本研究のアプローチ」では，第２章で述べた課題を解決するためのアプローチを述べている．仕様復元では，正確なドキュメントが残っていないシステムの業務仕様を復元することにコストがかかることが課題がある．これを解決するために，既存システムのログと組織情報を用いて仕様復元するアプローチをとることを述べている．新仕様定義では，漠然としたユーザの要求を業務仕様に定義し合意をとるのに手間がかかる課題がある．これを解決するために，復元した仕様に基づいてプロトタイプを生成し，このプロトタイプを用いてユーザと開発者の仕様確認を繰り返すことで，プロトタイプを構築する労力を削減し，ユーザとの合意を効率的に行うアプローチをとることを述べている．

第４章「システムログと組織情報を活用した業務フロー仕様復元」では，第３章で述べた仕様復元の課題を解決する手法の詳細を述べている．正確なドキュメントが残されてい

ない既存のシステムを理解するために，システムログと組織情報から業務フローを復元する手法を提案している．本手法は，複数のユーザによって行われる業務に対し，一つの事務手続きなどの業務対象にアクセスするユーザが切り替わるポイントに着目することによって，ユーザの業務上の役割（ロール）と代表的なアクションを抽出し，抽象化されたアクティビティを復元することができる．ケーススタディによる評価で，ロール復元精度が 99.5%，抽象化アクティビティの再現率が 99.2% という高い精度で業務フローを復元できていることを示している．

第５章「Web アプリケーションのユースケース駆動プロトタイプによる要求獲得方法」では，第３章で述べた新仕様定義の課題を解決するためのプロトタイプ生成手法の詳細を述べている．本方式では，ユースケースシナリオとプロトタイピングを組み合わせた要求獲得方式を提案している．ユースケースの典型的なシナリオをパターン化した「シナリオパターン」，シナリオパターンの画面実装方法をパターン化した「画面パターン」を用いて，開発者がユースケースシナリオを組み立て，このシナリオからプロトタイプを生成する．このプロトタイプを用いて開発者とユーザが仕様を確認することを繰り返すにより，両者の新仕様への共通理解が可能になる．プロジェクト管理システムの開発で適用実験を行い，従来手法より 3 割多くのユーザの指摘を得られ，後工程に持ち越すと修正範囲の影響の大きい指摘を前もって得ることができることを示している．

第６章「仕様復元とプロトタイプ生成の関係」では，第４章で述べた仕様復元技術と，第５章で述べたプロトタイプ生成技術を組合せることにより，全体として第２章で示した課題を解決することを述べている．仕様復元技術で復元した仕様（業務フロー図，業務機能階層図）をインプットとしてプロトタイプ生成を行うことにより，開発者のユーザとの仕様確認作業を効率的に行うことができる．これにより，既存システムから，現行ユースケースを抽出することで踏襲すべき仕様を明確にし，これに新たな要求を加えた新ユースケースをカスタマイズしてユーザに提示し，そのフィードバックを早期に得ることが可能になることを示している．

第７章「結論」では，本研究で得られた成果を要約し，今後の課題を述べている．本論文では，エンタープライズシステム開発の再構築の際の２つの課題を解決する手法を提案し実現した．これにより，既存システムから現状の仕様が復元でき，仕様から生成したプロトタイプを用いてユーザと開発者が新仕様を早期に合意をとることができるという優れた解決法であると結論づけている．

目次

図目次

表目次

第 1 章

序論

1.1 研究の背景

エンタープライズシステムは対象組織が担っている業務の一部を計算機で支援することによって，業務の効率化や品質向上を行うことを目的としているシステムである．IT 技術の進歩によって多くの組織では既にシステムが導入されている．この中には開発当時から 10 年以上の長い期間使われ続けているものも存在している [57]．これらのエンタープライズシステムは，ハードウェアの老朽化や OS やミドルウェアのバージョンアップやサポート停止などにより，定期的に再構築が必要になることがある．再構築とは，既存システムの仕様をベースに，必要な機能を再定義しシステムを作り直す開発である．近い概念に保守開発があるが，保守開発は現行システムの実装に対して機能追加を行うため，アーキテクチャにかかわるような要求に答えることが困難である．そのような場合でも再開発を行うと，現行システムから仕様をベースにして，機能追加を仕様レベルで行い，この仕様を実装することにより要求を実現できる．

これらのシステムが対象とする業務は，時が経つにつれて変化するが，既に 70 %以上の業務がシステム化されているため [58]，業務全体が刷新されることはまれであり，多くの業務が現行システムの仕様を踏襲する中で一部の業務が要求に応じて変更される．そのため，再開発を行う際には，現行システムの仕様を踏襲しつつ新たな要求を取り入れた仕様を定め開発する必要がある．

しかしながら，既存システムの仕様は明確に文書化されているとは限らない．ソースファイルや実行モジュールしか存在しない場合や，文書化されていても修正や拡張が反映されておらず，実際に動いているシステムとは乖離してしまっていることも少なくない [19]．

リエンジニアリングとは，既存システムを再構築する際に，既存システムから仕様を復元して，復元した仕様をもとに要求を獲得しなおし，獲得した要求から新システムの仕様

を設計し実装するという考え方であり [4]，既存システムから，仕様の復元をおこない（仕様復元），復元した仕様をベースに要求獲得しその結果を新仕様として定義する（新仕様定義）という過程で開発が進むプロセスは馬蹄モデルと呼ばれている．この馬蹄モデルの仕様復元，新仕様定義というサブプロセスに対してそれぞれ以下の技術が提案されている．

1. 仕様復元のサブプロセス
   正確なドキュメントが残っていないシステムの業務仕様を復元する．
2. 新仕様定義のサブプロセス
   漠然としたユーザの要求を業務仕様として記述し，合意をとる．

プログラム可視化技術は，システムのプログラムや実行時情報を用いて，仕様復元を行う技術である．これにはプログラムを解析する静的な可視化技術 [26, 24, 10, 22, 44] と，実行時情報を用いる動的な可視化技術 [35, 7, 23, 13] がある．静的な可視化技術は，処理のロジックを可視化することができる. しかし, ライブラリやミドルウェア等を利用するソフトウェアに対しては十分な復元ができないという課題がある．また，ワークフロー実現のために多数の機能を有するシステムに対しては，機能間の関係は静的に判断できない場合が多く，システムがどのように利用されるかを表す業務フローを復元することが困難である．一方，動的な可視化技術は，業務フロー復元に必要な情報を取得することができるが，ログに含まれる操作を分析の単位とするため，操作単位での詳細な業務フローが復元される．このためシステムが実現する業務の全体像を把握することが困難となり，ユーザとの仕様の議論で用いるには適さない.

プロトタイピングは，要求・仕様のあいまいな部分をあらかじめ簡易的に作ることで，開発におけるリスクを軽減し，新仕様定義を行う手法である [2, 15, 39, 31]．プロトタイピングにより，ユーザはアプリケーションの具体的なイメージを得ることができるが，ユースケースに比べ，プロトタイプの作成と保守にはコストがかかるため，実際のプロジェクトでは適用に踏み切れない場合がある．また，プロトタイプで実物を見せてしまうことによって，仕様確認時にユーザが目につきやすい画面の色や配置などの細かい部分の議論に時間がとられてしまい，要求の本質である業務の流れや業務データの検証を効率的に行うことができないといった弊害も懸念される.

1.2 研究のアプローチ

これまで述べた関連技術は，エンタープライズシステムの開発においてそれぞれ一定の課題を解決しているが，依然として解決に至っていない以下の課題が残っている.

仕様復元の課題に関しては，既存のリエンジニアリング技術によって，処理レベルの仕様の復元は可能になった．しかし，システムのリエンジニアリングの際にユーザとの合意

に用いる業務レベルの仕様の復元には，ヒアリングや仕様確認などの人による作業が必要になる．これを既存システムの情報を用いて自動的に復元することができれば，コスト低減を期待できる．

新仕様定義の課題に関しては，ゼロからユーザが具体的な要求を仕様として明確に提示することができないことも多く，ユーザと開発者の試行錯誤により要求獲得が行われ新仕様が定義される．その際の検討漏れやあいまいな決定が，開発後半の大きな手戻りにつながってしまう．しかし，ユーザは具体的な仕様を提示されれば，その良しあしを判断することは可能である．この性質を利用し，開発前半で，上記で復元した仕様に基づいて，具体的な仕様をユーザにイメージしてもらうためのプロトタイプを開発者が示す．そこで得られたフィードバックを仕様に反映させ，プロトタイプをつくり提示するということを低コストで実現することができれば，効率的に要求を獲得し，仕様を合意することが期待できる．

上記のアプローチを実現するために本研究では，仕様復元，プロトタイプ生成をそれぞれのサブゴールを定めた．仕様復元のサブゴールは，業務レベルの仕様（業務機能階層，業務フロー）を復元することを目指す．エンタープライズシステムの仕様復元では，4 種類の仕様（業務，画面，ロジック，ER モデル）が必要であるが，このうち，既存技術で解決している処理レベルの仕様から，抽象度の高い業務レベルの仕様復元を行う．業務レベルの仕様は次のプロトタイプ生成のサブゴール 2 のインプットとなるものとする．

プロトタイプ生成のサブゴールは，仕様復元のサブゴールの結果を叩き台に，ユーザと開発者があるべき仕様を合意するためのプロトタイプを生成する．ユーザから要求を引き出すために，より具体的にシステムの利用イメージを持ってもらうことが重要である．そのため，復元した仕様をベースにプロトタイプを生成し，ユーザと開発者との共通言語として，新仕様定義に活用する．このことで，早期に仕様を決定する．

1.3 本論文の構成

第 1 章では，ここまで述べてきたように，本研究で扱うエンタープライズシステムの開発における課題と解決アプローチについて簡単に述べた．

第 2 章では, 本論文が扱う課題を詳細に示す．

第 3 章では, 第 2 章で示した課題を解決するために，仕様復元とプロトタイプ生成の 2 つのサブゴールからなる本研究のゴールを示す．

第 4 章では, 仕様復元のサブゴールを解決するために，既存システムのシステムログと組織情報を入力として業務フローを復元する方法について述べる．

第 5 章では, プロトタイプ生成のサブゴールを解決するために，エンタープライズシステムのにおける典型的な業務の流れをパターン化したシナリオパターンを用意する．このシ

ナリオパターンを組合わせてユースケースシナリオを組み立てることで，ユーザとの仕様確認を行うための動く仕様書としてのプロトタイプを生成する方法について述べる．

第 6 章では, 第 4 章と第 5 章で述べた 2 つの技術を組合わせることにより，第 2 章で述べた課題を解決できることを示す．

第 7 章では, まとめと今後の課題について述べる．

第 2 章

エンタープライズシステムのリエンジニアリングにおける課題

2.1　エンタープライズシステム

我々は公私にかかわらずの日々の活動の中で，多くのシステムにかかわっている．1949 年にフォン・ノイマン型計算機 EDSAC[43] が登場して以来，無数のソフトウェアが開発され，その多くが現在も現役で使われている．業務・情報通信業基本調査報告書 [47] によると 2015 年度の日本国内の情報サービス業の会社が約 3500 社あり，17 兆円の売り上げになっている．これらの企業では，毎年何らかのシステムを開発していることになる．

システムは表 2.1 に示すように広い範囲で用いられている言葉である．企業や官庁，自治体の業務を支援する企業の業務を支える「エンタープライズシステム」と呼ばれるものや，ゲームに代表される娯楽のために利用される「エンターテインメントシステム」，OS やミドルウェアのような「基盤システム」と呼ばれるものもシステムに属する．本研究では，エンタープライズシステムに着目する．エンタープライズシステムは対象組織が担っている業務の一部を計算機で支援することによって，業務の効率化や品質向上を行うことを目的としているシステムである．

「エンタープライズシステム」は，金融（銀行オンライン，ATM，チャネル系，損保，生保），公共（年金，税，介護，消防)，産業（生産，在庫管理，物流），一般（給与，納税，勤怠管理，経理，経営管理）などの様々な業務を支援するために開発が進められてきた．日本情報システム・ユーザ協会の調査によると，その割合は 7 割を超えるといわれている [58].「エンタープライズシステム」は 1980 年代からしきりに開発され，この中には開発当時から 20 年〜30 年という長い期間使われ続けているものも存在している．現在運用中の多くのシステムが 15 年に迫るライフサイクルを持っている [57] といわれている．

「エンタープライズシステム」は，企業や公共団体などの組織の業務を支援するという性

表 2.1 システムの分類

分類		説明
アプリケーション	エンタープライズ	企業や官庁, 自治体の業務を支援するシステム．本研究のターゲット.
	エンターテインメント	ゲームなど人に楽しみを与えるシステム.
	組込み	装置や機器を制御するためのシステム.
基盤		OS，ミドルウェア，フレームワークなど上記他のシステムで共通して利用するコンポーネントとなるシステム.

質上，その業務を行う組織から受託型開発されることが多い．情報通信業基本調査報告書 [47] によると受託開発ソフトウェア業の売上高が 8 兆円で，情報サービス業の 47% を占めているといわれている．同調査によると，そのうち 76.5% の企業が外部委託しているというデータもあり，1 つのシステムを開発する際には多くの組織の多くの人がかかわっている．また，その中には数 10 年以上も前に開発されているものも少なくない.

これらのシステムを記述している言語は，COBOL(1959 年発表)，PL/I(1964 年)，C(1972 年) などの手続き型言語，C++(1980 年)，Java(1995 年) などのオブジェクト指向言語，Perl(1987 年)，Python(1991 年), Ruby(1995 年), PHP(1995 年) などのスクリプト言語と多岐にわたっている [42].

エンタープライズシステムは表 2.2 のように分類される [48, 50]．本研究ではエンタープライズシステムの中で，特にユーザインターフェイスを伴い，多くのユーザが連携して行う業務を支援することを目的としたシステムを対象とする．作業毎に複数のユーザがそれぞれの担当範囲 (ロール) を分担し，処理対象が引き回されることで処理が行われるシステムを想定している．これは，エンタープライズシステムのカテゴリではチャネル系や基幹系でよく見られる.

表 2.2　エンタープライズシステムの分類

チャネル系	顧客などのサービスを受けるエンドユーザと内部のサービス（基幹業務）をつなぐインターフェイスを担うシステム.
基幹業務系	その組織の業務のコアとなる業務を遂行するためのシステム.
対外接続系	パートナー組織のシステムや人とのインターフェイスを担うシステム.
情報系	基幹業務が作り出す情報を俯瞰して，次の戦略や戦術を立案するためのシステム.
OA 系	ワードプロセッサやメールなど業務を行うためのツール．業務の色はなく汎用的なシステム.

2.2　エンタープライズシステムの課題

エンタープライズシステムが対象とする業務は，時が経つにつれて変化するが，業務全体が刷新されることはまれであり，多くの業務が現行を踏襲する中で一部の業務が要求に応じて変更される．そのため，現行システムの仕様を踏襲しつつ新たな要求を取り入れた仕様を設計し構築する必要がある．新たなシステムを開発する際には，これらの既存システムの影響を受けることになる．例えば，既存システムとの接続，既存システムの仕様の一部の踏襲，既存システムの性能改善という要件が開発する際に求められる．

しかしながら，既存システムの仕様は明確に文書化されているとは限らない．ソースファイルや実行モジュールしか存在しない場合や，文書化されていても修正や拡張が反映されていない場合に，実際に動いているシステムとは乖離してしまっていることも少なくない．

本研究は，このように既存システムをリエンジニアリングする際に，既存システムの情報を効率的に活用し，システム開発の生産性を向上することを目的とする．

以上で述べたように，エンタープライズシステムの開発には次に示すような要求がある．

1. 大きく業務を変えたくないので多くの機能は踏襲したい．
2. ビジネスの変化への追随や業務効率を向上させる機能は拡張したい．
3. その際，正確な仕様が文書として残っていないシステムに対しても対応したい．

表 2.3　システムモダナイゼーションの分類

ハードウェア更改	互換性のある新しいハードウェアに交換する．性能面では改善するが，運用効率，開発効率，保守性は現状のままとなる．
リホスト	現行との互換性を保つミドルウェア等を導入して，アプリケーションはそのまま移行する．性能，運用効率は向上するが，開発効率，保守性は現状のままとなる．
リライト	開発言語を言語変換等を用いて移行する．モダンな言語になった分開発効率は向上するが保守性向上は限定的っとなる．
リビルド	現行の仕様を踏襲して，新たなプラットフォーム向けに再構築する．すべての性質が改善するが，移行コストが高い．

2.3　エンタープライズのリエンジニアリングに関する従来技術

2.3.1　レガシーシステム・モダナイゼーション技術

これらの課題を解決するためにレガシー・システム・モダナイゼーションという考え方がある [6]．システムモダナイゼーションとは，現行のシステムの資産を活用しつつ，性能，運用効率，開発効率，保守性の高いプラットフォームやアーキテクチャに移行することである．システムモダナイゼーションには，表 2.3 で示す手法が存在する [45].

このうち，ハードウェア更改, リホスト, リライトの３つの手法は, ソフトウェアそのものを温存する手法である. そのためプラットフォーム移行コストは低く押さえることができる．反面, 課題で挙げた, ビジネスの変化に応じた変更を行う際には影響波及調査のために多くのコストがかかってしまうことがある.

リビルドによる再構築は，性能，運用効率，開発効率，保守性を改善できるが，移行コストが高くなってしまう．この移行コストを低減させる目的として, 以下に示す手法が提

案されている.

2.3.2　静的プログラム可視化技術

リビルドにおいて多くのコストを占める，現行システムを理解する作業のために, 古くからシステムを実装しているソースコードを解析することで仕様を得る取り組みが提案されている [26, 24, 10, 22, 44] これらの技術はソースコードを解析することで, 処理のロジックを可視化することができる. しかし, ライブラリやミドルウェア等を利用するソフトウェアに対しては十分な復元ができないという課題がある．また，ワークフロー実現のために多数の機能を有するシステムに対しては，機能間の関係は静的に判断できない場合が多く，システムがどのように利用されるかを表す業務フローを復元することが困難であるという課題がある.

2.3.3　動的プログラム可視化技術

静的プログラム可視化技術では，復元が困難だったシステムの利用状況を把握できる手法として動的にプログラムを可視化する手法が提案されている [35, 7, 23, 13]．これらの動的プログラム可視化技術は，システムログやトレース情報などのシステムを利用する際に取得する実行時の情報に基づくため，業務フロー復元に必要な情報を取得することができる.

しかし，実行時情報は膨大な量となるため，蓄積及び解析方法の効率化が課題となる．また，ログに含まれる操作を分析の単位とするため，粒度が細かすぎてしまう．このため，システムが実現する業務の全体像を把握することが困難となり，ユーザとの仕様の議論で用いるには適さない.

2.3.4　コード自動生成技術

UML などのモデリング言語を用いて，仕様をモデルとして記述し，このモデルからコードを自動生成する技術も多く提案されている [12, 38, 29]．これらの技術は，制御系組込み分野のように，モデルのセマンティックスが確立していて，その実装方法が明確な分野においては非常に強力な技術である．しかし，エンタープライズシステムは，情報操作を行う処理が中心となるため，実装可能なレベルのモデルが実質的にコードと同じ抽象レベルになってしまうことがあり，効果を発揮できない．そのため，ドメインを限定した上で適切な抽象レベルのモデルから実装を決め打ちできる範囲を定めることが重要となっている.

2.3.5　ソフトウェア/サービスパッケージ技術

業務のうち，多くの組織で共通で行われるものに関しては，あらかじめ標準的なシステムを開発し，組織によって異なる部分をパラメータとして切り出すことで，毎回システムを開発しなくてもシステムが実現できるパッケージ技術がある [54, 52, 46]. これらの技術を適用する際には，パラメータとして可変部のチューニングが許されているが，大枠としてはパッケージが標準としている業務プロセスやデータに合わせる必要がある．この業務の標準化が行えるか否かが効果を上げられるポイントとなっている．日本企業に代表される擦り合わせによって業務が遂行される分野においては，業務の標準化に苦戦しパッケージ導入が失敗する例も存在している.

2.3.6　プロトタイピング技術

ユーザにとってシステムの完成イメージを理解し，要求を引き出すために最も有効といわれる方法として，プロトタイピングがある [2, 15, 39, 31]．プロトタイピングは，要求・仕様のあいまいな部分をあらかじめ簡易的に作ることで，開発におけるリスクを軽減する方法である．プロトタイピングにより，ユーザはアプリケーションの具体的なイメージを得ることができるが，ドキュメントベースの仕様記述であるユースケースなどに比べ，プロトタイプの作成と保守には，コストがかかるため，実際のプロジェクトでは適用に踏み切れない場合がある．また，プロトタイプで実物を見せてしまうことによって，仕様確認時にユーザが目につきやすい，画面の色や配置など細かい部分の議論に時間がとられてしまい，要求の本質である業務の流れや業務データの検証を効率的に行うことができないといった弊害も懸念される．プロトタイプのメンテナンス性，本質的要求の検証の問題を解決する方法には，紙に書いたラフスケッチを基に要求を獲得するペーパプロトタイピングの方法がある [30]．紙で描いたラフスケッチなので，作成が容易な反面，合意を取った仕様の厳密性に課題が残る.

2.4　エンタープライズシステムにおける既存技術の課題

前節で述べた技術は，エンタープライズシステムの開発において一定の課題を解決しているものの，完全な解決に至っておらず以下の課題が残っている.

1. 仕様復元の課題
   正確なドキュメントが残っていないシステムにおいて，業務レベルの仕様の復元にコストがかかる. 既存のリエンジニアリング技術によって，処理レベルの仕様の復

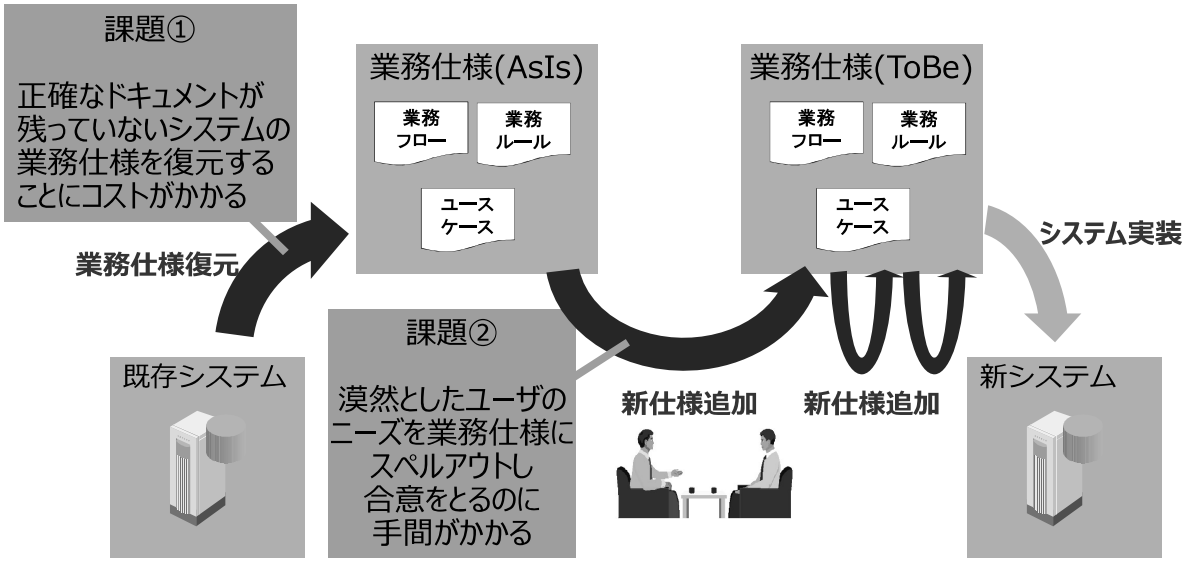


図 2.1　開発現場の課題

元は可能になったが，システムのリエンジニアリングの際にユーザとの合意に用いる業務レベルの仕様の復元にはヒアリングや仕様確認などの人による作業が必要になる．

2. 新仕様定義の課題

漠然としたユーザのニーズを業務仕様として記述し合意をとるのに手間がかかる．ユーザは具体的な仕様を提示されれば，その良しあしを判断することは可能である．ゼロからユーザが具体的な要求を仕様として明確に提示することができないことも多く，ユーザと開発者の試行錯誤により要求獲得が行われ，新仕様が定義される．その際の検討漏れやあいまいな決定が，開発後半の大きな手戻りにつながってしまう．

第 3 章

本研究のアプローチ

本論文は，既存システムの情報を活用してシステムのリエンジニアリングを効率化するために，図 2.1 で示した課題を解決するアプローチを図 3.1 に示す．

まず，既存システムの情報を活用して，そのシステムに関する業務の仕様を復元する（仕様復元）．次に，仕様復元で回復した仕様にユーザの要求を取り入れたプロトタイプを生成し，新仕様の叩き台とする．これを開発者がユーザに提示することで，さらなるユーザ要求を引き出す．引き出したユーザの要求は新仕様に反映させて，叩き台をブラッシュアップする．このプロセスをユーザと開発者がともに納得するまで繰り返すことで新仕様を確定させる（プロトタイプ生成）．

ここで示したアプローチを実現させるために，仕様復元とプロトタイプ生成とをサブゴール (SG) に定めた．

SG1 仕様復元　　業務レベルの仕様（業務機能階層，業務フロー）を復元する．

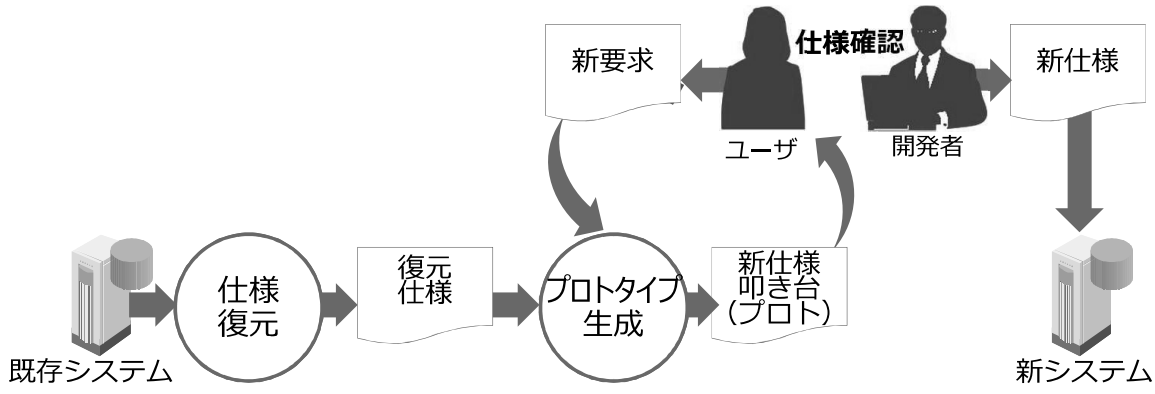


図 3.1　アプローチ

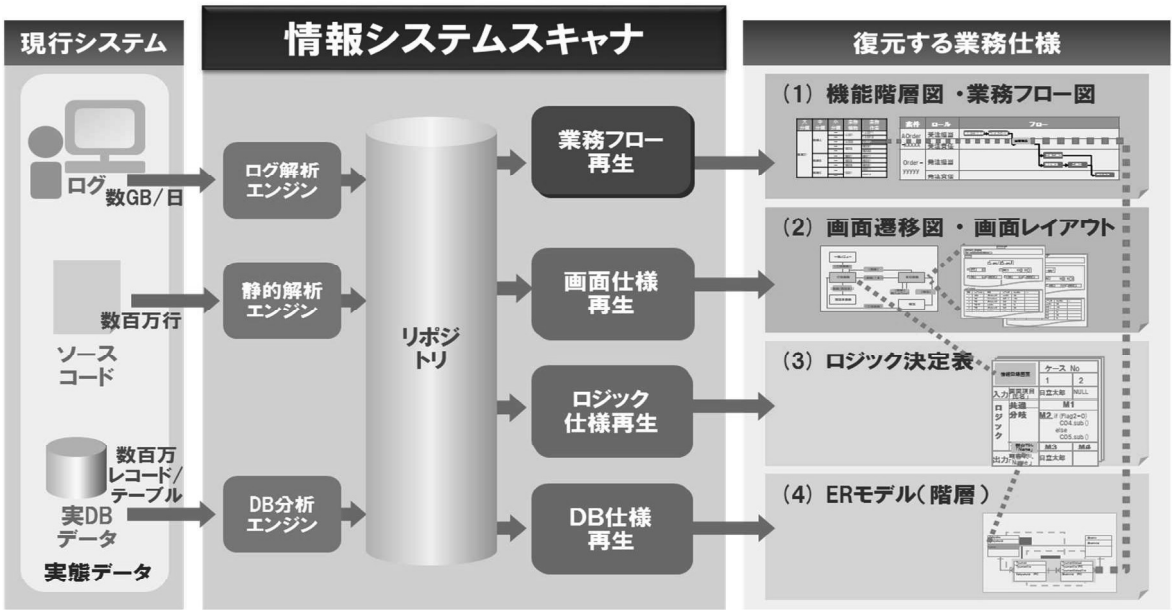


図 3.2　仕様復元

エンタープライズシステムの仕様復元では，４種類の仕様（業務，画面，ロジック，ER モデル）が必要である (図 3.2) が，このうち，画面，ロジック，ER モデルに関しては，前述した既存技術での復元が可能である．本研究では，最後に残された業務レベルの仕様復元を行う．業務レベルの仕様は SG2 のインプットとなるものとする．これらを組合わせることによって，既存のエンタープライズシステムの理解に必要な仕様復元を行う技術体系「情報システムスキャナ」が完成する．

SG2 プロトタイプ生成　　SG1 の結果を叩き台に，ユーザと開発者があるべき仕様を合意するためのプロトタイプを生成する．

ユーザのシステムに対する要求は初めの段階では漠然としていて，ユーザが明確に表現できないことが多い．このような状況で，ユーザからの要求を引き出すためには，より具体的にシステムの利用イメージを持ってもらうことが重要である (図 3.3)．そのため，復元した仕様をベースにプロトタイプを生成し，より具体的な動く仕様案としてユーザに提示する．これにより，ユーザと開発者が具体的な仕様を共通言語をとし，ユーザからの質の高いフィードバックを得ることで，要求獲得を効率的に行う．獲得した要求は，新仕様に取り込んで再度プロトタイプを生成することを繰り返すことで，ユーザと開発者が誤解のない新仕様の定義を行えるようにする．

本研究では，上記の２つのサブゴールを達成することで，エンタープライズシステムの

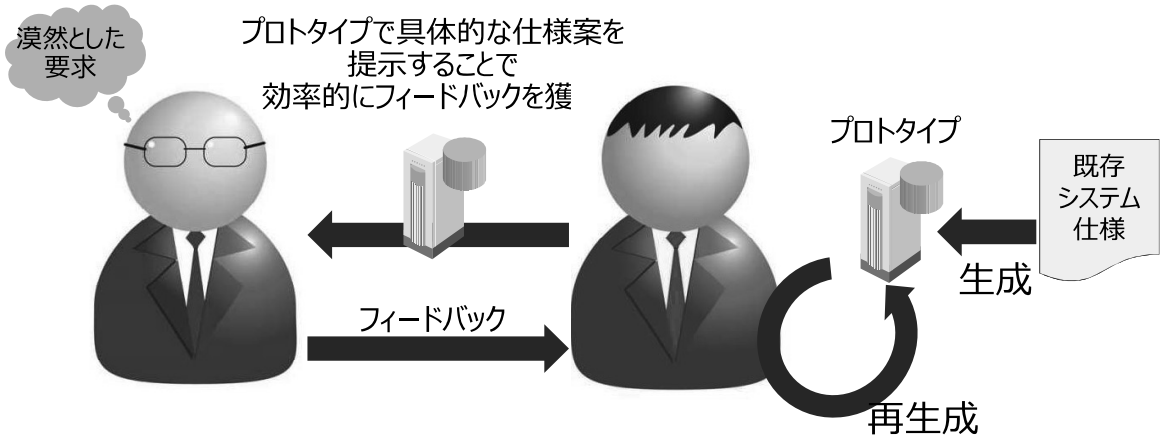


図 3.3　プロトタイプ生成

抱える課題を解決する．次章以降でそれぞれのサブゴールについて詳細に述べる．

第 4 章

システムログと組織情報を活用した業務フロー仕様復元

4.1 業務フロー仕様復元の課題

近年，企業の活動においてシステムが導入されている割合は約 7 割といわれている [58]. 企業において業務の多くは変化しないため，新たに開発する仕様は既に利用しているシステム (以下，既存システム) の仕様の一部を踏襲することが求められる．しかし，古くから運用している既存システムの仕様は，明文化されていることが少ない [19]．そのため，既存システムの仕様の理解に多くの人的コストが必要となり，システム開発コストを圧迫している．

この課題を解決するために，様々な仕様復元手法が提案されている．これらの手法は，ソースコードを入力とする静的解析に基づくものと，実行ログ等の実行時情報を入力とする動的解析に基づくものに大きく分類される．静的解析に基づく手法は，プログラムの静的構造と関数呼び出し関係などの依存関係に基づきドメインモデルやソフトウェアアーキテクチャの復元を行うものである [26, 24, 10, 22, 44]．これらは，ソースコードを用いることで網羅的に復元できる一方で，ライブラリやミドルウェア等を利用するソフトウェアに対しては十分な復元ができないという課題がある．また，ワークフロー実現のために多数の機能を有するシステムに対しては，機能間の関係は静的に判断できない場合が多く，システムがどのように利用されるかを表す業務フローを復元することが困難である．

動的解析に基づく手法 [35, 7, 23, 13] は，システム利用における実行時情報に基づくため，業務フロー復元に必要な情報を取得することができるものである．

実行時情報は一般に膨大な量となるため，蓄積及び解析方法の効率化が課題となる．また，ログに含まれる操作を分析の単位とするため，粒度が細かすぎてしまう．このためシステムが実現する業務の全体像を把握することが困難となり，ユーザと開発者で行う仕様

の議論で用いるには適さない．

本章では，低コストで蓄積可能な運用時のシステムログに着目し，システムログとシステム利用者の組織情報を用いることで，業務の全体像を俯瞰する業務フローを復元する手法を提案する．運用時のシステムログを解析することにより，そのプロセスに関与するアクター (以下ユーザ) の網羅的な実行時情報を取得し，それを分析することで抽象化したアクティビティを復元する．また，組織情報に基づいてアクティビティを階層化することにより全体像の俯瞰に適した抽象化を行う．さらに提案手法を実際のシステムに適用したケーススタディで有効性を評価する．

以降，第 4.2 節では関連研究について説明し，第 4.3 節では提案手法が想定するシステムとそのシステムログについて説明する．第 4.4 節で提案手法，第 4.5 節で提案手法を実現したプロトタイプについて述べる．第 4.6 節では，実際の業務アプリケーションに適用したケーススタディについて説明し，第 4.7 節で結論と今後の課題について述べる．

4.2　関連研究

システムログからフローを復元する取り組みは van der Aalst らによるビジネスプロセスマイニングの研究がある [36]．これらは，ログに含まれる操作の系列から操作間の関係をビジネスモデルとして復元する．しかし，前章で述べた目的で業務を俯瞰するためには，ログに出現する操作の単位では細かすぎてしまい俯瞰することには適さない．

これに対して，ログデータ全体をいくつかに分類し，分類されたログごとにフロー図を生成する方法 [41] や，クラスタリング手法等を適用することで複雑さを軽減する技術 [9] が研究されている．例えば，ActiTraC[41] では，能動学習の考え方を応用し，ログをベクトル空間モデルに変換し，トレース間の距離と出現頻度をもとに選択的サンプリングを行うことで，ログデータのクラスタリングを行う．クラスタリングされたログデータを既存のビジネスモデルマイニングに適用することによって再現性を大きく低下させることなく，複雑さを軽減させている．また，Francescomarino らは Web アプリケーションを対象として，実行トレースから復元された詳細なフローを GUI 情報と繰り返し構造に着目したクラスタリングを適用して抽象的なフロー表現する BPMN モデルの復元手法 [9] を提案している．

我々も，これまでにログデータに含まれるユーザに着目し，抽象化を行うアルゴリズム [18] を提案し，業務フロー生成工数が削減することを示した [56]．本章で提案する手法では，システムログに加えて，ユーザとその組織情報を用いることで業務の実態に即した業務フローを復元する点が異なる．

Time Stamp	User ID	Action ID	Case ID
7/1 10:00:00:00	佐藤	見積登録	00100
7/1 10:00:01:00	佐藤	登録エラー	00100
7/1 10:00:02:00	佐藤	見積登録	00100
7/1 10:00:05:00	佐藤	見積承認依頼	00100
7/1 10:00:06:00	小泉	見積登録	00101
7/1 10:00:06:00	小泉	見積承認依頼	00101
7/1 11:00:05:00	鈴木	承認依頼確認	00100
7/1 11:00:07:00	鈴木	見積承認	00100
7/1 11:00:05:00	鈴木	承認依頼確認	00101
7/1 11:00:07:00	鈴木	見積承認	00101
…	…	…	…

図 4.1 システムログの例

4.3 想定するシステムとシステムログ

4.3.1 想定するシステム

本研究で想定するシステムは，ユーザインターフェイスを伴い，多くのユーザが連携して行う業務を支援することを目的とした業務システムである．作業毎に複数のユーザがそれぞれの担当範囲 (ロール) を分担し，処理対象が引き回されることで処理が行われるシステムを想定している．これは，業務システムのカテゴリ [48, 50] ではチャネル系 (お客様とのやり取りのインターフェイスを担うシステム) や，バックオフィス または基幹系（社内の事務処理を行うシステム) でよく見られる．これらのシステムでは，ユーザによっては状況に応じて複数のロールを担当することがある．各ユーザは職務に応じた組織 (営業部，在庫管理部等) に所属し，組織は階層的な組織構造を持つ．

4.3.2 システムログ

システムログは，システム運用時における利用状況を記録したものである．大規模システムでは，システムの障害時に状況を把握したり，サービスレベルの計測に用いられたりするために一定期間残しておくことが一般的である．システムログには次の (1)〜(4) の情

報が含まれていることを想定する．図 4.1 に (1)〜(4) の具体的な情報が含まれたシステムログの例を示す．

(1) Time stamp : この操作が行われた時刻を示す．
(2) User ID : この操作を行ったユーザを示す．後に述べるユーザ情報と対応付けるための情報でもある．
(3) Action ID : 新規作成，登録，承認，修正など，行った操作の種類を示す．
(4) Case ID :案件番号，登録番号，受付番号，注文番号など，操作を行った対象 (以降，案件と呼ぶ) を示す．

4.3.3　組織情報

業務フローを階層化する際には「事業レベルから担当者が行う作業レベルまで，階層的に業務の流れを単純化して表現」[60] する．この階層を復元するために組織情報を用いる．組織情報は User ID で識別されるユーザの所属する組織構造（会社，事業部，部，課，担当など) やロールを示す情報である．組織情報は木構造で表現されることが多い．ユーザは必ず 1 つ以上の組織に属し，1 つ以上のロールを持っている．ユーザの担当は，システムのユーザ権限で定義されることも多く，この情報を利用してロールを復元する．

4.3.4　対象とする業務フロー

業務フローの復元にあたり，業務とは何かを定義する．本論文では，業務を「1 つの目的に対して特定のユーザや組織が行う一連の活動」と定義する．この定義は van der Aalst らが発表したプロセスマイニングマニフェスト [37] で示されている activity の定義 (“a well-defined step in some process, related to a particular case") よりも粒度が大きい概念である．上記の定義において「1 つの目的」がシステムログ上の Case ID に対応し，「ユーザや組織」が User ID と組織情報に対応する．本章で示す手法では，この性質を利用して抽象度の高い業務フローを復元する．

業務フローの抽象化は，複数のアクションから抽象化したアクティビティ（抽象化アクティビティ) を抽出することで抽象化が行われる．しかし，複数の組織にまたがる大規模な業務では，アクティビティレベルの抽象化だけでは全体を俯瞰するには不十分である．そこで，本論文では，抽象化アクティビティに加えてロール，組織も復元することでこのような大規模な業務も対象とする．

本章で提案する技術の目的は，「大規模なシステムの仕様をマクロ的な視点から俯瞰して理解する」ことにある．この目的を達成するためには，下記の要件を定めた．

要件 (1) ビジネスプロセスに登場する多数のアクションを数個からなる抽象化アクティビティにグループ化する．

要件 (2) 抽象化アクティビティとアクションの関係が明確になっている．

要件 (3) 復元した抽象化アクティビティによって業務全体が網羅されている．

要件 (4) 抽象化したアクティビティや抽象化アクティビティ間の遷移は，極力過不足がないものにする．

要件 (1) は，Cowan らの「成人のワーキングメモリの容量は 3〜5 項目」という知見 [20] により，多くのアクションを含む業務フローを一度に理解することが困難であるためである．また，牧野の「ビジネスプロセス・フロー分析では，事業レベルから担当者が行う作業レベルまで，階層的に業務の流れを単純化して表現」というビジネスプロセス分析の一般的な手法 [60] にも則っている．要件 (2) と要件 (3) は，全体を理解することを目的としているため，抽象化したフローで全体を俯瞰したあと最終的にはブレイクダウンしてアクションレベルでの理解も可能にするための要件である．要件 (4) は，人が物事を理解する時には，与えられた情報から存在しないものを見つけるよりも，不要なものを見つけるほうが容易である．このために必要な情報をできるだけ復元するために必要な要件である．

4.4 提案手法

4.4.1 提案手法の概要

図 4.2 に本章で提案する業務フロー抽象化手法の概要を示す．提案手法は次のステップで構成される．入力情報からロールを復元する「Step1:ロールモデル復元ステップ」，復元したロール内のアクション群からそのロールを代表するアクションを抽出する「Step2:代表アクション抽出ステップ」，抽出した代表アクションに他のアクションを紐づける「Step3:抽象アクティビティ復元ステップ」，復元したアクティビティを用いて業務フローを描画する「Step4:フロー描画ステップ」である．

4.4.2 業務機能階層

本章で提案する手法では，次の 4 つの階層を持つ業務機能階層を用いて抽象化を行う．組織における業務では，所属するメンバにそれぞれの役割（ロール) が与えられ，そのロールを果たすためにシステムを利用する．提案手法では，システムのユーザにとってわかりやすい仕様を復元するために，ユーザになじみの深い「組織-ロール-アクティビティ-アクション」の階層に沿った抽象化を行う．これにより，粒度の細かい「アクション」から，粒度の粗い「組織」を段階的に抽象化することができ，全体像を容易に把握できるように

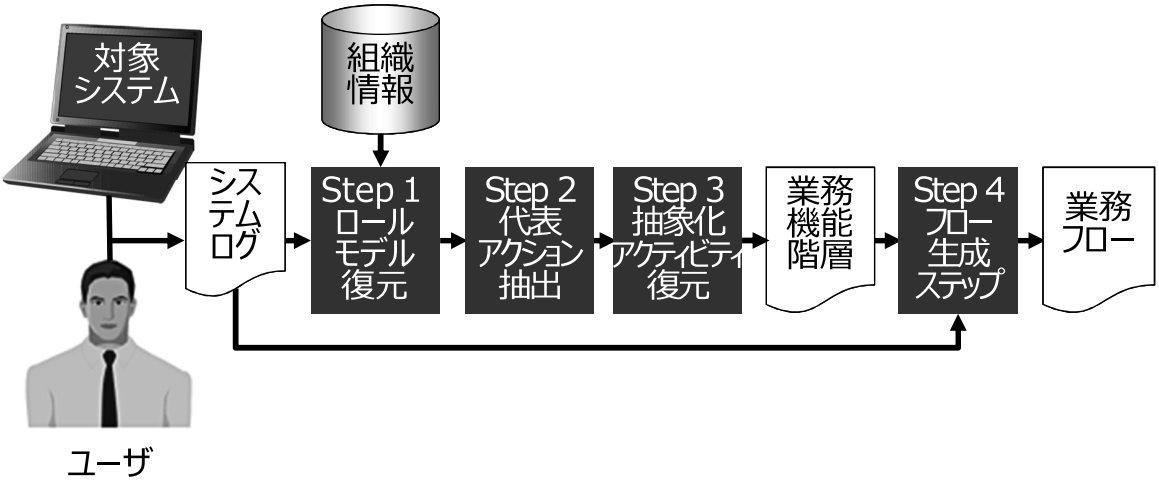


図 4.2　提案手法の概要

なる.

(1) 組織:ユーザの組織．必要に応じて組織がさらに階層化されることもある.
(2) ロール:組織においてのユーザの役割．ユーザ権限に対応することが多い.
(3) アクティビティ:ロールにおけるユーザの複数のアクションを抽象化したもの.
(4) アクション:ログに現れるユーザの操作.

4.4.3　諸定義

まず，システムログ E をログエントリ e の集合として定義する． ログエントリ $e_i \in E$ は対象システムでのユーザ操作の記録に対応し次のように定義する:

$$e_i = (e.u \in U,\ e.t \in T,\ e.a \in A,\ e.w \in W) \tag{4.1}$$

ここで， $U = \{u_1,\ u_2,\ u_3,\ \cdots\}$ はユーザの集合，T はタイムスタンプの集合（システムにおいていつユーザがこの操作を実行したかを示す)，$A = \{a_1,\ a_2,\ a_3,\ \cdots\}$ はアクションの集合（ユーザが行った操作の種類)，$W = \{w_1,\ w_2,\ w_3,\ \cdots\}$ は案件の集合 (操作の対象) を示す.

各ログエントリ e は，ユーザ $e.u$ が案件 $e.w$ に対して，対象システム上で時刻 $e.t$ にアクション $e.a$ を実行したことを表現する．ログエントリを案件単位で時系列順に集めることによって，その案件に対する全ユーザのアクションの履歴を得ることができる．この履歴をプロセスインスタンス (p_i) と呼ぶ:

$$p_i = \langle e_1, e_2, \cdots, e_n \rangle.\ (i < j \Rightarrow e_i.t < e_j.t,\ e.w = w_i) \tag{4.2}$$

業務機能階層の 1 つであり復元した業務フローの主要な要素となるアクティビティ $act \in ACT$ はアクションの集合として定義する．本研究では，1 つのアクションが複数のアクティビティに属することを許す ($ACT \subset 2^A$).

ロール $r \in ROLE$ もアクションの集合として定義する ($ROLE \subset 2^A$)． ロールはアクティビティの上位概念であり，復元した業務フローのスイムレーンを構成する．アクティビティと異なり，1 つのアクションは必ず 1 つのロールに所属する．

組織 $org \in ORG$ は所属するユーザの集合として定義する ($ORG \subset 2^U$).

4.4.4 Step1:ロールモデル復元ステップ

図 4.2 で示したロールモデル復元ステップでは，以下を実行することで，ロールとアクションの関係を復元する．

(1) システムログからプロセスインスタンスを抽出し，さらにユーザが切り替わる箇所で分割することで，ブロック (同一ユーザによる連続アクション) を特定する．
(2) ２つのアクションの同一ブロックでの共起関係を用いて有向関係 R を求め，R の連結成分となるアクションの集合をロールとする．
(3) 組織情報を用いて，ロールの名称や上位の構造化を行う．

以下では各ステップの実行を詳しく説明する．

Step1-1:ブロックの特定

業務システムは，複数のロールのアクティビティによって 1 つの案件を処理していく．この一連の操作の列がプロセスインスタンスとなる．ロールモデル復元の最初のステップとして，プロセスインスタンスからアクティビティの候補となるブロックを特定する．

ブロック $b_{i,j} \in B$ はプロセスインスタンス p_i 上で連続する同一ユーザのログエントリ列と定義する:

$$b_{i,j} = \{e_k \in p_i | j \leq k \leq n,\ e_k.u = e_j.u,\ e_{n+1}.u \neq e_j.u\} \tag{4.3}$$

図 4.3 に図 4.1 のシステムログから抽出したプロセスインスタンスとブロックの例を示す．図において，縦軸はプロセスインスタンスを示し，横軸はアクションを示す． 破線で括られている箇所がブロックである．ブロックは，1 つの案件に対して 1 人のユーザが連続して行った一連のアクションを表している．大規模なビジネスアプリケーションでは，案件 c_{00100} のユーザ「佐藤」と c_{00104} のユーザ「小泉」のように，同じロールを持つ複数のユーザが同じアクションを行うことがある．このため，本研究では，システムログを一度プロセスインスタンスに分解した後に，さらにユーザ単位で分割したブロックを分析の基

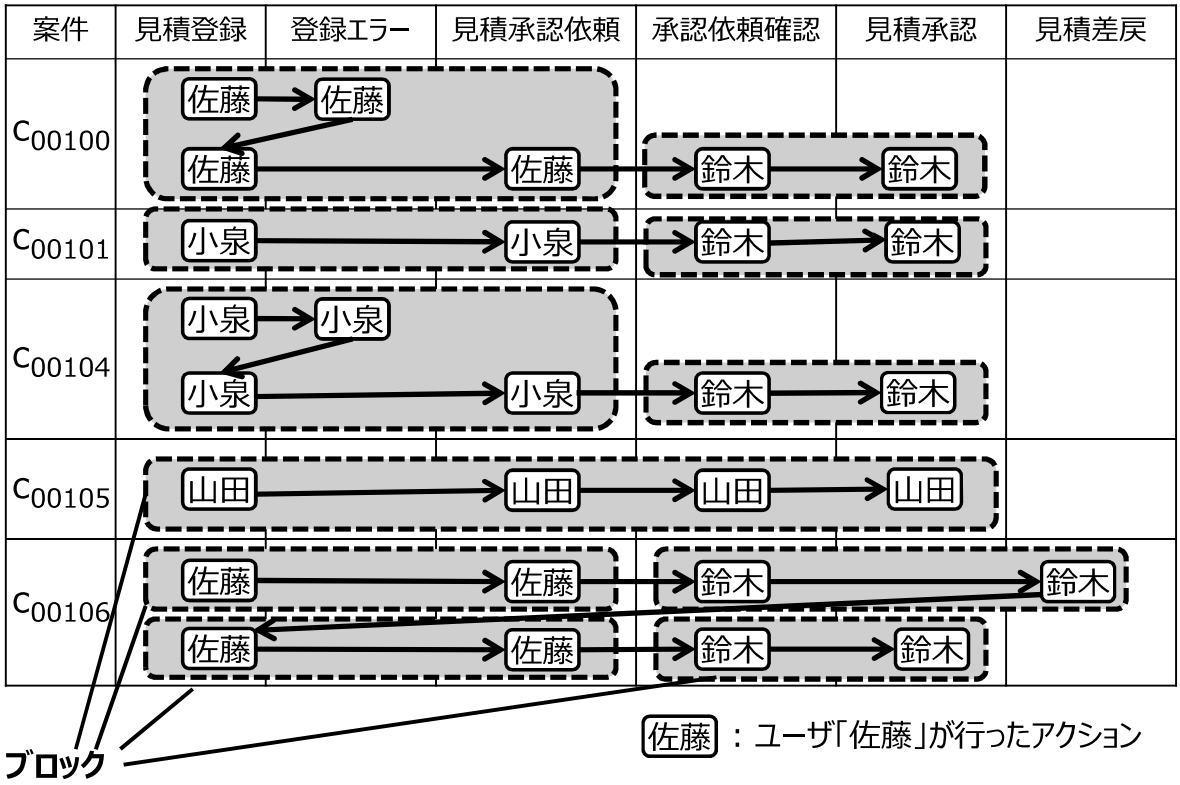


図 4.3　プロセスインスタンス，ブロックの例

本単位として採用している．ここでいうブロックは，ビジネスプロセス・フロー分析 [60] においてフローを階層的に表現する際の最小単位であるリーフプロセスの候補と考えることもできる．

同じロールを持つユーザは同じようなアクションを行う傾向があるため，ブロック単位で類似するアクションを集めることでロールとアクションの関係を推定できる．しかし，ブロックからロールを復元する際の問題として，案件 c_{00101}，c_{00104} のユーザ「小泉」のように必ずしも毎回同一のアクション群を行うわけではない．また，案件 c_{00105} のユーザ「山田」のように，複数のロールを担当するユーザが他の案件でユーザが切り替わる部分を超えてアクションを行うこともある．この問題に対応するために次項で示す手法が必要となる．

Step1-2:ロールの復元

様々なアクションを含むブロックからロールと関係するアクション集合を特定するために，ブロックの中で同時に登場するアクション間の有向関係 R を定義する．

本研究では，この関係に lift を用いる．lift はルールマイニングの分野でルールを構成する要素間の関係の尺度として用いられている [1][16]．提案手法では，アクション a_i からアクション a_j への関係の重み *weight* に lift 値を用い，次のように定義する．

組織	ロール	抽象化アクティビティ	アクション
	ロール1		見積登録 登録エラー 見積承認依頼
	ロール2		承認依頼確認 見積承認 見積差戻
	ロール3		見積受付 在庫確保依頼
	ロール4		在庫確保受付 在庫割り当て 割り当て失敗

図 4.4 ロールの推定結果

$weight(a_i \Rightarrow a_j) = lift(a_i \Rightarrow a_j)$ ここで，$aBlocks(a_i)$ はアクション a_i を含むブロックの集合を示す関数とする:

$$lift(a_i \Rightarrow a_j) = \frac{conf(a_i \Rightarrow a_j) \cdot |B|}{|aBlocks(a_i)|} \tag{4.4}$$

$$conf(a_i \Rightarrow a_j) = \frac{|aBlocks(a_i) \cap aBlocks(a_j)|}{|aBlocks(a_i)|} \tag{4.5}$$

$$aBlocks(a_i) = \{b \in B | \exists e \in b,\ e.a = a_i\} \tag{4.6}$$

関係 $a_i R a_j$ を，$weight(a_i \Rightarrow a_j) \geq MIN_{weight}$ となる 2 つのアクション間の関係とし，関係 R による連結成分をロール $r \in ROLE$ とする．つまり，r 内のアクション a_i は以下の性質を満たす:

$$\forall a_i \exists a_j \in ROLE.(a_i R a_j \vee a_j R a_i) \tag{4.7}$$

提案手法のロール復元手法は，同じユーザによって同時に行われるアクションは，同じロールのアクションであることが多いという性質に着目している．一般にログに含まれるアクションの出現頻度には大きなばらつきがある．これに対応するため，lift 値を用いることでクラスタリングの際に頻度の大きいアクションの過度な影響を除去することができる．結果として，アクションのバリエーションを含むクラスタを構成することが可能になる．また，複数ロールを持つユーザの影響を低減する効果もある．

例えば，図 4.3 のプロセスインスタンス c_{00105} は 1 つのブロックに 4 つのアクションを含んでいる．しかし，他の案件では異なるユーザが行っているため，アクション「見積承認依頼」から「承認依頼確認」への lift 値が低くなる．一方，「見積登録」から「見積承

組織	ロール	抽象化アクティビティ	アクション
営業部	承認依頼者		見積登録 登録エラー 見積承認依頼
	承認者		承認依頼確認 見積承認 見積差戻
在庫管理部	承認依頼者		見積受付 在庫確保依頼
	承認者		在庫確保受付 在庫割り当て 割り当て失敗

図 4.5　ロールモデル

認依頼」への lift 値は高くなる．したがって，図 4.4 に示すロール推定結果では 「見積承認依頼」と 「承認依頼確認」は異なるロールに属し，「見積登録」と 「見積承認依頼」は同じロールに属することになる．

Step1-3:組織情報の反映

大規模システムの業務フローの全体像を理解するためには，より高いレベルの抽象化を実現する必要がある．そのために，ユーザの情報だけでなく，組織情報（担当，課，部，事業部，アクセス権限など) を利用する．提案手法では，そのロール r に含まれるアクションを実行しているユーザが所属する組織に着目し，最も多くのユーザが所属する担当や組織 o_{max} をそのロールの組織に割り当てる:

$$g = \{e.u | a_i \in r, e.a = a_i\}, \qquad (4.8)$$
$$\forall o_i \in ORG, |g \cap o_{max}| \geq |g \cap o_i|$$

まず，上記アルゴリズムで，アクセス権限の情報を使ってロール名を特定した後に，同じアルゴリズムを用いて，階層的に上位の組織名を特定していく．

図 4.4, 図 4.5 にこのステップを適用した例を示す．図 4.4 におけるロール 1〜4 は，組織情報を用いて，アクションを実行しているユーザの多くが属するロール名が割り当てられる． つまり，図 4.5 の 4 列目のアクションを実行しているユーザ「佐藤」,「小泉」が承認依頼者,「鈴木」,「山田」が承認者のアクセス権限を持っていることから図 4.5 の 2 列目のようにロール名を割り当てられる．

4.4.5　Step2:代表アクション抽出ステップ

ロールモデルから抽象度の高い業務フローを復元するために，ロールを構成するアクションの中から代表的なアクションを見つけ出し，これをアクティビティとして業務フ

ローの要素にする．

ビジネスアプリケーションのユーザは，「ログイン」「メニュー選択」のような一般的なアクションから操作を開始し，さまざまなアクションを行った後，「注文確認」や「承認依頼」などのロールに特徴的なアクションで終了する傾向がある．これは，一連のアクションを行った後に次のユーザへ引き渡す（もしくは引き渡さない) ことを最終判断するアクションであり，一連のアクションを代表すべきアクションであると考える．システム内に保存されている情報に対して，ユーザは一連のアクションにより様々な変更を行う．アクティビティを複数人が更新・参照する情報に対するトランザクションとしてとらえた場合，「コミット」「ロールバック」のトリガになるアクションが代表とみなされることとなる．この性質に着目し，ブロックの中で最後に行った割合を示す関数 $fRate(a_i)$ を次のように定義する:

$$fRate(a_i) = \frac{|fBlocks(a_i)|}{|aBlocks(a_i)|} \tag{4.9}$$

ここで $fBlocks(a)$ は アクション a_i が最後に登場するブロックの集合を返す関数であり以下のように定義する:

$$\begin{aligned} &fBlocks(a) = \{b \in aBlocks(a)| \\ &\exists e_0 \in b, e_0.a = a, \forall e_k \in b\ e_0.t \geq e_k.t\} \end{aligned} \tag{4.10}$$

上記定義を用いて，Step2:代表アクション抽出ステップでは，ロール r それぞれに対して，閾値 MIN_{final} 以上の割合で，ロール内の最後のアクションとなるアクション a_i を求め代表アクションとする．したがって，代表アクションはロールごとに複数抽出され得る:

$$mainAct(r) = \{a_i|fRate(a_i) \geq MIN_{final}, a_i \in r\} \tag{4.11}$$

4.4.6 Step3:抽象化アクティビティ復元

前節で抽出した代表アクションを用いて，抽象化アクティビティを復元する．抽象化アクティビティはアクションの集合として定義する．それぞれの代表アクション a_0 を含むブロックの中に含まれるアクションのうち，同一ロールに含まれるアクションを抽象化アクティビティ act_{a_0} に加える．抽象化アクティビティ act_{a_0} は，その元となった代表アクティビティ a_0 で識別する:

$$act_{a_0} = \{e.a|b \in aBlocks(a_0), e \in b\} \tag{4.12}$$

この方法は 1 つのアクションが複数の抽象化アクティビティに属することを許すソフトクラスタリングである．図 4.6 に復元した抽象化アクティビティの例を示す．抽象化ア

組織	ロール	抽象化アクティビティ	アクション
営業部	承認依頼者	見積承認依頼	見積登録 登録エラー 見積承認依頼
	承認者	見積承認	承認依頼確認 見積承認
		見積差戻	承認依頼確認 見積差戻
在庫管理部	承認依頼者	在庫確保依頼	見積受付 在庫確保依頼
	承認者	在庫割り当て	在庫確保受付 在庫割り当て
		割り当て失敗	在庫確保受付 割り当て失敗

図 4.6　復元した業務機能階層

クティビティを構成するアクション集合は抽象化アクティビティを理解する上で重要である．このため，抽象化アクティビティに属するアクションを実際の実行履歴の断片であるブロックとすることで，抽象化アクティビティを具体的に理解しやすくなる．その時，複数のアクティビティを構成するブロックに共通して登場するアクションが存在することがある．そこで提案手法では，ハードクラスタリングではなくソフトクラスタリングを採用した．例えば，営業部の承認者ロールにおいて，「承認依頼確認」は２つの抽象化アクティビティのメンバとなっている．

4.4.7　Step4:フロー生成

最後のステップでは，復元した情報から業務フロー図を描画する．業務フロー図の縦軸に組織情報とロールを配置し，スイムレーンにロールに属するアクションを配置する．次に，それぞれのプロセスインスタンスにおけるアクションの出現順に矢印の遷移で結んでプロットする．同じ遷移が登場した場合には重ねて描画する．図 4.7 は，図 4.3 のプロセスインスタンスから復元したアクションレベルの業務フローを描画した例である．抽象度を上げる場合には，ロールの代表アクションのみを配置し，プロセスインスタンスの代表アクションだけに着目して，上記の描画を行う．図 4.8 はアクティビティレベルの業務フローである．案件 c_{00100} でのロール「承認依頼者」のユーザ「佐藤」が実施した代表アクション「見積承認依頼」と，ロール「承認者」のユーザ「鈴木」が実施した代表アクション「見積承認」とが順番に遷移で結ばれている．

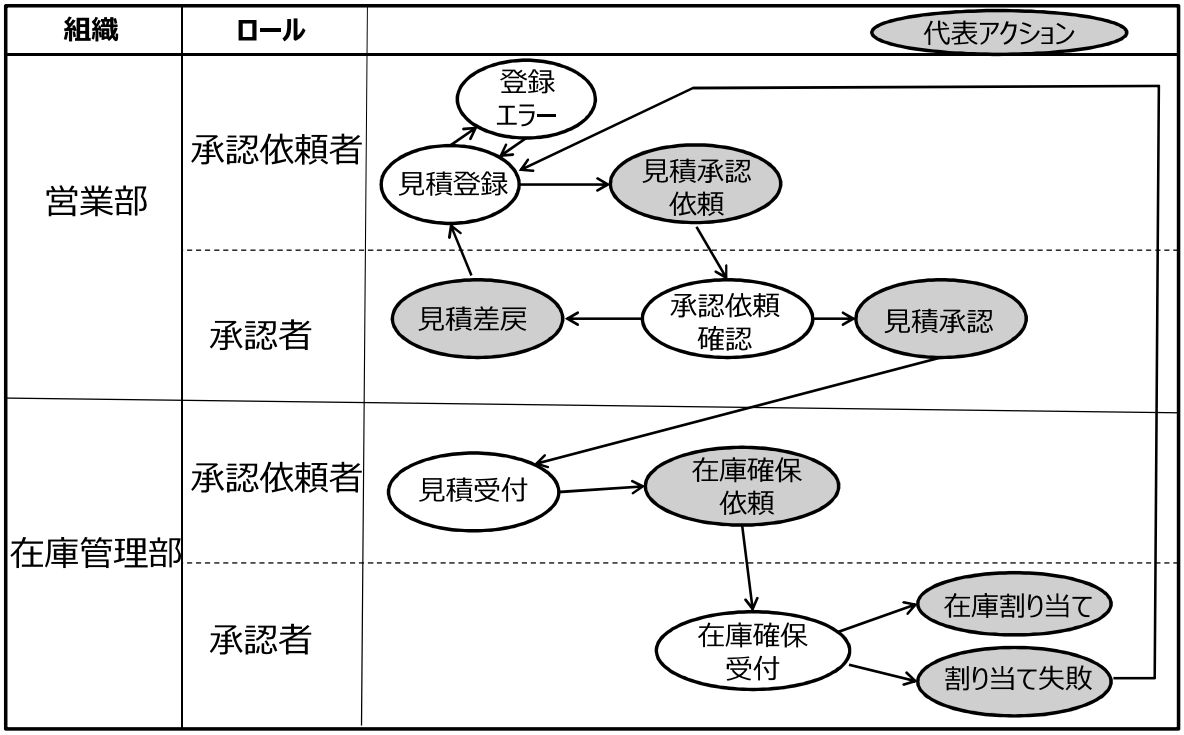


図 4.7　業務フロー (アクションレベル)

4.5　プロトタイプの実装

図 4.9 に開発したプロトタイプツールの概要を示す．ツールは 4 つのモジュール（業務機能階層復元，業務フロー描画，ログ変換，アドインプログラム) からなっている．業務フローの出力は，仕様書として利用することを考え，市販のオフィスツールの描画エンジンを利用した．それぞれのモジュールは次の特徴を持つ．業務機能階層復元は，前章で示したロールモデル復元や抽象化アクティビティ復元を行う．業務フロー描画は，前章で示した業務フロー復元のための描画データを作成する．ログ変換は，システム独自のフォーマットのログを図 4.1 で示した「標準ログ形式」に変換する．アドインプログラムは，市販のオフィスツールを GUI として用いるためのメニュー拡張，モジュールの起動，データ連携を行う．

このツールを利用する際には，あらかじめログ変換モジュールを使って，ログを「標準ログ形式」に変換しておく．そして，オフィスツールのメニューから「標準ログ形式」のログファイルを選択して実行させることで，業務フローが描画される．次章で示すケーススタディでの業務フロー図 (図 4.10, 図 4.11) は本ツールのスクリーンショットである．

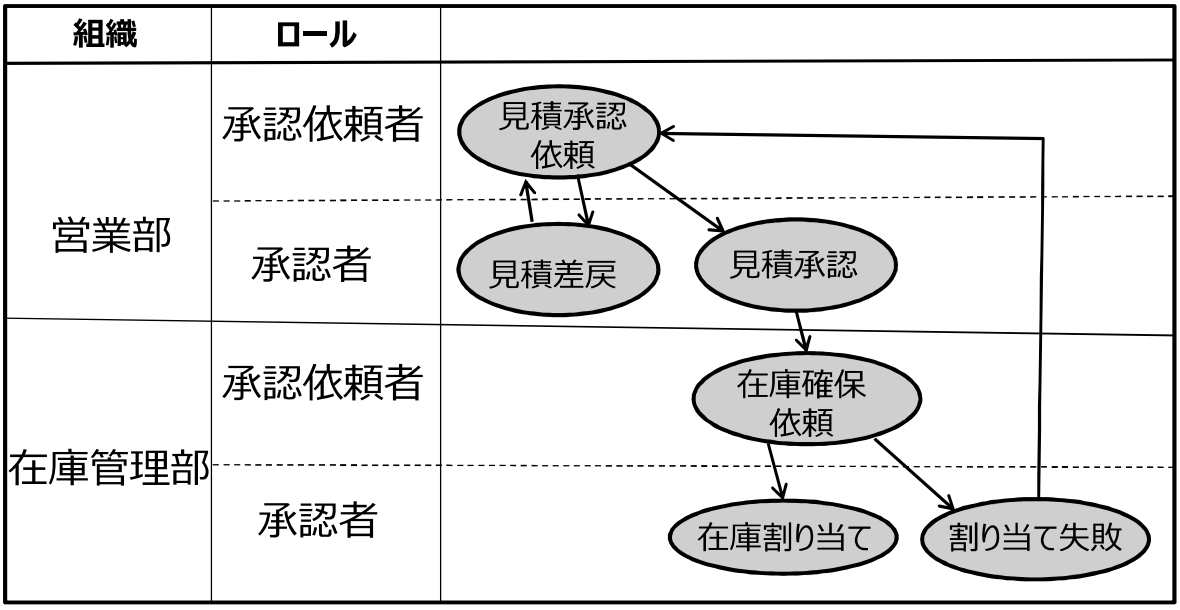


図 4.8　業務フロー (アクティビティレベル)

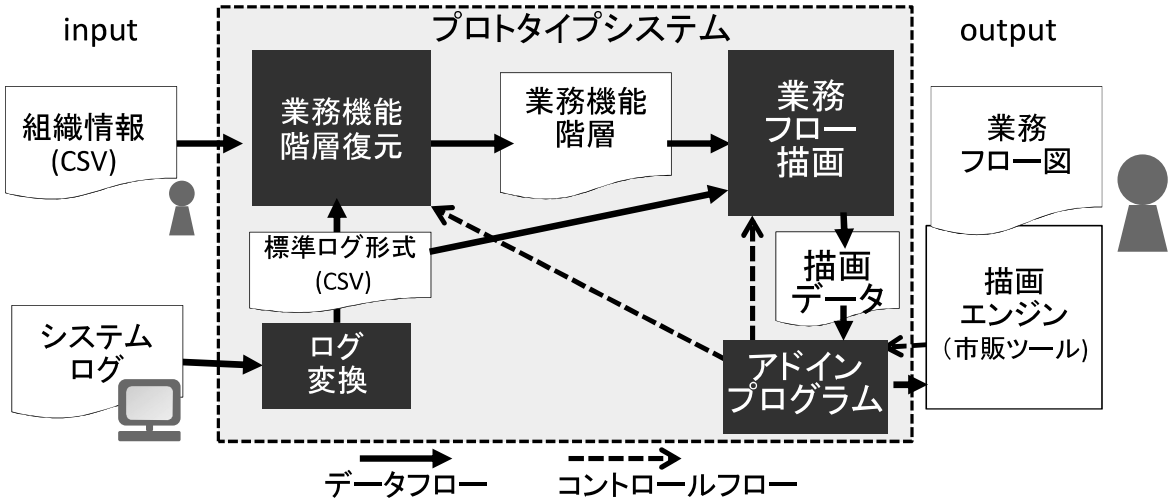


図 4.9　プロトタイプの概要

4.6　ケーススタディ

提案手法のケーススタディとして実際の業務アプリケーションに適用した．システムの実際のシステムログと，組織情報としてユーザ権限からユーザの役割の情報を入手し，前節のプロトタイプツールを利用して業務フローの復元を行った．

4.6.1 対象システム

対象システムは，出張申請・精算を行うシステムである．このシステムは，従業員が出張前の出張を申請し，これを上長が承認し，出張後の出張旅費を精算申請し承認する業務等を扱う．ユーザはこのシステムを Web ブラウザをインターフェイスとして利用する．ユーザのロールは承認依頼者（出張と旅費の精算申請を行う)，承認者（申請の承認を行う)の２つである．従業員である承認依頼者が申請した内容を上長である承認者が承認するというロールを持つ．このシステムは５つサブシステムを持ち，それぞれ５つの業務 (BP.A, BP.B, ... , BP.E) に対応する．これらの業務は，出張の申請，出張旅費の計算と精算，住所や勤務地等の基本情報の登録，という企業の事務処理で行う典型的な業務を扱うシステムである．

これは，田中の分類 [50] でいうとバックオフィス系，薦田の分類 [48] でいうと基幹系と呼ばれるカテゴリのシステムである．提案手法は，この分類に属するシステムだけでなく，チャネル系と呼ばれる顧客とのやり取りを行うシステムでも有効と考えている．しかし，チャネル系システムは顧客の個人情報を扱うことが多く，評価に使うことが難しいことから，評価の対象をバックオフィス系システムとした．

このシステムのシステムログと組織情報としてユーザ権限（ユーザとロールの関係の表)を用いてケーススタディを実施した．システムログは３ヶ月分のデータで合計 107M バイトで，410,054 のログエントリ，3,969 のユーザ, 247 のアクション, 44,604 の案件を含んでいた．パラメータとして，$MIN_{weight} = 30$ を用い, $MIN_{final} = 0.3, 0.5, 0.75, 0.9$ で比較した．

4.6.2 評価方法

提案手法の評価は，第 4.3.4 節で示した要件を満たしたことを確認するために，表 4.1 で示す観点で行った．

要件 (1)，(3) の充足を確認するために，Step1〜3 で復元した業務機能階層図に関する評価を行い，要件 (4) の充足を確認するために，Step4 で復元した業務フローの評価を行った．

上記に加え，各ステップ単位で以下の項目で評価を行った．

Step1 の評価：

ロールモデル復元ステップで，アクションのロール分割が正しく行われているか．

Step2 の評価：

代表アクション抽出ステップで，代表アクションの抽出が正しく行われているか．

Step3 の評価：

抽象化アクティビティ復元ステップで，どの程度情報が圧縮され，どの程度業務が

表 4.1　評価方法

	要件	評価項目
要件 (1)	ビジネスプロセスに登場する多数のアクションを数個からなる抽象化アクティビティにグループ化する	Step1～3 で復元した抽象化アクティビティが元のアクションからどの程度情報を圧縮できるかをもとのアクション数と抽象化したアクティビティの比（圧縮率）で評価
要件 (2)	抽象化アクティビティとアクションの関係が明確になっている	提案方式では，アクションをクラスタリングして抽象化アクティビティを抽出している．その結果を業務機能階層図で出力しているため，要件を満たすことは明らかであり評価不要
要件 (3)	復元した抽象化アクティビティによって業務全体が網羅されている	復元した抽象化アクティビティの網羅性を，全プロセスインスタンスに対する，いずれかの代表アクションを含むプロセスインスタンスの割合（再現率）で評価
要件 (4)	抽象化したアクティビティや抽象化アクティビティ間の遷移は，極力過不足がないものにする	Step4 で復元した業務フローに過不足ないかを開発者による正誤チェックの適合率と再現率で評価

網羅されているか.

Step4 の評価：

フロー生成ステップで，復元した業務フローに過不足がないか.

表 4.2 ロールモデル復元結果

BP	role	アクション	正解値	正解数	適合率	再現率
A	依頼者	47	47	47	1.00	1.00
A	承認者	10	10	10	1.00	1.00
B	依頼者	35	35	35	1.00	1.00
B	承認者	11	12	11	1.00	.917
C	依頼者	33	33	33	1.00	1.00
C	承認者	7	7	7	1.00	1.00
D	依頼者	36	36	36	1.00	1.00
D	承認者	8	8	8	1.00	1.00
E	依頼者	21	21	21	1.00	1.00
E	承認者	4	4	4	1.00	1.00
	計	212	213	212	1.00	.995

4.6.3 評価結果

Step1 の評価:アクションのロール分割が正しく行われているか

表 4.2 に Step1 で行ったアクションを分割したロールと，対象システムの開発者が作成した操作マニュアルに記載されているロールとを比較した結果を適合率と再現率で示す．それぞれの BP で承認依頼者（以後依頼者と表記)，承認者の２つのロールを復元できた．BP.B のロール. 承認者の 1 つのアクションを除き，操作マニュアルと同じ分割が行われた．正しく分割されていなかったアクションは，承認依頼の後に依頼した内容の詳細を確認するアクションであった．このアクションは他のロールと混在して使われるケースがあったため正しく分割できなかった．

Step2 の評価:代表アクションの抽出が正しく行われているか

表 4.3 に BP の中で最もアクション数が多い BP.A に対して，Step2 で抽出した代表アクション抽出の精度評価の結果を示す．パラメータ $MIN_{final} = 0.3, 0.5, 0.75, 0.9$ をそれぞれを用いて，復元した結果を評価した (終端率)．評価は，抽出した代表アクションに対して，このシステムの開発者にロールを代表する作業として適切かどうかを評価してもらった．比較のため，ロールに対応する全ブロックの中で，アクションが登場する割合 (頻度) の閾値によって抽出したアクションを合わせて評価した．

結果，終端率 (fRate) の閾値 $MIN_{final} = 0.75$ で生成したものが適合率と再現率がともに

表 4.3　代表アクション抽出結果の評価

	終端率				頻度			
	.3	.5	.75	.9	.3	.5	.75	.9
復元数	10	9	7	4	5	4	1	1
正解	7	7	7	4	3	2	1	1
適合率	.70	.78	1.0	1.0	.60	.50	1.0	1.0
再現率	1.0	1.0	1.0	.57	.43	.29	.14	.14

1.00 であり最も成績が良く，頻度の閾値で抽出したものと比較しても高い精度を得られた．抽出した 7 つの抽象化アクティビティに対して過不足がないことを対象システムの開発者が確認している．終端率が 0.75 よりも小さい場合には，本来他のアクティビティの中で途中段階でオプション的に登場するアクション「〜書出力」「〜削除」などが抽出されてしまい適合率が低下した．0.9 に設定した場合は，重要な作業まで取り除かれてしまい，再現率が下がることも確認できた．また，頻度を閾値とした場合，多くの業務で共通して用いられる「一覧表示」など重要ではない作業が代表アクションとして抽出されてしまい，正しい結果にならないものがあった．

Step3 の評価:どの程度情報が圧縮され，どの程度業務が網羅されているか.

表 4.4 に復元した抽象アクティビティによって，どの程度の情報が圧縮されたかを示す．それぞれの BP ごとに，抽象化前のアクションの数 A と抽象化後の抽象化アクティビティの数 B から，$(A-B)/A$ で圧縮率を算出している．平均して約 9 割の情報が圧縮されていることがわかる．

表 4.5 に復元した抽象化アクティビティが，元のプロセスインスタンスをどの程度カバーしているかを示す．抽象化アクティビティの代表アクションを含むプロセスインスタンスの全体に対する割合を再現率とした．表中の「依頼」はロール「承認依頼者」,「承認」はロール「承認者」を表している．取得したログには，取得期間の切れ目でプロセスインスタンスの前半，後半のログが取得できていないものや，ユーザが操作の途中で中断して別の作業を始めたものが含まれていた．これらは，プロセスとして開始から終了までがそろっていないため，プロセスインスタンスとして不完全なものになっている．このようなノイズを除去するために，1 つのロールしか含まれないプロセスインスタンスは除外して評価した．各評価の「2+r」列は，ノイズ除去を適用した場合の結果である．ノイズ除去後の再現率は，のべ約 45,000 のプロセスインスタンスに対して 0.992 との高い精度で復元できていることがわかる．

再現率が 0.8 未満の BP.B, BP.D について調査を行った．これらは海外出張関係の業務で

表 4.4　業務フロー復元圧縮率

bpid	抽象化前	抽象化後	圧縮率
A	58	7	.899
B	84	6	.929
C	43	3	.931
D	44	4	.910
E	25	3	.880
計	59	6	.899

あり，それぞれのトランザクションの期間が長いために案件のスタートがログ取得期間以前から始まっているものが含まれていた．このため，本来あるべき承認依頼プロセスがログに残っておらず，その後の承認プロセス以降がログに存在した．そして，次の同様な業務を行う際に参照するために，承認依頼者が過去の結果を確認する「明細表示」のアクションを行っていた．このアクションは前の抽象化アクティビティに属してしまい，本来属すべき次の抽象化アクティビティとして認識されていないことが原因であることがわかった．

Step4 の評価:復元した業務フローに過不足がないか

復元した業務フローが正確にシステムの仕様を反映しているかどうかを確認するために，対象システムの開発者に内容を評価してもらった．図 4.10 と図 4.11 に，提案手法による復元の例としてビジネスプロセス BP.A を復元した業務フローを示す．アクションレベルでは，57 個のアクションが複雑に関係しあうものであったが，アクティビティレベルでは 7 個の抽象化アクティビティにより表現できている． この業務フロー図を開発者に確認してもらったところ，復元した抽象化アクティビティには過不足はないという評価を得た．他の４つ（ＢＰ.B～BP.E）のビジネスプロセスの復元結果を付録として第 4.9 節に示す．さらに，業務フロー図上で復元した抽象化アクティビティ間の遷移に関して，仕様として正しいことを評価してもらった．対象システムをもっとも理解している開発者に，復元した業務フロー図を提示し，生成した抽象化アクティビティのリストと遷移リストに○×をつけるワークシートを記入してもらった．表 4.6 に開発者による評価結果を示す．評価の結果，84% の遷移が正しいという結果となった．不正解となった遷移は，「却下」した後「引き戻し」て「承認」するなど，終端率でフィルタリングした処理 (この場合は「引き戻し」) を挟んだ時に本来あるべき中間の処理が取り除かれたために生じていた．

表 4.5　プロセスインスタンスカバー率

	元データ		抽象化		再現率	
bp role	all	2+r	all	2+r	all	2+r
A 依頼	9,836	811	909	755	.092	.930
A 承認	906	801	867	786	.956	.981
B 依頼	2,507	535	509	403	.203	.753
B 承認	576	486	529	466	.918	.958
C 依頼	20,152	19,630	19,992	19,613	.992	.999
C 承認	28,900	19,707	28,897	19,707	.999	1.00
D 依頼	471	411	302	300	.641	.729
D 承認	540	421	540	421	1.00	1.00
E 依頼	1,620	1458	1,560	1,450	.962	.994
E 承認	2,448	1,458	2,447	1,457	.999	.999
のべ	67,956	45,718	56,552	45,358	.832	.992
合計	44,604	22,427	33,555	22,416	.752	.999
$\|E\|$	410,054	348,204	66,055	54,098		
$\|A\|$	247	245	22	22		

all: 全プロセスインスタンス

2+r: 2 以上のロールを持つプロセスインスタンス

表 4.6　抽象化アクティビティの評価

BPid	正解の遷移	不正解の遷移	適合率	再現率
A	15	0	1.00	1.00
B	15	2	.88	1.00
C	6	3	.67	1.00
D	7	2	.78	1.00
E	6	2	.75	1.00
計	49	9	.84	1.00

4.7　考察

4.7.1　業務フロー復元の精度について

ケーススタディによる評価により，以下のことがわかった．Step1 のロール分割の精度が 99.5% とほぼ復元できていると考える．

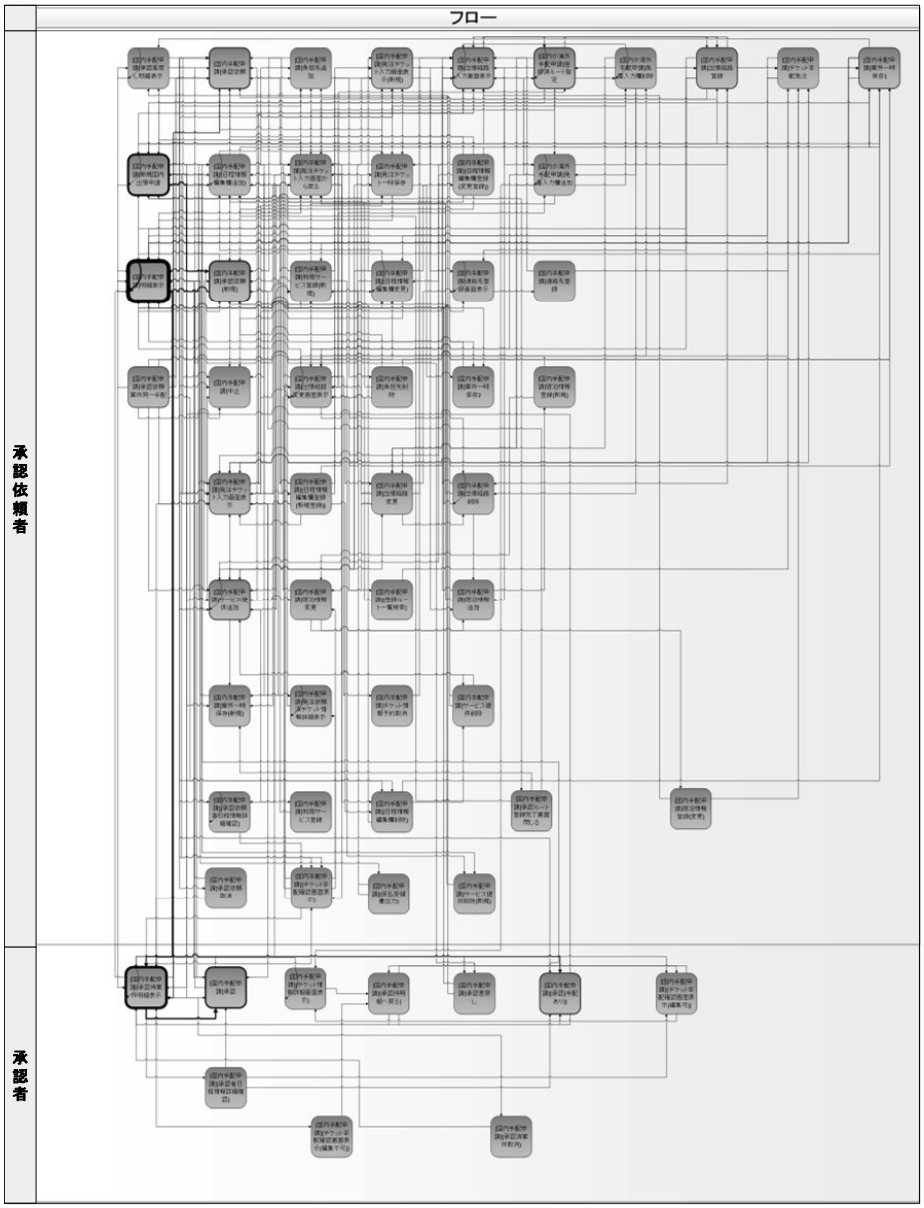


図 4.10　業務フロー (抽象化前)

Step2 の代表アクション抽出については，終端率 0.75 とした時が最も精度が高く，単なるブロックへの出現率と比較してもこの手法による代表アクション抽出は有効である．

Step3 の評価によって，元のログの情報からアクション数にして 89.9% の情報を圧縮でき，その圧縮した情報には全体の 99.2% のプロセスインスタンスの情報が含まれていることがわかり，情報の圧縮が効率的に行われているといえる．

Step4 の評価で，復元した業務フローが開発者の視点でみて，抽象化アクティビティのレベルでは問題なく，アクティビティ間の遷移で見ても 84% の適合率で復元できていることがわかり，全体として提案手法による業務フロー復元は有効であることがいえる．

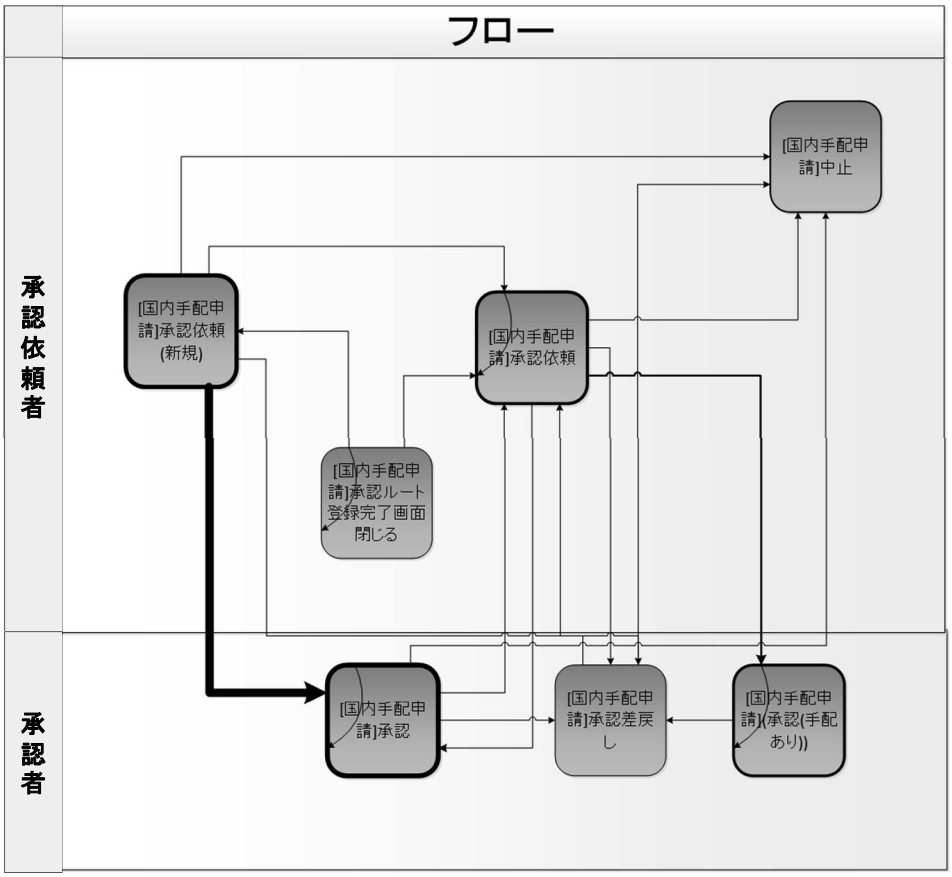


図 4.11　業務フロー (抽象化後)

4.3.4 節で示した要件に対しての考察を以下で述べる．

要件 (1)　表 4.4 でそれぞれの BP の 25〜84 あるアクションを 3〜7 の抽象化アクティビティに抽象化することができた．これによって，抽象化した業務フローを用いて，マクロ的に全体像を俯瞰することが可能になる．

要件 (2)　抽象化アクティビティとアクションの関係を業務機能階層で抽出することができた．これにより，抽象化アクティビティと元のアクションの対応をとることが可能になる．さらに，抽象化した業務フローで全体を俯瞰した後に，詳細レベルのフローを業務機能階層を手掛かりに理解することで，段階的に詳細レベルまでの理解が可能になる．

要件 (3)　ソフトクラスタリングを用いたことにより，それぞれの抽象化アクティビティに対応する意味のあるアクション群を復元でき，表 4.5 により全体の 99.2% のプロセスインスタンスの情報が含まれていた．これにより，抽象化アクティビティに関係するアクション群は，実際にプロセスインスタンスの断片として業務全体を網羅していることがいえる．

要件 (4) 開発者の評価により，抽象化アクティビティレベルでは適合率，再現率ともに 1.0，遷移レベルでは，適合率 0.84，再現率 1.0 となった．再現率を 1.0 に近づけるために，ソフトクラスタリングを採用したため，一定数の不要な遷移が描画されている．しかし，当初の目的である「大規模なシステムの仕様をマクロ的な視点から俯瞰して理解する」という意味では，遷移の情報が不足しているよりも，余分な情報を詳細レベルのフローで確認して除去するという方法で補完することができる．このため，要件は満たせたといえる．ハードクラスタリングを用いることで適合率は改善可能である．しかし，これにより，要件 (3) を満たせなくなってしまう．この影響は大きいので，本研究の目的には合致しない．

4.7.2 残された課題

前節の考察により，提案手法が有効であることを示したが，さらなる効率化のために以下の課題が残されている．

再現率優先の要件 (4) を満たすために，ソフトクラスタリングを採用したことにより遷移の適合率が 0.84 となり，16% の誤った遷移が復元されている．このため，ユーザと開発者が全体像を誤った形で理解してしまう恐れがある．この課題を解決するために，遷移の頻度やロールの切れ目になるブロックにまたがる遷移をブロックの長さによってフィルタリングをすることが考えられる．これにより，ハードクラスタリングによって満たせなくなる要件 (3) を満たしたまま，この適合率を向上できる可能性がある．

今回 0.75 とした終端率の閾値 MIN_{final} の設定は，今回複数パタンで比較して良い結果を選んだが，実用上は何らかの方法で決める方法を検討する必要がある．これは，ログの特性を前もってスクリーニングすることによって，適切な閾値を算出できる可能性がある．

プロセスインスタンスのカバー率を低下させる原因となっている，ログ取得期間と案件のライフサイクルがずれる問題に対して考慮する必要がある．これは，事前に途中開始や途中終了の案件をフィルタリングして分析するなどの前処理を行うことによって，プロセスインスタンスとしてふさわしくないデータをノイズとして除去し，より高いカバー率を得ることができる可能性がある．

「明細表示」のような複数のプロセスインスタンスにまたがる業務 (過去のケースを参照して，別のケースを承認するなど) を扱うことができるようにプロセスインスタンスの考え方の拡張を行うことで，再現率を向上できる可能性がある．

4.8　本章のまとめ

正確な仕様が残されていない既存の業務システムを理解するために，システムログと組織情報から業務フローを復元する手法を提案した．提案手法は，複数のユーザによって行われる業務に対し，ユーザの切れ目に着目することによって，ロールや代表的なアクションを抽出し，抽象化されたアクティビティを復元することができる．

ケーススタディによる評価では，ロール復元精度が 99.5%, 抽象化アクティビティの 再現率が 99.2% であり，本手法の有効性を示すことができた．

今後，より正確な業務フロー復元を行うためには，4.7 節で述べた課題を解決する必要がある．その方向性を以下に示す．

1. ログの特性を前もってスクリーニングすることによって，適切な閾値を算出できる可能性がある．
2. 事前に途中開始や途中終了の案件をフィルタリングして分析するなどの前処理を行うことによって，プロセスインスタンスとしてふさわしくないデータをノイズとして除去しより高いカバー率を得ることができる可能性がある．
3. 複数のプロセスインスタンスにまたがる業務（過去のケースを参照して，別のケースを承認するなど）を扱うことができるようにプロセスインスタンスの考え方の拡張を行うことで，再現率をあげられる可能性がある．
4. 再現率優先の要件により，抽象化アクティビティの遷移の適合率は 0.84 となっている．遷移の頻度やブロックのまたがり状況によるフィルタリングをすることにより，この適合率が向上し，さらに仕様理解の負担をさらに減少させられる可能性がある．

4.9　付録：他の BP の復元結果

図 4.12, 図 4.13, 図 4.14, 図 4.15 に本文中では省略した BP.B～BP.E の復元した業務フローを掲載する.

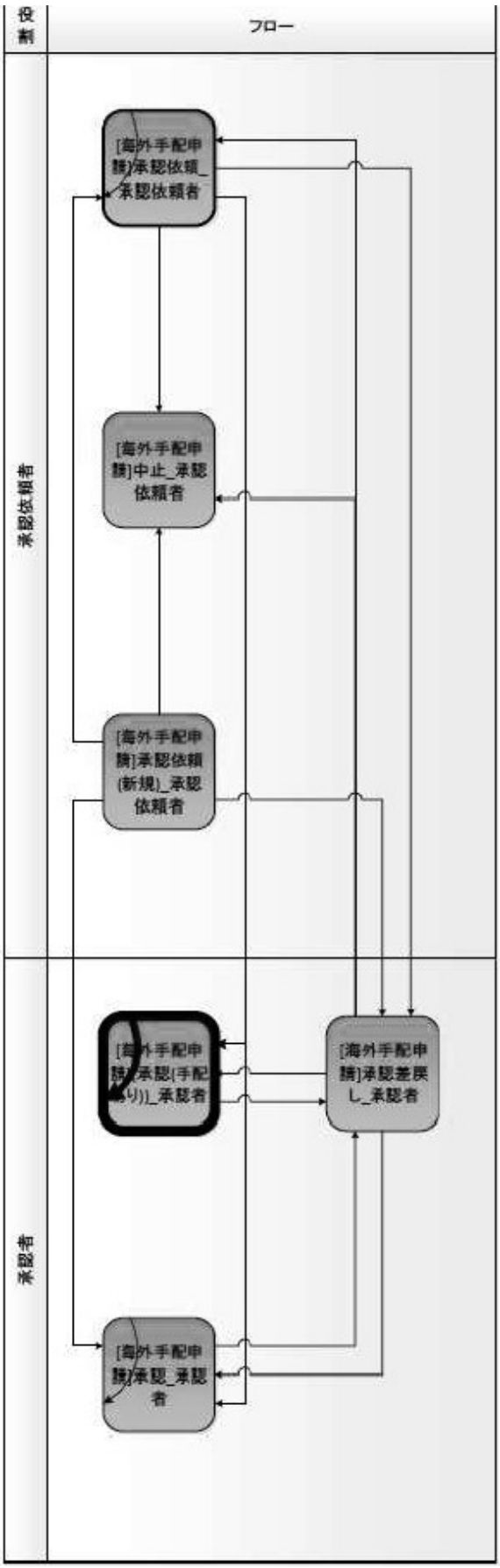


図 4.12　BP.B の復元した業務フロー

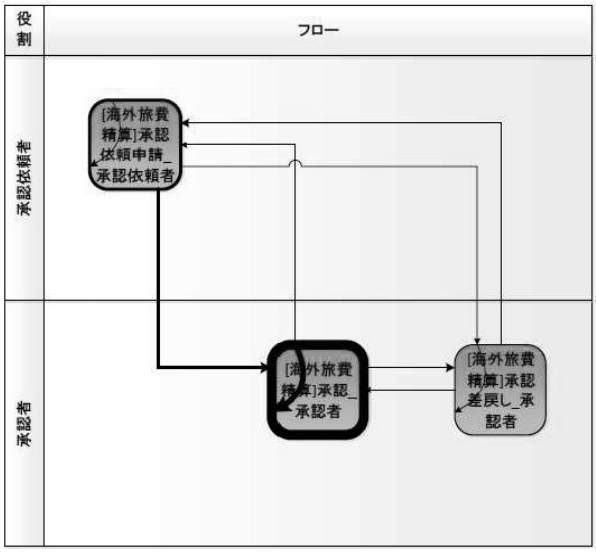


図 4.13　BP.C の復元した業務フロー

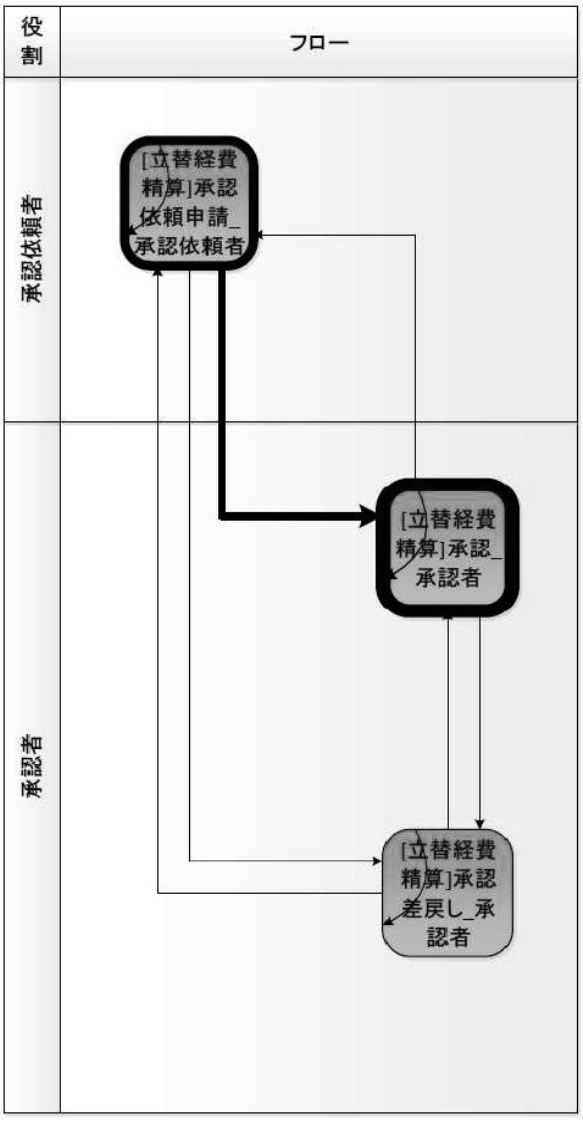


図 4.14　BP.D の復元した業務フロー

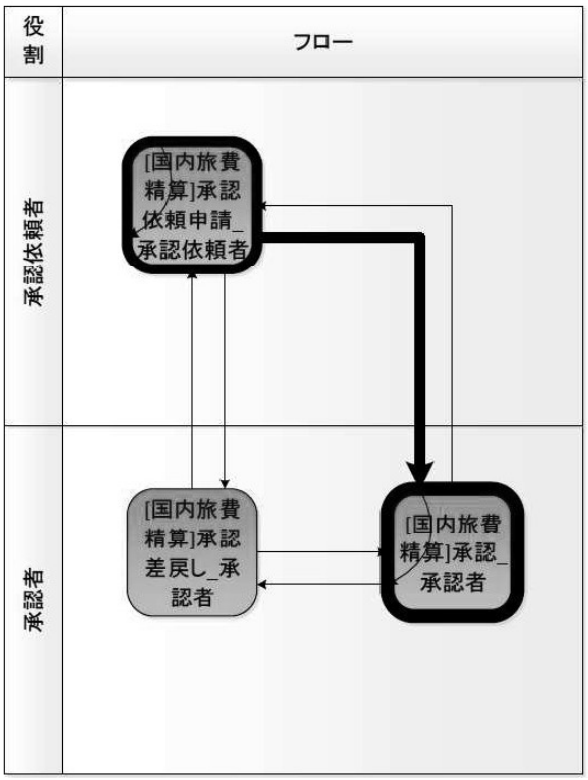


図 4.15　BP.E の復元した業務フロー

第 5 章

Web アプリケーションのユースケース駆動プロトタイプによる要求獲得手法

5.1 要求獲得における課題

ソフトウェア開発では，仕様変更への対応工数は開発の終了に近付くほど高くなることから，開発の最上流である要求分析工程で仕様変更のリスクを抑えることが開発成功の鍵となる [14, 33]．特に，Web アプリケーション開発に代表される小規模・短期間プロジェクトでは，「質の高い」要求を「手早く」収集し，新仕様として定義する手法が必要とされている．ユーザからの要求の妥当性を確認する際には，インタビューや会議で，次の 3 つのものをユーザに提示することがよく行われている．

1. 自然言語と非形式的な図からなるドキュメント
2. UML などのモデリング言語で記述されたモデル
3. プロトタイプ

ドキュメントによる要求獲得は，最も広く用いられている手法である．この手法は，自然言語と図で構成されるドキュメントを用いてユーザの要求を確認するため，自由度が高く，どのような仕様も表現できるという利点がある．その反面，どうしても表現のあいまい性が残ってしまうという課題がある．これに対して，人による推敲作業を確実に行い，あいまい性を改善するための推敲方法 [33][51] が提案されている．この方法では，推敲作業を効率化し，見つけにくい不具合を発見することが可能になるとされている．しかし，自然言語を扱う以上は，本質的なあいまい性を完全に排除することは難しい．そのため，あいまい性を排除する試みとして，自然言語に制約を加えた要求仕様記述言語を用いて要

求を記述する方法も提案されている [49].

UML などのモデリング言語を用いて，要求を記述する方法も多く提案されている [14]. 特に要求分析の段階では，ユースケース図，ユースケースシナリオを用いることが多い [5]．ユースケース図は，アプリケーションが提供する機能とアプリケーションの外部要素との相互作用を表現する．また，ユースケースシナリオは，ユーザが理解することができる自然言語で，個々のユースケースにおけるユーザとアプリケーションの振舞いの流れを記述する．ユースケースシナリオは，アプリケーション構築の知識を持たないユーザが受け入れやすい反面，自然言語で記述されるため，前述のドキュメントと同様にあいまい性，一貫性に課題が残る [55]．この課題を解決するために，システム化範囲を意識せずに対象とする世界をモデル化するドメインモデルが用いられる．このドメインモデルを構築する際に，ユーザの動作のシナリオをパターン化し，パターンを組み合わせるアプローチが提案されている [25][40]．これらのアプローチにより，ドメイン分析段階でのモデルのあいまい性は軽減されるが，これをシステム化する際には，データモデルと機能の関連付けを行う設計段階でのあいまい性の課題が依然残っている．

また，他の UML の図を用いて，ユースケースシナリオ相当の内容を記述する試みもされている．しかし，クラス図やアクティビティ図などの UML の図は，ソフトウェアの専門家ではないユーザにとっては，なじみのあるものではない．そのため，理解に時間がかかるとともに，具体的なシステムのイメージを喚起することが難しく，結果的にユーザが満足する要求を引き出すことを阻害してしまう可能性が高い．

ユーザにとってシステムの完成イメージを理解し，要求を引き出すために最も有効といわれる方法として，プロトタイピングがある [39][31]．プロトタイピングは，要求仕様のあいまいな部分をあらかじめ簡易的に作ることで，開発におけるリスクを軽減する方法である．プロトタイピングにより，ユーザはアプリケーションの具体的なイメージを得ることができるが，ユースケースに比べ，プロトタイプの作成と保守には，コストがかかるため，実際のプロジェクトでは適用に踏み切れない場合がある．また，プロトタイプで実物を見せてしまうことによって，仕様確認時にユーザが目につきやすい，画面の色や配置などの細かい部分の議論に時間がとられてしまい，要求の本質である業務の流れや業務データの検証を効率的に行うことができないといった弊害も懸念される．プロトタイプのメンテナンス性や本質的要求の検証の問題を解決する方法には，紙に書いたラフスケッチを基に要求を獲得するペーパプロトタイピングの方法がある [30]．本章で示す方法がめざすところは，ペーパプロトタイピング相当の効果を，パターンを用いて生成したプロトタイプによって実現し，その結果から後工程で活用できる仕様を作るところにある．

本章では，先に述べた，設計段階でのドキュメントのあいまい性の問題，モデリング言語のイメージ喚起不足の問題，プロトタイピングのコストの問題，本質的議論への誘導の問題をそれぞれ解決するために，下記の特徴を持つユースケース駆動プロトタイプ環境を

提案する（図 5.1）.

まず，典型的なユースケースシナリオをパターン化（シナリオパターン）することで定型化（Scenario Patterns）する.

step1 シナリオパターンを組み合わせてシナリオを作成.
(Select scenario patterns to design a usecase scenario)

step2 シナリオで使用するデータ項目（Design Parameters）を定義.
(Setup design parameters to design data scheme)

step3 シナリオパターンごとに定義されている典型的な画面のパターン（GUI patterns）を選択.
(Select GUI Patterns to design page transition)

step4 step1〜step3 の結果を用いてプロトタイプを生成.
(Generate a prototype)

上記 step1〜step4 のステップをユーザに確認しながら繰り返す.

ユースケースシナリオに基づいてプロトタイプを生成することにより，修正を繰り返してもプロトタイプとユースケースの一貫性を保つことが可能である．また，シナリオパターンと，画面パターンを独立に設けたことにより，ユースケースシナリオで業務の流れを確認し，その後に，画面の構成を独立して決定することが可能となる.

以下第 5.2 節では，関連研究を，第 5.3 節ではシナリオパターン，画面パターンの構造を用いたユースケース駆動のプロトタイピング手法の基本コンセプトを，第 5.4 節では支援環境のアーキテクチャを，第 5.5 節では，本環境をプロジェクト管理システムの開発に適用した実験を，第 5.6 節では，実験の結果を，第 5.8 節では全体のまとめと今後の課題をそれぞれ説明する.

5.2 プロトタイプ生成に関する関連研究

ユースケースシナリオなどのモデルからプロトタイプを生成する関連研究は以下がある．Somé の研究 [32] や Elkoutbi らの研究 [8] では，操作レベルで定義されたシナリオとクラス図からシステムの内部状態を表すステートチャートを生成することで，プロトタイプを生成する．また，小形らの研究 [21, 53] では，クラス図でデータを定義し，アクティビティ図を用いて，ユーザとシステムのやり取りを定義することでプロトタイプを生成する．これらの手法では，ステートチャートやアクティビティ図で詳細仕様を自由度高く表現しプロトタイプに反映させることは可能であるが，仕様を事細かに定義する必要があり，定義に多くのコストと専門知識が必要となる．このため，システムの専門知識のない顧客との合意形成には適さない．一方，白銀らの研究 [59] は，ユーザの操作ステップ（入力，ク

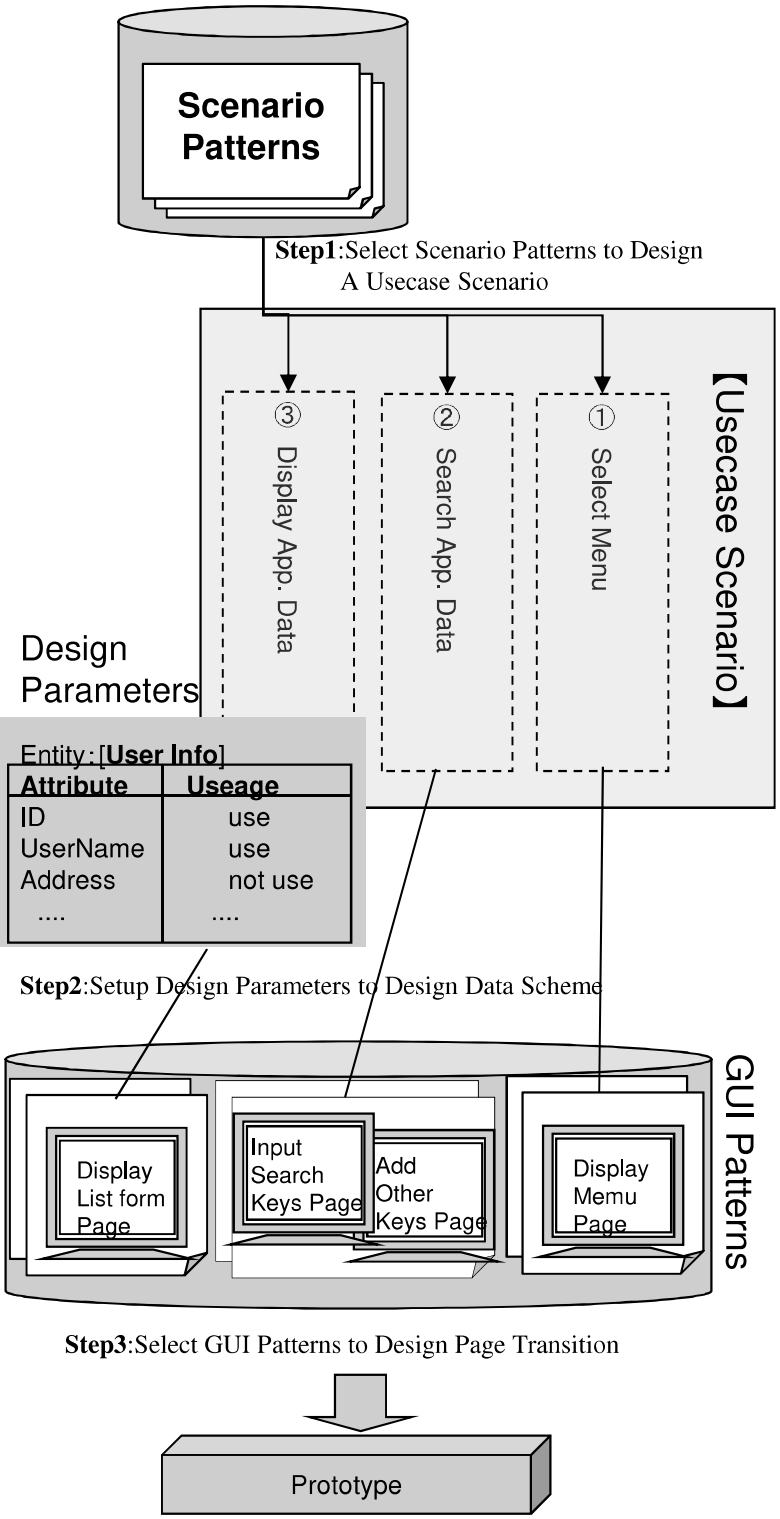


図 5.1　ユースケース駆動プロトタイプ環境

リックなど）の単位で記述された複数のシナリオを縒り合わせたグラフを生成し，そのグラフに対応して，入出力情報や画面構成情報を定義することで GUI プロトタイプを生成する. 顧客との合意形成が容易なシナリオからプロトタイプを得ることができるが，システム再構築に必要な仕様であるデータ仕様を扱えていない．Saito らの研究 [27, 28] では，5 つの Requirement Models(Business Process, Business Rule, User Inteface, User Authentication, Access Authorization) をそれぞれ定義することでプロトタイプを生成する．この中でユースケースシナリオに相当するものは Business Process である．Business Process で定義するアクティビティの最小単位は Function Type として Create, Seach, Output, Select などデータ操作の単位で与えられる．この手法により Business Process から段階的にプロトタイプを示して仕様を決めていくことができるが，多くの仕様が定型化されていて，既存システムの仕様を踏襲するエンタープライズシステムで適用する際には，定義内容が重複する懸念がある．

本章で示す手法はこれらのアプローチとは異なり，典型的なシナリオのパターンを組み合わせることでシナリオを作成する．シナリオを作成する際に，操作の単位ではなく複数の操作が組となった動作の単位で考えることができ，シナリオだけでなく，入出力情報や画面構成に関してもあらかじめバリエーションを用意しておくことで，定義者の作業量を減らし，ユーザに対するプロトタイプ提示回数を増やすことができる．

5.3 ユースケース駆動プロトタイプ環境

5.3.1 前提となる開発プロセス

本章で提案する環境の前提となる開発プロセス（プロトタイプ利用型開発プロセス）を図 5.2 に示す．本開発プロセスは，開発を要求獲得フェーズ（Requirements Elicitation），設計フェーズ（Design），実装・テストフェーズ (Implementation And Test）の 3 つのフェーズからなる．

要求獲得フェーズ (Requirement Elicitation) では，開発者は図 5.1 で示したユースケース駆動プロトタイプ環境を用いて，プロトタイプを作成する．ユーズケース駆動プロトタイプ環境を用いて，開発者はまず，シナリオパターン（Scenario Patterns) を割り当てることでユースケースシナリオを定義する (Define Usecase Scenario). 次に，デザインパラメータを設定し (Setup Degin Parameter)，画面パターン (GUI Patterns) を割り当てることでシナリオの詳細を定義する．このシナリオからプロトタイプを生成する．生成したプロトタイプを開発者がユーザに提示し，仕様確認することで (Review Prototype)，ユーザからの指摘事項として新たなユーザ要求を獲得する．これを繰り返すことにより，システムの完成イ

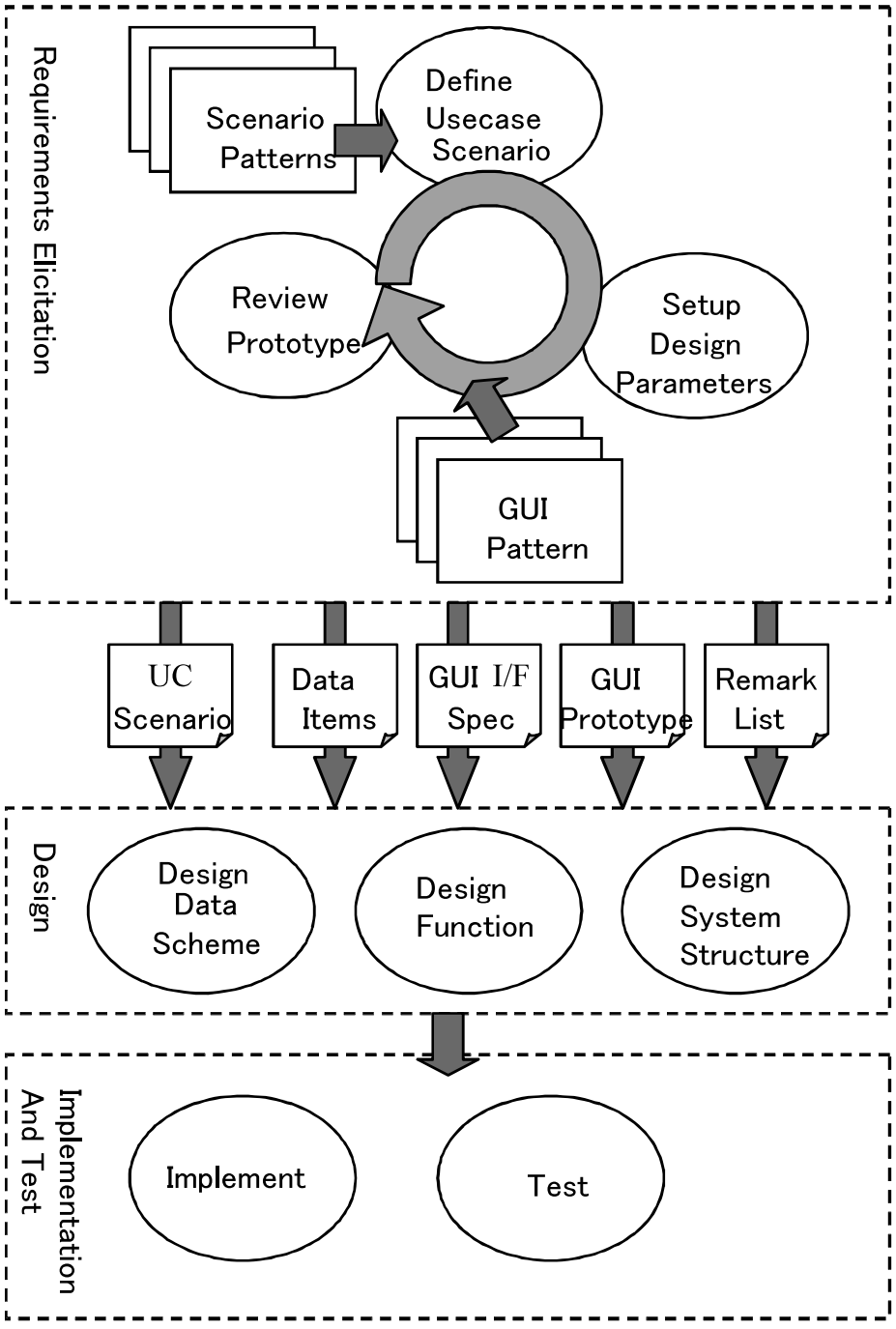


図 5.2　プロトタイプ利用型開発プロセス

メージを固めていく．

設計フェーズ (Design) では，要求獲得フェーズで得られた外部的な要求に対して，開発者が効率性を考慮した内部設計を行う．具体的には，データベース設計 (Design Data Scheme)，機能設計 (Design Function)，システム構造設計 (Design System structure) を行う．例えば，要求獲得フェーズで各ユースケースシナリオでやりとりされるデータ項目のビューを得て，これをデータベースのテーブルとするために，正規化，最適化，インデックス定義を行い，データベーススキーマを定義する．また，要求獲得フェーズで得られた機能のうち，共通して開発できる部分を切り出してコンポーネント化することもこのフェーズで行う．

実装・テストフェーズ (Implementation and test) では，開発者が設計フェーズで設計した仕様に基づいてソフトウェアを実装 (Implement) し，テスト (Test) を行う．

本章で提案する手法は，上記プロセスでの，要求獲得フェーズで開発者がユーザの要求を漏れなく，正確に獲得することによって，以降で開発者が行う設計フェーズや実装・テストフェーズでの手戻りを削減することを目指す．

5.3.2 要求獲得フェーズの要件

第 5.3 項で述べた目的のためには，要求獲得フェーズにおいて以下の要件を達成する必要がある．

要件 (1) 成果物のあいまい性を除去し一貫性を保てる．

要求獲得でユーザとのコミュニケーションに用いる成果物（ユースケースシナリオ，プロトタイプ）に対して，設計者のスキル，癖への依存性，成果物間の矛盾を排除する．

要件 (2) ユーザのイメージを喚起できる情報を提示して仕様確認できる．

仕様確認で提示するユースケースシナリオとプロトタイプを組み合わせることによって，ユーザに対してシステムの完成形を具体的にイメージさせる必要がある．

要件 (3) 本質的な仕様を中心に仕様確認できる．

具体的なイメージを与えると，ユーザは本質的な仕様に関係ない細かいレベルの要求（画面の色や配置など）を提示してくる可能性がある．これらは下流工程でも容易に修正が可能なので，修正が困難なデータ項目や機能を中心に仕様確認を進められるようにする．

要件 (4) 仕様確認に必要な情報の作成を効率化できる．

プロトタイプを何回も見せることにより，ユーザの要求を繰り返し確認することができる．このため，プロトタイプの作成を効率的に行うことが必須となる．具体的

には，4〜5 程度のユースケースシナリオ，プロトタイプも含めた成果物を 1 週間で作成し，週 1 回の仕様確認でユーザに確認できることを要件とした．ここでのユースケースの粒度は，1 人のユーザがシステムを通して 1 度に行う操作を 1 ユースケースとし，複数の画面で実現されてるものとする．

5.3.3　パターンによる定型化

第 5.3.2 項の要件を達成するために，図 5.1 で示したシナリオパターン，画面パターンを用意しておく．シナリオパターンは，ユースケースシナリオの断片をパターン化して蓄積しておくために，エンタープライズシステムのユースケースに現れる典型的なシステムとユーザのインタラクションをステップ単位で表現したものである．また，シナリオパターンのそれぞれのステップは，操作対象となるデータのエンティティをパラメータとして切り出しておく．図 5.3 にシナリオパターンの例を示す．図中の『メニューの表示』,『業務データの検索』,『業務データの選択』がそれぞれシナリオパターンである．多くのエンタープライズシステムのユースケースに登場するこれらのインタラクションをシナリオパターンとして切り出している．また，シナリオパターン『メニューの表示』における [対象メニュー] や，シナリオパターン『業務データの検索』における [エンティティ 1]，シナリオパターン『業務データの選択』における [エンティティ 2] は，シナリオパターンのパラメータであり，対象とするデータエンティティを差し替えられるようになっている．これらのシナリオパターンを開発者が組合わせて，1 つの「登録されたユーザ情報を更新する」というユースケースシナリオを構成する．

シナリオパターンには，業務仕様のバリエーションを吸収するためのパラメータを用意しておく．このパラメータを設計パラメータと呼ぶ．設計パラメータには，そのシナリオで扱うエンティティと，シナリオを遂行する上でアクセスするエンティティの属性，型を設定する．図 5.3 の 3 番目のシナリオパターン『業務データの選択』の設計パラメータである「エンティティ 2」に「顧客」を設定し，それぞれのステップで入出力する属性の出力を「O」，入力を「I」として設定している例を図 5.4 に示す．図では，ステップ 7 のパラメータとして，顧客エンティティの住所，名前，ID のフィールドに「O」を設定している．これで，このステップで顧客に関するこの３つのフィールドを一覧で出力するシナリオを定義している．次のステップ 8 のパラメータとして，顧客エンティティの ID のフィールドに「I」を設定している．これで，このステップでユーザが顧客を選択した結果を ID で入力するシナリオを定義している．ステップ 9 は，イベントを発生する処理だけを行うため，エンティティに対しての入出力は発生しないため，設定しない．

ユースケースの一部を切り出して，複数の箇所で再利用し，ベースのユースケースを拡

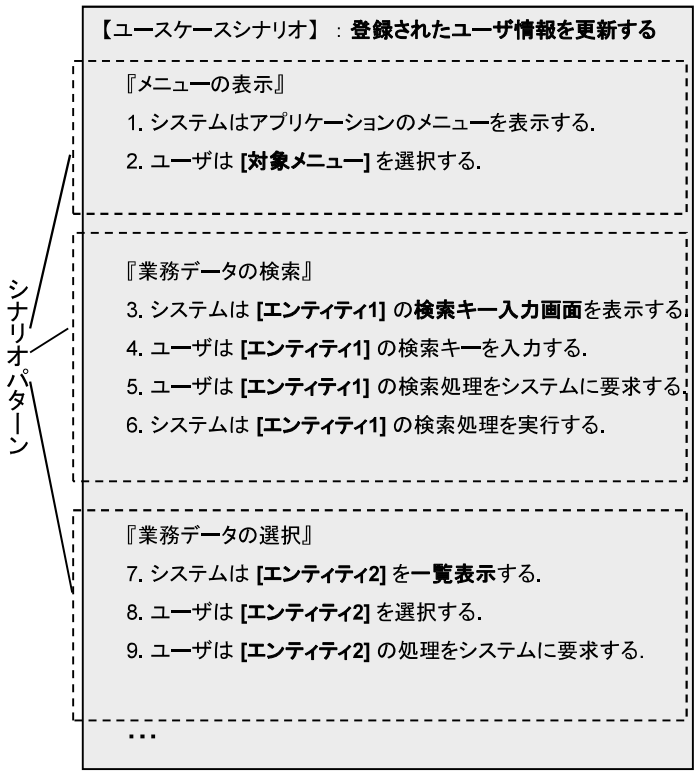


図 5.3　シナリオパターンの例

	エンティティ						
	顧客			商品			
シナリオパタン『業務データの選択』の扱うエンティティ	2						
ステップと使用する属性	住所	名前	ID	単価	名称	品番	
7. システムは [エンティティ2] を一覧表示する.	○	○	○				
8. ユーザは [エンティティ2] を選択する.			I				
9. ユーザは [エンティティ2] の処理をシステムに要求する.							

図 5.4　設計パラメータの設定

張する仕組みには << *extend* >> や << *include* >> がある [11]．これらは，ユースケースの単位で，共通部分を括りだしたり，拡張部分のみを記述したりすることは可能であるが，本章のシナリオパターンで実現するシナリオの中身のパラメータ化を対象としたものではない．

画面パターンには，シナリオパターンを実現する一連の画面のレイアウト情報をパターン化して蓄積しておく．1 つのシナリオは複数の画面で実現される．その複数の画面の構成を 1 つの画面パターンとして扱う．1 つのシナリオパターンを実現する画面の実装方法は複数存在する場合には，それぞれに対応する画面パターンを作成する．例えば，図 5.3

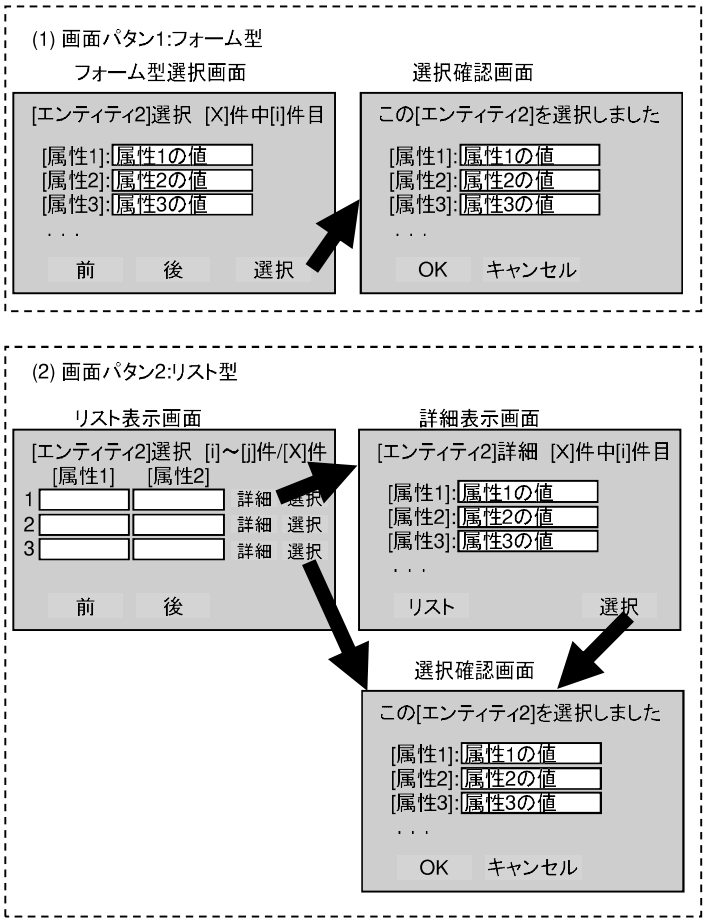


図 5.5　画面パターンの例

の『業務データの選択』シナリオパターンに対して次の 2 つの画面パターンが考えられる (図 5.5)．1 つは，まず情報をフォーム形式で表示（フォーム型選択画面) し，画面を切り替えることで対象とする情報を選択し，最後に選択結果を確認（選択確認画面）する画面パターン 1「フォーム型」である．もう 1 つは，まず選択肢をリスト形式で表示（リスト表示画面）し，リストの中から選択した情報の詳細を確認（詳細表示画面）し，最後に選択結果を確認（選択確認画面）する画面パターン 2「リスト型」である．前者はスマートフォンのように限られた画面に適していて，後者は PC 画面のような大きな画面での操作に適している．

シナリオパターンと画面パターンを独立して用意することにより，ユースケースシナリオで業務の流れを確認し，その後に，画面の構成を独立して決定することが可能となる．よって，開発者とユーザが合意をとるべきフォーカスポイントを絞って仕様確認を行うことができるため，第 5.3.2 項で示した提案手法への要件 (2)「ユーザのイメージを喚起できる情報を提示して仕様確認できる」を満たすことができる．

要求された仕様が用意されたパターンにあてはまらない場合には，対応するパターンを

追加する．ユーザの要求を単純にモデル化する場合に比べ，パターン追加という作業を明示化することで，この要求実現にともなうコスト増およびリスクの有無を要求分析の段階でユーザと開発者が認識することができる．このため，本質でない要求による無意味なコストやリスクを抑制する効果も期待できる．これは第 5.3.2 項で示した提案手法への要件 (3)「本質的な仕様を中心に仕様確認できる」に対応する．

ここで，開発者があらかじめ用意されたパターンを利用することで，ユーザの要求を限定してしまうのではないか議論がある．本来のユーザの要求を，選択肢の中から選ぶことでゆがめてしまうのではないかという懸念である．しかし，実現できない要求を定義したとしても，意味がない．このため，パターンが実現可能な十分なバリエーションを持っていることで，その懸念は軽減されると考える．一方，ユーザの持つ漠然とした要求は，ユーザ自らが明示することは困難であり，具体的な仕様を見ることで，初めてそれが要求を満たしているかどうか判断できるという面もある．本論文では，要求をゆがめる危険性を回避するために，仕様そのものではなく，仕様を実現したプロトタイプを用いてユーザがシステムを用いて行いたい目的が達成できるかという視点で仕様確認を行うことで，ユーザの要求が獲得できるという立場をとる．

5.4 支援環境

5.4.1 支援環境の構成

第 5.3 節で示した手法を支援するための開発支援環境を開発した (図 5.6)．開発支援環境は次の 3 つのコンポーネントからなる．

1. パターンライブラリ
   ユースケース定義ツールやプロトタイプ生成エンジンで用いるシナリオパターン，画面パターン，画面パターンに対応するプログラム部品である画面部品をあらかじめ用意しておくライブラリである．それぞれのパターンおよび部品の持つ依存関係もパターンや部品の属性としてパターンライブラリで保持する．
2. ユースケース定義ツール
   シナリオパターン，画面パターン，設計パラメータを用いてユーザがユースケースシナリオを作成するためのエディタを提供する．
3. プロトタイプ生成エンジン
   ユースケース定義ツールで定義したユースケースシナリオと設計パラメータから，画面部品を組み合わせて，プロトタイプを生成する．

以下で，コンポーネントの詳細を述べる．

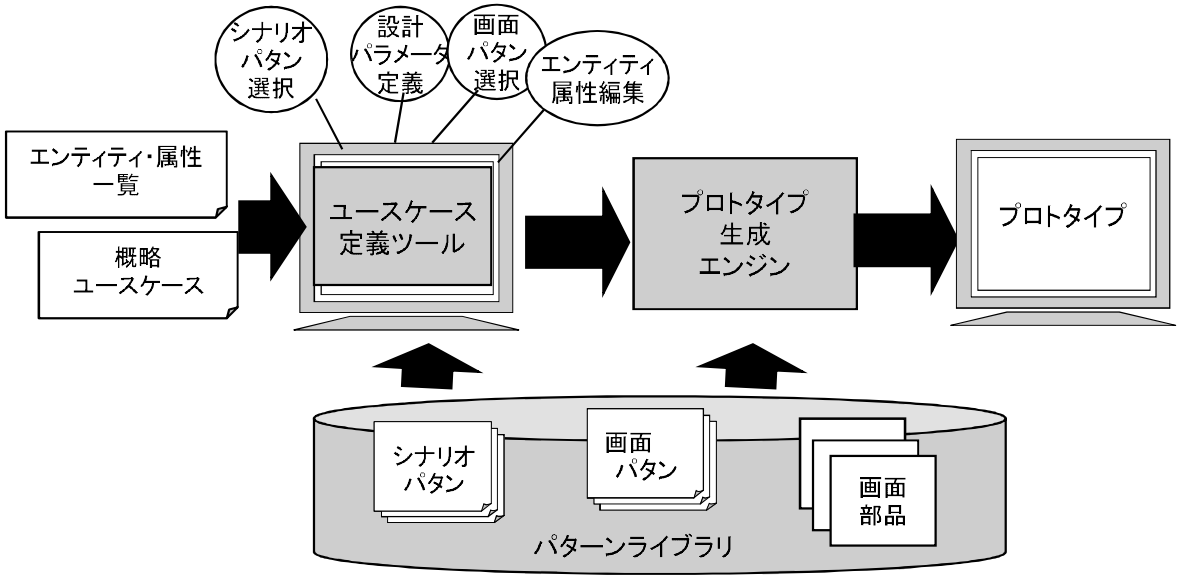


図 5.6　支援環境

5.4.2　パターンライブラリ

開発支援環境では，シナリオパターン，画面パターン，画面部品のラインナップをあらかじめ用意しておき，パターンライブラリに蓄積しておく．

シナリオパターンは，情報システムの汎用的な処理（ログイン，メニュー表示など），データの CRUD 処理（生成，参照，更新，削除），ユースケースシナリオの拡張フロー（例外，分岐など）から 22 のパターンを抽出した．これらのシナリオパターンに対応する画面パターン，画面部品を作成して，ライブラリに登録してある．

シナリオパターンは，その前後関係に制約がある．例えば，業務情報選択パターンは，業務情報検索パターンの後にのみ選択可能である．この前後関係制約をパターンの属性として定義する．

画面パターンは 1 つのシナリオパターンに対して 1 対多の関係を，画面部品は画面パターンと 1 対 1 の関係を持っている．

パターン編集機能もこのコンポーネントでサポートする．パターンの追加，コピー，修正，削除を行い，制約の矛盾をチェックする．

5.4.3　ユースケース定義ツール

ユースケース定義ツールは，シナリオパターンと画面パターンを組み合わせることでユースケースシナリオを作成するツールである（図 5.7）．ツールでは，第 5.3 節で述べた

図 5.1 の step1～step3 に対応し，以下の手順でユースケースを定義する．

step1　パターンライブラリから「シナリオパターン」を選択して追加する．

step2-1　追加したシナリオパターンに含まれる「シナリオ詳細」を参照して，そのシナリオのステップで扱う「エンティティ選択」を行う．

step2-2　シナリオ詳細の各シナリオのステップに対して，選択したエンティティの「エンティティ属性選択」を行い，どの属性に対してどのような操作を行うかを定義する．

step3　選択したシナリオパターンの「画面パターン定義」を行う．

シナリオパターンの選択の際，パターンライブラリで定義されたシナリオパターン間の依存関係を用いて，ユースケースシナリオの状態に対応した支援（追加可能なシナリオパターンの絞り込み，必要な例外処理や分岐処理の漏れに対する警告）を行う．例えば，「シナリオの先頭はメニュー選択から始まる」，「検索結果表示パターンの前には検索条件入力パターンがなければならない」というパターンのとりうる組合せを定義しておき，パターン選択の際に前後のパターン選択状況に応じた選択可能なパターンを絞り込んで表示する．この機能によりパターン選択作業を効率化し，ありえない組合せの選択ミスを防止することができる．また，シナリオパターンに設定するエンティティやその属性は，ER（Entity-Relationship）図のエンティティ単位であらかじめ定義したものを選択する．必要であれば，ユースケース定義ツールを用いてエンティティや属性の追加・変更を行うこともできる．詳細なデータ属性，型の定義や正規化の検討は，シナリオの検討とは別に段階的に行えるようにし，要求の変更により作業の手戻りが最小限になるように考慮した．具体的には，初期の段階では，ツール画面上で「シナリオパターン」「エンティティ選択」「シナリオ詳細」のみを表示し，先にこれらの項目を定義してから残りの項目を表示することもできる．

シナリオのそれぞれのステップには，メモとしてテキストデータを添付することができる．性能，セキュリティなど，シナリオやデータ項目で表現できない非機能的な要求はメモに記述する．メモの内容はプロトタイプには反映されないが，仕様として後工程に引き継ぎ，設計フェーズ以降で改めて設計する．

5.4.4　プロトタイプ生成エンジン

第 5.4.3 項で示したユースケース定義ツールで定義したユースケースシナリオ，画面パターン，エンティティ属性の情報から，画面部品を用いてユースケースを実現するプロトタイプを生成する．このようにユースケースシナリオから直接プロトタイプを生成することで，ユースケースシナリオとプロトタイプの整合性を確保する．

また，プロトタイプを最終アプリケーションと同じプラットフォーム上で生成すること

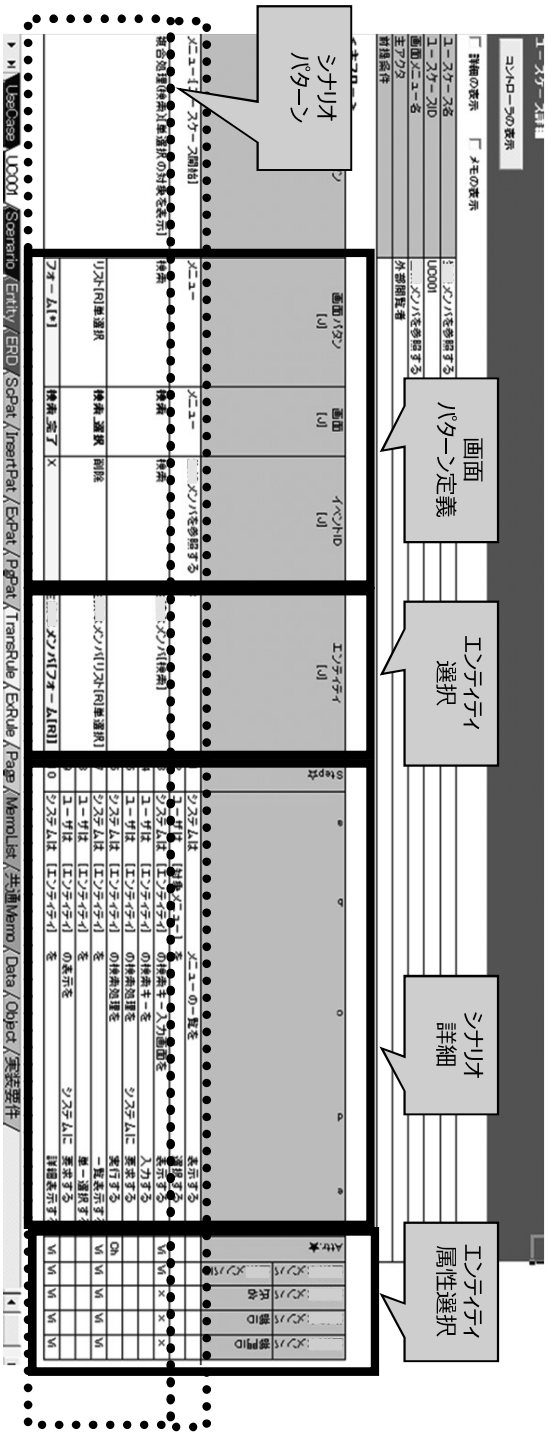


図 5.7　ユースケース定義ツール

表 5.1 評価方法

	要件	評価項目
要件 (1)	成果物のあいまい性を除去し一貫性を保てる	仕様曖昧性に関する指摘数
要件 (2)	ユーザのイメージを喚起できる情報を提示して仕様確認できる	ユースケース・画面イメージのみとプロトタイプを用いた時の指摘数の差
要件 (3)	本質的な仕様を中心に仕様確認できる	従来手法と比較した軽微指摘の要求獲得フェーズ，実装フェーズの分布
要件 (4)	仕様確認に必要な情報の作成を効率化できる	プロトタイプ作成の工数

で，プロトタイプを実装の叩き台として用いることができ，下流工程への移行性と保守性を確保する．

5.5 適用実験

5.5.1 実験の目的

第 5.1 節で述べた課題に対応する第 5.3.2 項で述べた要件，を第 5.3 節で手法で解決できること評価することを目的に，第 5.4.3 節で述べた支援環境を用いて，実システムの開発を行った．

5.5.2 実験方法

対象システムへの適用を通じて，第 5.3.2 項で述べた要件を満たしていることを確認するために表 5.1 で示す評価項目を計測し評価した．

まず，要件 (1)「成果物のあいまい性を除去し一貫性を保てる」という要件の確認のために，ユースケース，画面イメージとプロトタイプが矛盾しているという仕様確認指摘の有無を計測した．次に要件 (2)「ユーザのイメージを喚起できる情報を提示して仕様確認で

きる」の確認のために，ユースケース，画面イメージのみのケースとプロタイプも用いたケースのユーザからの指摘数の差を計測した．さらに，要件 (3)「本質的な仕様を中心に仕様確認できる」の確認のために，仕様確認のタイミングによる指摘の内容の変化を計測する．ここで，指摘内容の分類は次とした．

1.「機能」: はシステムの機能に関する指摘.
2.「画面」: 画面の遷移やレイアウト，操作方法に関する指摘.
3.「データ」: システムで扱うデータに関する指摘.

最後に，要件 (4)「仕様確認に必要な情報の作を効率化できる」の確認のために，仕様確認およびプロトタイプ作成に要した工数を計測した.

5.5.3　対象システムとユーザ

対象システムは，ソフトウェア開発のプロジェクト管理システムとした．このシステムは，プロジェクトサポート部署が利用するもので，それぞれが担当する開発プロジェクトの状況とサポート内容を報告・管理するシステムである．プロトタイプの仕様確認をしてもらうユーザは，本システムの管理者と本システムを主に利用する担当者の 2 名の合計 3 名に参加してもらった．担当者のうち 1 名は本システムに類似するシステムの開発経験があった.

5.5.4　開発プロセス

対象システムの開発は 図 5.2 で示したプロセスに従って行った.

要求獲得フェーズ

実験は，開発者がユースケースとプロトタイプを作成し，ユーザによるレビューを行い，ユーザの持つ要求と異なる部分を指摘してもらう形で行った．前回の指摘事項を反映させた結果を次回に反映させる形式で 5 回仕様確認を繰り返した．仕様確認は，適用実験の対象システムの各成果物ごとの要求獲得の効果を確認するために以下の手順で行った.

(A)　ユースケースシナリオの仕様確認
(B)　画面構成を紙に印刷したもの（画面イメージ）を用いた仕様確認
(C)　プロトタイプの仕様確認

仕様確認は，1 回あたり（A）～（C）の合計で 2 時間かけて行った.

設計フェーズ

設計フェーズは，要求獲得フェーズで獲得した要求を実装するための物理 DB 設計，コンポーネント設計を行うフェーズである．設計フェーズでは，要求獲得フェーズで得た要求を固定し，設計内容の仕様確認は行わなかった．

実装・テストフェーズ

設計フェーズで行った設計に基づいてシステムを開発するフェーズである．完成したシステムも上記ユーザが 2 回仕様確認し，得られた指摘に対して修正を行った．このフェーズでの仕様確認は，1 回あたり 1 時間かけて行った．

上記プロセスの実行に先立ち，下記の 3 つをインプット情報としてあらかじめ用意した．

1. 概略ユースケース図
   対象システムで扱うユーザの主なユースケースを列挙したもの（シナリオを含まない）
2. 概略 ER 図
   対象システムの扱う主なエンティティとその主な属性を図にしたもの
3. パターンライブラリ
   シナリオパターン，画面パターン，画面部品とその関係

5.6 実験結果

5.6.1 規模の推移

図 5.8 に適用プロジェクトでのユースケース数の推移を示す．ここでのユースケースの粒度は，1 人のユーザがシステムを通して 1 度に行う操作を 1 ユースケースとし，一般に複数の画面で実現されるものとする．例えば，「プロジェクトの情報を新規に作成する」が 1 つのユースケースとなる．グラフ中の U1〜U5 は要求獲得フェーズで行った仕様確認を示し，E1〜E2 は実装・テストフェーズで行った仕様確認を示す．当初 16 件あったユースケースが要求獲得フェーズでの仕様確認を行う過程で追加され，最終的に 23 件となった．実装・テストフェーズでのユースケースの追加はなかった．

具体的には，U1 で「ユーザ認証が必要だ」という指摘を受け，U2 に認証に関するユースケースを 1 件追加した．U2 で，「データをファイルエクスポートしたい」という指摘を受け，U3 にエクスポートに関連するユースケースを 3 件追加している．U2，U3 で，検索・編集方法に関する機能追加の指摘を受け，それぞれ，U3 に 2 件，U4 に 1 件のユース

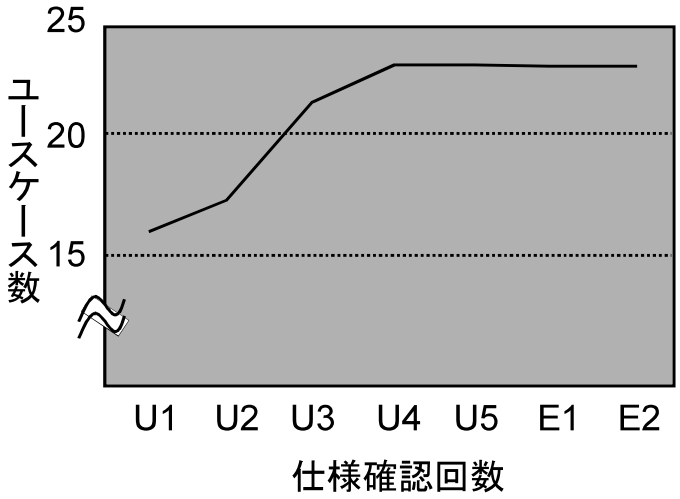


図 5.8　仕様確認とユースケース数の推移

ケースを追加した．

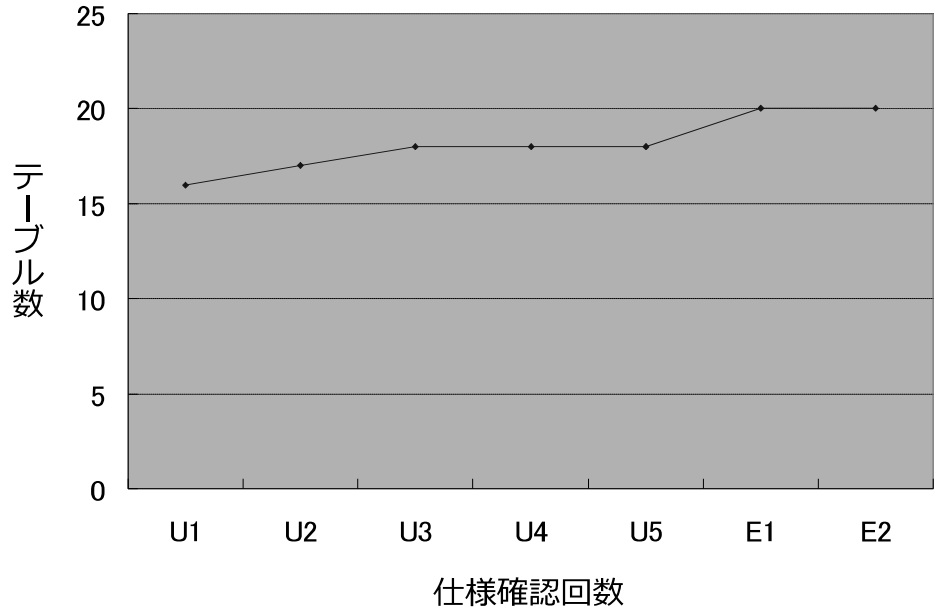
図 5.9 にテーブル数（エンティティ数），図 5.10 に属性数の推移を示す．最終的には 20 のテーブルが必要となったが，ユースケースと同様に，すべて要求獲得フェーズでの指摘を受け追加した（グラフ上では，各仕様確認のプロトタイプに含まれるテーブル数，属性数を示しており，これらの追加の要因となる指摘は 1 つ前の仕様確認で行われたものである）．

具体的には，「データのラインナップはシステムと同一にしてほしい」「基本情報と詳細情報を分けて扱いたい」「データ項目の増減に対応できるようにしてほしい」などの指摘に対応するためにテーブルを追加した．属性は「ここでは ×× も確認したい」などの情報項目の追加につながる指摘を受けたため追加した．

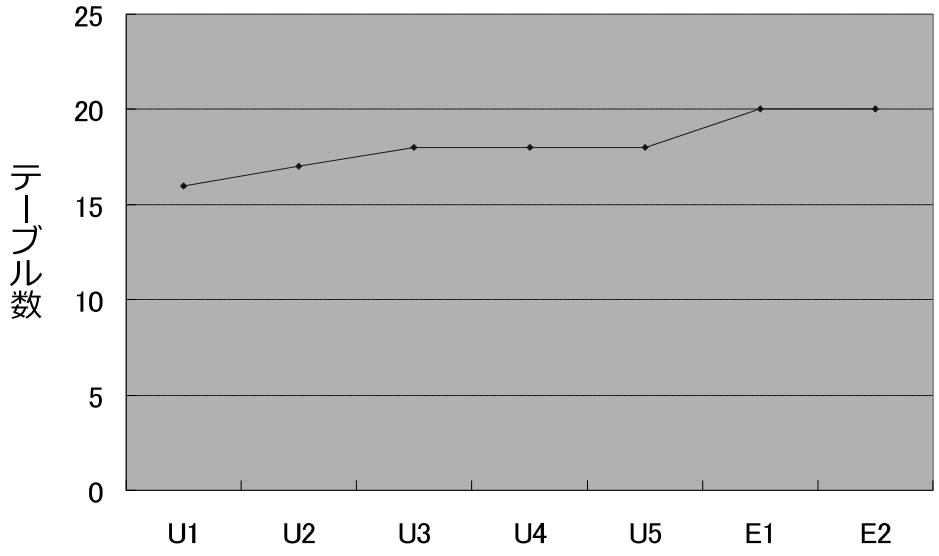
5.6.2　仕様確認での指摘数

図 5.11 に各仕様確認での指摘内容および指摘数の推移を示す．ここで，「業務」はユーザの業務が開発者の想定と食い違っていたことに起因する指摘，「機能」はシステムの機能に関する指摘，「画面」は画面の遷移やレイアウト，操作方法に関する指摘，「データ」はシステムで扱うデータに関する指摘を表す．1 つの指摘が複数の要因にまたがる場合には複数カウントしている．指摘内容としては，全体的に画面に対するものが多く，要求獲得フェーズ，実装・テストフェーズにかかわらず指摘されていた．業務に関する指摘は，要求獲得フェーズの初期に多くなされ，仕様確認を追うごとに収束していく様子が分かる．

図 5.12 に要求フェーズ（要求），実装・テストフェーズ（実装）での指摘数を重要度別



<<<<< HEAD



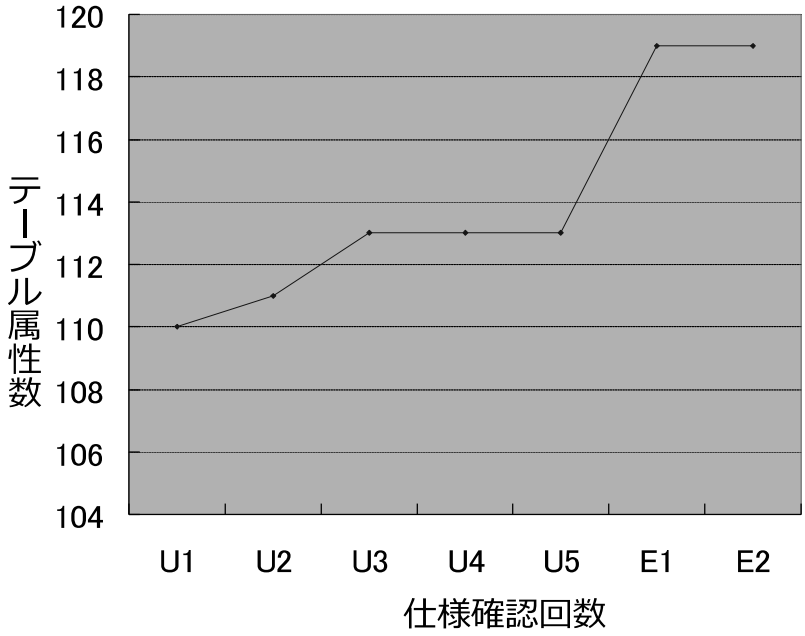
=======

>>>>>

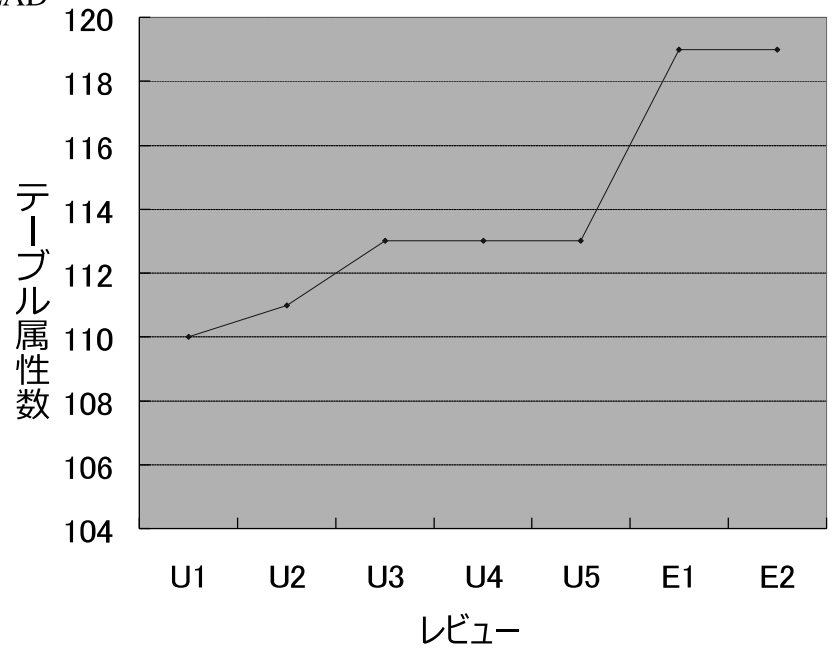
b4e31bd90288851518659 53b310663568a0f1df2

図 5.9　仕様確認とテーブル数の推移

に示す．従来手法は，ユースケースと画面イメージによる仕様確認を想定し，プロトタイプに対して行われた指摘が実装段階に持ち越されると仮定して算出した．重要度は，3 段階（重要：複数ユースケースに影響のある指摘，普通：1 ユースケースの機能に影響のある指摘，軽微：機能には影響なく画面上の修正で対応できる指摘）で分類した．分布によると，ユースケース，画面イメージと一貫したプロトタイプを併用することにより，要求フェーズにおいて 32 件多く（約 27%．内重要な指摘 4 件）指摘がなされている．



<<<<< HEAD



=======

>>>>>

b4e31bd90288851518659 53b310663568a0f1df2

図 5.10　仕様確認と属性数の推移

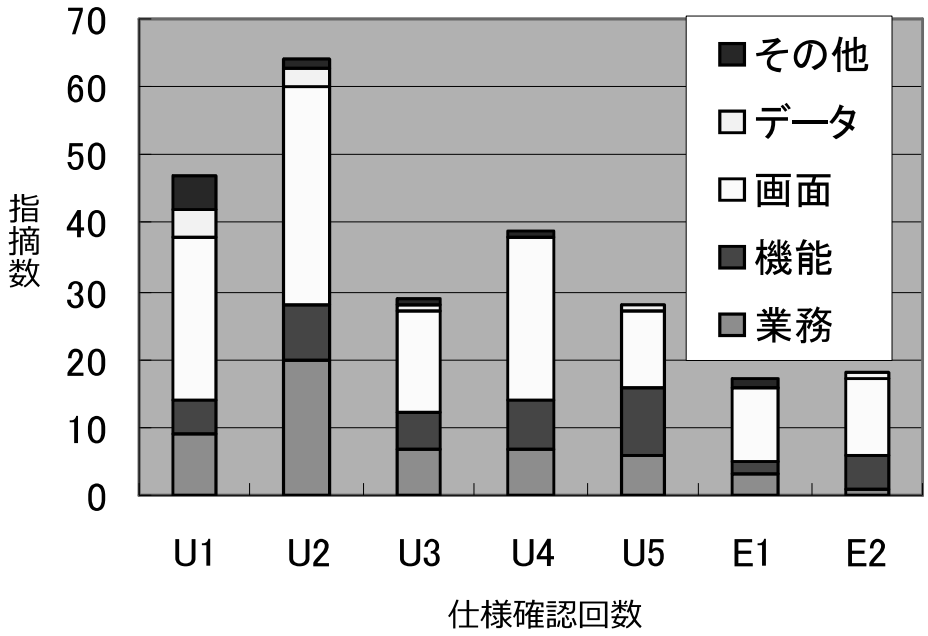


図 5.11　仕様確認と指摘内容および指摘数の推移

5.6.3　工数

仕様確認そのものにかけた工数は，要求獲得フェーズで 50 人時（5 人 × 2 時間 × 5 回）．実装・テストフェーズで 10 人時（5 人 × 1 時間 × 2 回）だった．仕様確認準備は，2 人で行った．パターンの作成に 360 人時，要求獲得フェーズでは，仕様確認準備に 5 回分合計で 361 人時，1 回の平均 約 72 人時かかった．

実装・テストフェーズは，他のプロジェクトと並行して行ったので，正確な工数を測ることができなかった．

5.7　考察

評価実験によって明らかになったことを第 5.3.2 項で述べた要件に対応付けて以下にまとめる．

要件 (1)　成果物のあいまい性を除去し一貫性を保てる．

本章で示した手法では，シナリオパターンを組み合わせることによって，ユースケースシナリオを作成する．組み合わせたパターンは，後にプロトタイプを生成できるくらいに具体的なものであり，ドキュメントと比較してあいまい性が少ない．また，生成したプロトタイプで再度仕様確認することにより，さらにあいまい性を排除で

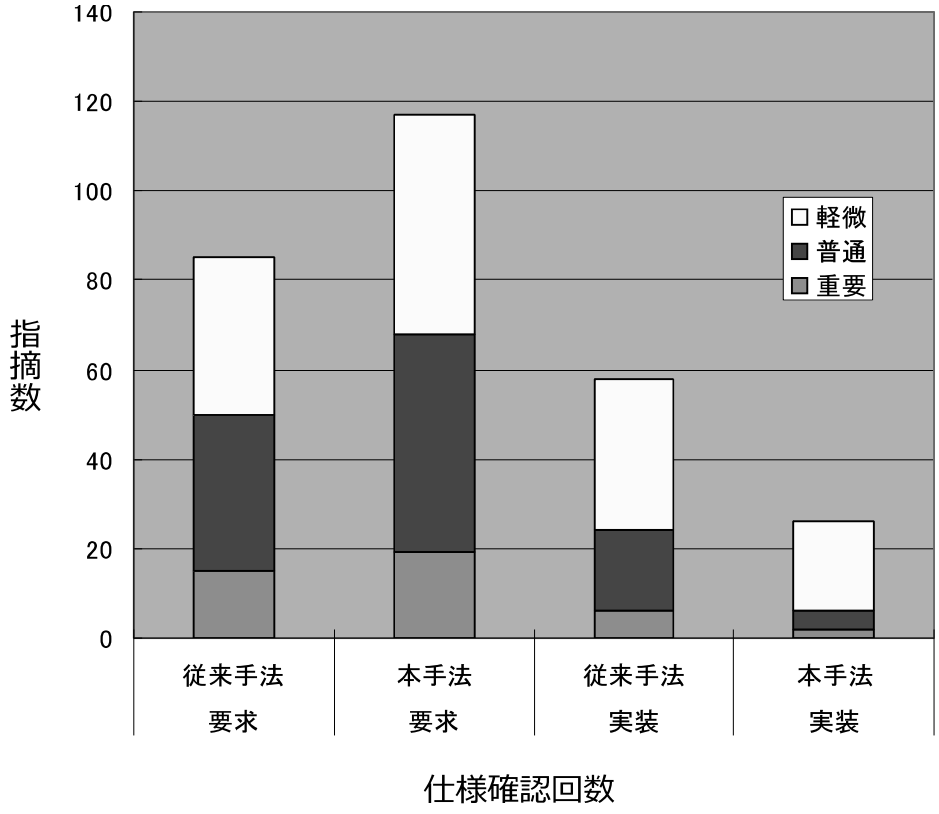


図 5.12　フェーズと指摘重要度および指摘数の推移

きる．仕様のあいまい性に起因する実装・テストフェーズの指摘は 1 件（ID の取扱いに関する仕様の勘違い）のみだった．これは，シナリオで記述しきれなかった内部処理に関する仕様であり，この部分のあいまい性を排除することは今後の課題となる．

要件 (2)　ユーザのイメージを喚起できる情報を提示して仕様確認できる．

本章で提案する手法で作成するプロトタイプは，通常の画面プロトタイプと異なり，ユーザの画面イメージを忠実に作り込むのではなく，画面で扱うべき項目に着目して，画面パターンのテンプレートを用いて，いわばラフスケッチ風に提示するものであった．このようなプロトタイプを用いることで，全体の 3 割近くの指摘を得ることができ，テーブル，属性を要求獲得フェーズで収束させることができた．これは想定通りに，生成したプロトタイプによるイメージ喚起効果が得られたものと考える．これらの指摘はプロトタイプを用いない従来手法では，システム構築後になされる可能性が高い．

要件 (3)　本質的な仕様を中心に仕様確認できる．

画面レイアウトなど画面の見た目に関する要求は，実装段階へ持ち越している．これは，要求獲得フェーズで仕様確認を進めていく際に，プロトタイプで生成できる

レベルの画面仕様を中心に仕様確認を進め，プロトタイプを生成できない細かいレベルの画面仕様は，実装フェーズで行うように誘導できたためである．ここで節約できた仕様確認の時間を業務やデータモデルの話に費やすことができている．これにより当初想定したペーパプロトタイピング相当の効果が得られたと考える．

要件 (4) 仕様確認に必要な情報の作成を効率化できる．

前述のように，ユースケースシナリオとプロトタイピングは，それぞれ要求獲得に効果がある．しかし，両者を矛盾なくメンテナンスしていくには多大な工数を要していた．提案手法では，ユースケースシナリオに画面パターンを対応付けることによりプロトタイプを生成する．このため，常にユースケースとプロトタイプの一貫性を容易に保つことが可能となった．23 のユースケース，約 100 の画面を持つシステムで，72 人時（約 2 人週）でメンテナンス（ユースケースシナリオの修正とプロトタイプ開発）ができたことは，支援環境によるプロトタイプ生成の効果によるところが大きい．

5.8 本章のまとめ

ユースケースシナリオとプロトタイピングを組み合わせた要求獲得手法とその手法を効率的に行うための支援環境を提案した．提案手法は，ユースケースの典型的なシナリオをパターン化したシナリオパターンと，シナリオパターンの画面実装方法をパターン化した画面パターンを用いて，プロトタイプを生成する．本手法より，ユースケースとプロトタイプの一貫性を保ちながら，ユーザと開発者が行った仕様確認のフィードバックをプロトタイプに反映させることができる．プロジェクト管理システムの開発での適用実験において，ユースケースシナリオ，プロトタイピングによって，ユーザの指摘（開発者の想定する仕様とユーザの要求との差異）を促し，後工程に持ち越すと修正範囲の影響の大きい指摘を前もって得ることができることを示した．特に，データモデルのビューに相当するデータ項目の仕様については，本章で示す手法により，要求定義フェーズでユーザ要求が獲得でき，実装フェーズに持ち込まれないことを確認した．

今後の方向性を以下に示す．

実験により，提案手法の実現性と，帳票を中心としたアプリケーションにおいての定性的な有効性は示せたが，定量的な効果を確かめるには，さらなる事例による検証が必要となる．特に，パターンの新規追加の頻度とその影響は分析する必要がある．また，提案手法では，非機能的な要求をメモとして扱い，下流工程への伝達を行った．この部分を支援，設計展開する手法も検討することでさらなる効率向上を実現できると考える．

また，提案手法では，機能的な要求を獲得する目的でパターンを設計したものを利用し

ている．このパターンに GUI のスタイルガイドに適合するためのパターン [17] や人間と計算機のインタラクションを効果的に行うためのパターン [34][3] を取り入れることで，機能面だけではなく，人間中心設計の観点からも効果的な要求獲得が実現できる可能性がある．

第 6 章

仕様復元とプロトタイプ生成の関係

第 4 章では，既存システムのログと組織情報を用いた業務フロー復元について，第 5 章では，仕様からのプロトタイプ生成によるユーザ要求の早期獲得について述べた．本章では，これらの技術を組合わせることでえられる効果について述べる．

6.1　仕様復元からプロトタイプ生成の流れ

図 6.1 に 2 つの技術の関係を示す．図で示したように，まず，第 4 章で述べた手法によって，既存システムから業務フローと業務機能階層図を復元する（仕様復元）．復元した業務フローを用いて，業務の全体像を抽象化アクティビティの粒度でマクロ的に把握し，業務機能階層図を用いて，抽象化アクティビティと操作ステップの関係を得ることができる．次に，第 5 章で述べた手法によって，業務機能階層図の抽象化アクティビティの単位にユースケースシナリオを設計する．その際，抽象化アクティビティに属する各アクションに対して，シナリオパターンを対応付けてユースケースシナリオを定義する．この時，ユーザの新要求があればユースケースシナリオに反映させる．設計したユースケースシナリオに，画面遷移をパターン化した画面パターンを対応付けることによって，ユーザと開発者の仕様確認に用いるプロトタイプを生成することができる．このプロトタイプをユーザに提示することによって，既存システムの仕様を共有し，ユーザからの要求が正しく仕様に反映されていることを確認できる．さらに具体的なシステムのイメージを動く仕様として共有することで，新たな要求を引き出すこともできる．

以降で，具体例を用いて詳細に説明する．

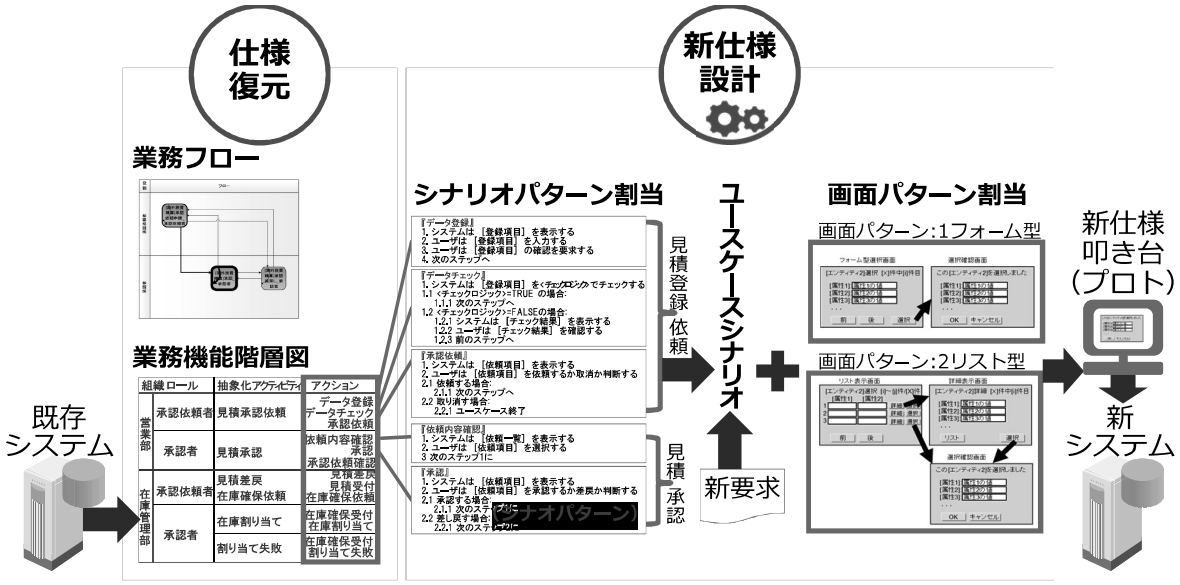


図 6.1　仕様復元とプロトタイプ生成の全体像

6.2　仕様復元からプロトタイプ生成の例

6.2.1　仕様復元

図 6.2，図 6.3 に第 4 章の手法で復元した業務フローと業務機能階層図を示す．業務機能階層図では，業務をロール，代表アクションに代表される抽象化アクティビティ，アクションの階層で表現する．業務フローは，この階層における抽象化アクティビティの粒度での流れをフロー図の形で表現したものである．抽象度の高い抽象化アクティビティの粒度の業務フロー図を用いて，仕様の概要を理解した上で，業務機能階層図を用いて具体的なアクションを確認することで，現行システムの仕様を把握することができる．現行の仕様を把握した上で，新しい要求に従った新仕様へのカスタマイズもこの表現を用いて行うことができる．

図 6.2，図 6.3 では，仕様復元の結果「営業部」と「在庫管理部」という 2 つの組織でそれぞれ「承認依頼者」と「承認者」という合計 4 つのロールが存在することがわかる．さらに，ロールに対して「見積承認依頼」「見積承認」「見積差戻」「在庫確保依頼」「在庫割り当て」「割り当て失敗」という 6 つの抽象化アクティビティが抽出できている．そして，図 6.3 では抽象化アクティビティに対して，それぞれ対応するアクションが復元できている．このように，業務フローで業務の全体像を把握した上で，業務を構成するアクションを階層的に理解し，検討をすることが可能となる．

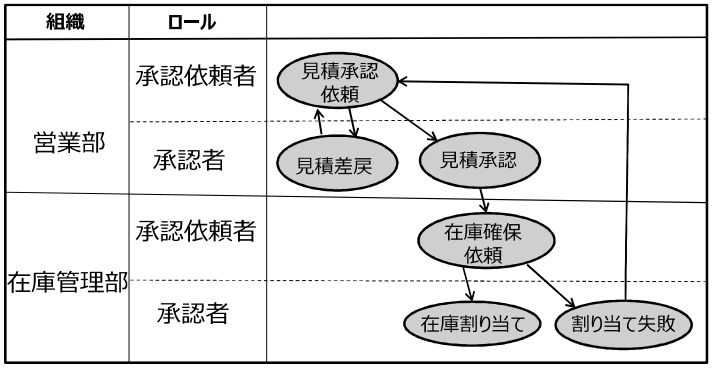


図 6.2　仕様復元（業務フロー）

組織	ロール	抽象化アクティビティ	アクション
営業部	承認依頼者	見積承認依頼	見積登録 登録エラー 見積承認依頼
	承認者	見積承認	承認依頼確認 見積承認
		見積差戻	承認依頼確認 見積差戻
在庫管理部	承認依頼者	在庫確保依頼	見積受付 在庫確保依頼
	承認者	在庫割り当て	在庫確保受付 在庫割り当て
		割り当て失敗	在庫確保受付 割り当て失敗

図 6.3　仕様復元（業務機能階層図）

6.2.2　シナリオパターン

要求獲得をする前に，あらかじめ，図 6.4 に示すエンタープライズシステムで一般的なアクションをシナリオパターンとして用意した．業務における必要な情報を入力しチェックする『データ登録』『データチェック』，担当者が入力した情報を上司に確認依頼し，上司が確認を行う『依頼』『依頼内容確認』『承認』『差戻し』という業務の塊は，特定の業務に依存しない事務処理でよく登場する．これらをシナリオパターンとしている．

6.2.3　アクションへのシナリオパターン対応付け

業務機能階層図の代表アクションとアクションは，第 5 章で述べた手法では，それぞれユースケースとシナリオパターンに対応付けることができる．あらかじめ定義しておいたシナリオパターンに，復元したアクションを対応付けることで，アクションの動きを定義

図 6.4　シナリオパターン

することができる．図 6.5 に復元した抽象化アクティビティ【見積承認依頼】,【見積承認】,【見積差戻し】に対してシナリオを定義した例を示す.

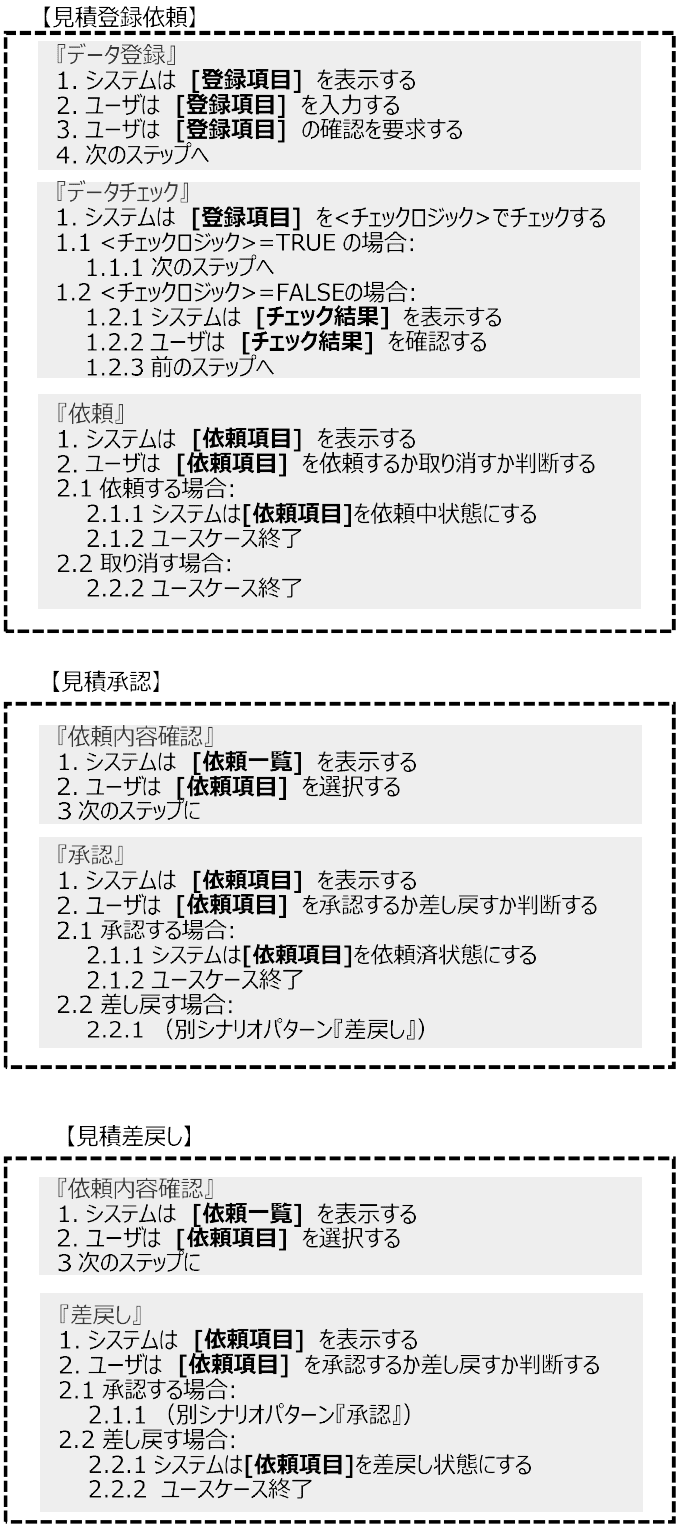


図 6.5　シナリオパターンの対応付け

まず，承認依頼者の抽象化アクティビティ【見積承認依頼】に対応するアクション「見積登録」「登録エラー」「見積承認依頼」に対して，それぞれシナリオパターンから『データ登録』『データチェック』『依頼』を対応付けた．次に，承認者の抽象化アクティビティ【見積承認】に対応するアクション「承認依頼確認」「見積承認」に対して，それぞれシナリオパターンから『依頼内容確認』『承認』を対応付けた．最後に，承認者の抽象化アクティビティ【見積差戻し】に対応するアクション「承認依頼確認」「見積差戻」に対して，それぞれシナリオパターンから『依頼内容確認』『差戻し』を対応付けた．ここで，2 つの抽象化アクティビティ【見積承認】【見積差戻し】で同一のアクション「承認依頼確認」が登場する．これに対してシナリオパターンも同一の『依頼内容確認』を対応付けている．これは，同一のアクションが，抽象化アクティビティのレベルでは異なるアクティビティとして抽出された２つのアクティビティを分解したアクションに含まれることを示している．このことは，異なる業務であっても部分的に同じことを行うことがあるという性質を忠実に復元するために必要なことである．

ここでは，復元したアクションにそのままシナリオパターンを対応付けたが，ユーザの要求を取り込む際には，復元した抽象化アクティビティやアクションと異なる業務をシステムで扱わせたい場合がある．そのような時には，該当するアクティビティに対してシナリオパターンを組み替えた新たな要求に準拠したユースケースシナリオを再構成することで新仕様を定義する．組織における業務が抜本から刷新されることはまれであるため，多くのアクティビティが復元した仕様を踏襲し，新たな要求に影響のあるアクティビティのみを組み替えることで，効率的に新仕様定義を進めることが可能になる．

6.2.4　シナリオパラメータの設定

それぞれのアクションに対応付けられたシナリオパターンに対して，パラメータとなっているデータ項目の設定を行うことで，ユースケースとして完成させることができる．図 6.6 にアクション「見積登録」に対してシナリオパラメータを設定した例を示す．

「見積登録」に対応付けられたシナリオパターン『データ登録』は，登録するデータの項目がパラメータ [登録項目] として切り出されている．このパラメータに対して，既存システムのデータスキーマから該当するエンティティを設定する．ここでは，エンティティ「見積り」をパラメータ [登録項目] に設定している．

ユーザの新たな要求を取り込む際には既存システムでは扱っていない項目が追加される可能性がある．その時には，エンティティを拡張し項目を追加した上で設定することで，ユーザの要求を取り込んだ新仕様の定義を可能にする．

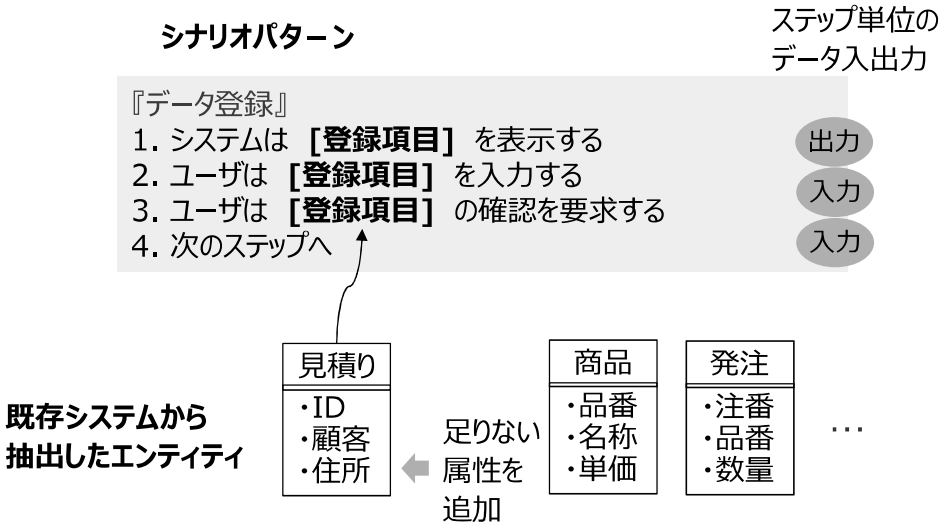


図 6.6　シナリオパラメータの設定

6.2.5　画面パターン対応付け

シナリオパターンに対して，画面パターンを対応付けることでプロトタイプを生成することが可能になる．アクション「見積登録」に対して GUI パターンの対応付けのイメージを図 6.7 に示す．「見積登録」は，多くの項目の入力を行う．この時の入力の効率化のため，入力するデータを過去の入力結果をコピーして修正する機能が必要となる．この機能の実現にはいくつかの方式が存在する．スマートフォンなど限られた画面で表示する必要がある場合にはレコード単位で入力項目を表示して切り替えて選ぶ形式（フォーム型）が有効である．一方 PC 画面など画面を広く使える場合にはレコードの主要項目の一覧を出してそこから選ぶ形式（リスト型）が有効である．ここで示した画面遷移に関する仕様は，業務上は本質的ではないが，実装するプラットフォームによって決定される性質の仕様である．提案手法では，業務上の仕様と区別することで議論すべき仕様のスコープを明確にすることができる．

6.2.6　プロトタイプ生成とユーザ確認

本章で述べてきたように，復元した業務フローで抽出した抽象化アクティビティに対応するアクション群に対して，シナリオパターンを対応付け，パラメータを定義し，画面パターンを対応付けることによって，ユースケースシナリオを具体化することができる．このユースケースシナリオに登場するアクションは，扱うデータを示すエンティティと表示形式を定める画面レイアウトが関連付けて定義されているため，画面，処理，DB からなる

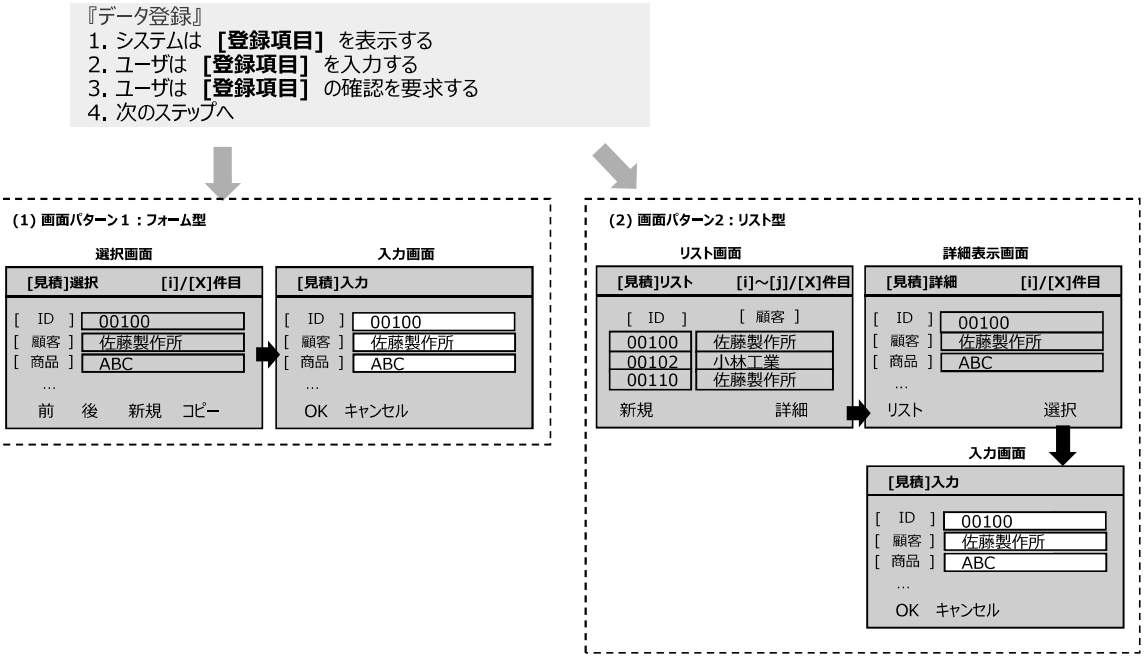


図 6.7　GUI パターン

プロトタイプを生成することが可能になる．このプロトタイプを用いてユーザと開発者が仕様確認をすることによって，ユーザの要求が正しく開発者に伝わったことを確認するとともに，動作可能な具体的な動く仕様によって新たな要求を獲得しやすくなる．

6.3 技術全体としての効果

第 6.2 節で示した例のように，仕様復元技術で復元した仕様（業務フロー図，業務機能階層図）に対してシナリオパターン，画面パターンを対応付けていくことによって，エンドユーザへの仕様確認を効率的に行うことができるプロトタイプを生成することができる．これにより，既存システムから，現行ユースケースを抽出し，これに新たな要求を反映させた新ユースケースを定義し，新ユースケースから生成したプロトタイプをユーザに提示することができる．その結果，ユーザからのフィードバックを早期に得られる．以上により，第 3 章で掲げていたアプローチ (図 6.8 再掲) の 2 つのサブゴール

SG1:仕様復元

正確なドキュメントが残っていないシステムの業務レベルの仕様（業務機能階層，業務フロー）をを復元する．

SG2:プロトタイプ生成

SG1 の結果を叩き台に，ユーザと開発者があるべき仕様を合意するためのプロトタイプを生成する．

に対して，以下が実現できた．

- 既存システムの情報を用いて業務仕様を人手をかけずに復元．
- 復元した仕様をたたき台にした新プラットフォーム向けのプロトタイプを生成．
- プロトタイプをユーザと開発者が確認し新たな得られた要求を反映させてプロトタイプを再生成．

これにより，レガシー化した既存システムでは修正コストがかかりすぎて実現が困難だった，動く仕様によるユーザと開発者との新仕様に関する合意形成を効率的に行うことが可能になった．また，シナリオパターンという粒度で合意形成を行うことにより，システム開発において他に影響の大きい機能構成やデータ構造にかかわる仕様を中心に議論し，影響の少ない些末な仕様よりも優先的に決めることが可能になった．

また，生成したプロトタイプを提示することによって新たな要求が得られた場合には，仕様（シナリオ，パターン割当て，パラメータ）を変更することになる．その際，その要求によって影響を受けるシナリオから，関係するパターン，パラメータで指定しているデータ項目への対応関係が明確になっているため，修正漏れによる仕様矛盾が生じにくい．また，上記仕様から新たなプロトタイプを再生成することで，実装レベルでの修正に比べて早く修正に対応することができる．

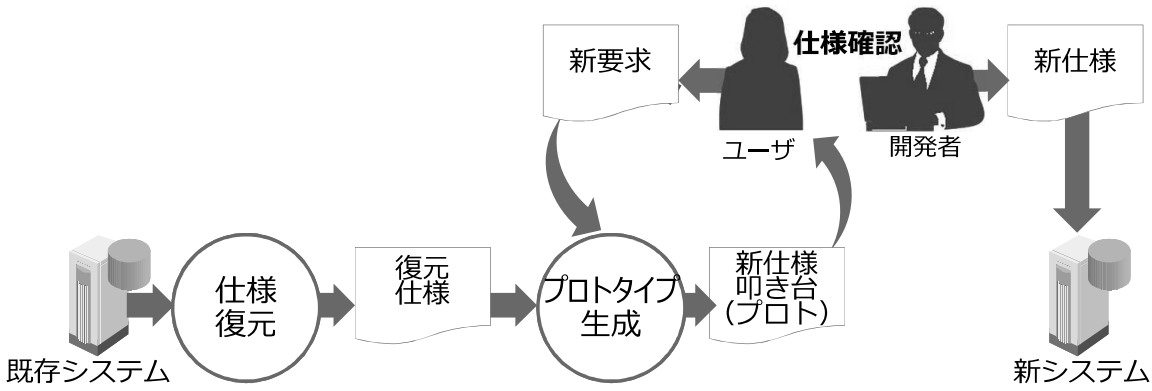

新要求
仕様確認
新仕様
ユーザ
開発者
仕様
復元
復元
仕様
プロトタイプ
生成
新仕様
叩き台
(プロト)
既存システム
新システム

図 6.8　アプローチ

第 7 章

結論

7.1　おわりに

エンタープライズシステムは，対象組織が担っている業務の一部を計算機で支援することによって，業務の効率化や品質向上を行うことを目的としているシステムである．IT 技術の進歩によって多くの組織では既にシステムが導入されている．この中には，開発当時から 20 年〜30 年という長い期間使われ続けているものも存在している．

エンタープライズシステムが対象とする業務は，時が経つにつれて変化するが，業務全体が刷新されることはまれであり，多くの業務が現行を踏襲する中で一部の業務が要求に応じて変更される．そのため，現行システムの仕様を踏襲しつつ新たな要求を取り入れた仕様を設計し構築する必要がある．

しかしながら，既存システムの仕様は，明確に文書化されているとは限らない．ソースファイルや実行モジュールしか存在しない場合や，文書化されていても修正や拡張が反映されておらず，実際に動いているシステムとは乖離してしまっていることも少なくない．本研究は，このように既存システムに影響を受けるシステムを開発する際に，既存システムの情報を効率的に活用し，システム開発の生産性を向上することを目的とした．

これらは，エンタープライズシステムの開発においてそれぞれ一定の課題を解決しているが，依然として解決に至っておらず，以下の課題が残っていた．

1. 仕様復元の課題
   正確なドキュメントが残っていないシステムの業務仕様を復元することにコストがかかる. 既存のリエンジニアリング技術によって，処理レベルの仕様の復元は可能になったが，システムのリエンジニアリングの際にユーザとの合意に用いる業務レベルの仕様の復元にはヒアリングや仕様確認などの人による作業が必要になる．
2. 新仕様設計の課題

漠然としたユーザの要求を業務仕様として記述し合意をとるのに手間がかかる. ユーザは具体的な仕様を提示されれば, その良しあしを判断することは可能であるが, ゼロからユーザが具体的な要求を仕様として明確に提示することができないことも多く，ユーザと開発者の試行錯誤により要求獲得が行われ，新仕様が定義される．その際の検討漏れやあいまいな決定が，開発後半の大きな手戻りにつながってしまう.

上記の課題を解決するために本研究では，仕様復元，新仕様設計それぞれのサブゴール (SG) を定めた.

SG1 仕様復元　　業務レベルの仕様（業務機能階層，業務フロー）を復元する.

エンタープライズシステムの仕様復元では，4 種類の仕様（業務，画面，ロジック，ER モデル）が必要であるが，このうち，既存技術で解決できていない業務レベルの仕様復元を行う．業務レベルの仕様は SG2 のインプットとなるものとする.

SG2 新仕様設計　　SG1 の結果を叩き台に，ユーザと開発者があるべき仕様を合意するためのプロトタイプを生成する.

ユーザからの要求を引き出すために，より具体的にシステムの利用イメージを持ってもらうことが重要である．そのため，復元した仕様をベースにプロトタイプを生成し，より具体的な動く仕様案としてユーザに提示する．これにより，ユーザと開発者の具体的な仕様を共通言語とし，要求獲得を効率化する.

SG1 として，正確な仕様が残されていない既存の業務システムを理解するために，システムログと組織情報から業務フローを復元する手法を提案した．提案した復元手法では，複数のユーザによって行われる業務に対し，ユーザの切れ目に着目することによって，ロールや代表的なアクションを抽出し，抽象化されたアクティビティを復元することができる．ケーススタディによる評価では，ロール復元精度が 99.5%, 抽象化アクティビティの再現率が 99.2% であり，本手法の有効性を示すことができた.

SG2 として，ユーザと開発者の間でこれから開発する仕様の具体イメージを共有し，要求獲得を早期に実施するために，ユースケースシナリオとプロトタイピングを組み合わせた要求獲得手法とその手法を効率的に行うための支援環境を提案した．提案した要求獲得手法では，ユースケースの典型的なシナリオをパターン化したシナリオパターン，シナリオパターンの画面実装方法をパターン化した画面パターンを用いて，プロトタイプを生成する．本手法により，ユースケースとプロトタイプの一貫性を保ちながら，ユーザとの仕様確認のフィードバックをプロトタイプに反映させることができる．プロジェクト管理システムの開発での適用実験において，ユースケースシナリオ，プロトタイピングによって，ユーザの指摘（開発者の想定する仕様とユーザの要求との差異）を促し，後工程に持ち越すと手戻りコストの高い指摘を前もって得ることができることを示した．特に，データモ

デルのビューに相当するデータ項目の仕様について，本手法により，要求定義フェーズでユーザ要求が獲得でき，実装フェーズに持ち込まれないことを確認した．

以上の２つのサブゴールを実現したことによって，全体として以下が達成できる．SG1 の成果によって復元した業務フローを用いて，既存システムの業務仕様の全体像をユーザと開発者が共有できる．この上で，業務フローの各ノードである抽象化アクティビティに対応するアクション群に対して，シナリオパターンを対応付け，パラメータを定義し，画面パターンを対応付けることによって，ユースケースシナリオを具体化しすることができる．SG2 の成果を用いて，このユースケースシナリオから画面，処理，DB からなるプロトタイプを生成することが可能になる．このプロトタイプを用いてユーザと開発者が仕様確認をすることによって，動作可能な具体的な動く仕様によって新たな要求を獲得しやすくなる．これをユースケースシナリオにフィードバックして，再度プロトタイプを生成し確認することを繰り返すことができる．この一連のプロセスにより，既存システムのリエンジニアリングを効率的に行うことが可能になる．

7.2 今後の方向性

本論文で述べた SG1，SG ２それぞれの手法の，今後の研究の方向性について述べる．

7.2.1 仕様復元の精度および導入効率向上

SG1 で提案した手法を改良し，復元精度や適用の際の効率を向上させることによって，本論文で示した効果をさらに向上させることが期待できる．その方向性を以下に示す．

1. 今回は閾値に固定値を与えたが，ログの特性を前もってスクリーニングすることによって，適切な値を算出できる可能性がある．
2. 事前に途中開始や途中終了の案件をフィルタリングして，分析するなどの前処理を行うことによって，プロセスインスタンスとしてふさわしくないデータをノイズとして除去し，より高いカバー率を得ることができる可能性がある．
3. 複数のプロセスインスタンスにまたがる業務（過去のケースを参照して，別のケースを承認するなど）を扱うことができるようにプロセスインスタンスの考え方の拡張を行うことで，再現率をあげられる可能性がある．
4. 再現率優先の要件により，抽象化アクティビティの遷移の適合率は 0.84 となっている．遷移の頻度やブロックのまたがり状況によるフィルタリングをすることにより，この適合率が向上し，仕様理解の負担をさらに減少させられる可能性がある．

実験により，本提案の実現性と，旅費精算のような事務処理を中心としたアプリケー

ションにおいての定性的な有効性は示せたが，定量的な効果を確かめるには，さらなる事例による検証が必要となる．特に，パターンの新規追加の頻度とその影響は分析する必要がある．また，本章の提案手法では，非機能的な要求をメモとして扱い，下流工程への伝達を行った．この部分を支援，設計展開する手法も検討することでさらなる効率の向上を実現できると考える．

7.2.2　ユーザビリティ・セキュリティ要件を考慮したパターン化

SG2 では，ユーザとシステムのやり取りするデータ項目とタイミングに関する要求を獲得する目的でパターンを設計したものを利用した．このパターンに GUI のスタイルガイドに適合するためのパターン [17] や人間と計算機のインタラクションを効果的に行うためのパターン [34][3] を取り入れることで，機能面だけではなく，人間中心設計の観点からも効果的な要求獲得が実現できる可能性がある．また，セキュリティの脅威分析を行い，それに対する対策がなされた操作手順をパターン化しておき，そのパターンを利用することで脅威を取り除くことも期待できる．

発表論文一覧

論文

- Web アプリケーションのユースケース駆動プロトタイプによる要求獲得方法
  三部良太, 河合克己, 竹内拓也, 石川貞裕, 福士 有二:
  情報処理学会論文誌 Vol.49　No.4, 1669-1679, 2008.
- システムログと組織情報を活用した業務フロー仕様復元
  三部良太, 田中匡史, 小林伸悟，小林隆志 情報処理学会論文誌 Vol.59 No.4,pp.1150-1160 ，2018.

国際会議

- Business Process Recovery based on System Log and Information of Organizational Structure
  Ryota Mibe, Tadashi Tanaka, Takashi Kobayashi Shingo Kobayashi:
  Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER2017), pp. 531–535, 2017

国内会議

- システムログと組織情報を活用した業務フロー仕様復元に向けて
  三部良太, 田中匡史, 小林隆志, 小林伸悟:
  電子情報通信学会 信学技報 No.SS2016-15, pp. 143-148, 2016.

謝辞

本研究にあたり，熱心にご指導いただきました指導教官の小林隆志准教授に深く感謝いたします．そして，徳田雄洋先生には前段の研究のご指導をいただきました．また，貴重なご意見をいただき実験にも協力していただいた小林研究室，旧徳田研究室の皆様に深くお礼申し上げます．

また，本論文で述べた手法やツールのアイデアは，株式会社日立製作所が長年積み重ねてきたソフトウェア工学の研究の積み重ねの結果に触発されている．横横浜研究所や関連する事業部の方々は，議論を通じて多くのアドバイスをいただくとともに，適用実験の題材提供や評価などで多大な協力をいただきました．深く感謝いたします．

最後に，平日の夜，週末，ゴールデンウィーク，夏休み，リフレッシュ休暇を研究や執筆に費やすことを許してくれた家族に感謝します．

参考文献

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of of International Conference on Management of Data*, pp. 207–216, 1993.

[2] Paul Beynon-Davies, Chris Carne, Hugh Mackay, and Douglas Tudhope. Rapid application development (RAD): an empirical review. *European Journal of Information Systems*, Vol. 8, No. 3, pp. 211–223, 1999.

[3] J.O. Borchers. A pattern approach to interaction design. In *Proceedings of Conference on Designing Interactive Systems*, pp. 369–378, 2000.

[4] Elliot J. Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *IEEE software*, Vol. 7, No. 1, pp. 13–17, 1990.

[5] A. Cockburn. *Writing Effective Use Cases (Agile Software Development Series)*. Addison-Wesley Pub., 2000.

[6] Santiago Comella-Dorda, Kurt Wallnau, Robert C Seacord, and John Robert. A survey of legacy system modernization approaches. Technical Report CMU/SEI-2000-TN-003, Carnegie-Mellon University Software Engineering Institute, 2000.

[7] Chiara Di Francescomarino, Alessandro Marchetto, and Paolo Tonella. Cluster-based modularization of processes recovered from web applications. *Journal of Software: Evolution and Process*, Vol. 25, No. 2, pp. 113–138, 2013.

[8] Mohammed Elkoutbi, Ismaïl Khriss, and Rudolf K. Keller. Automated prototyping of user interfaces based on UML scenarios. *Automated Software Engineering*, Vol. 13, No. 1, pp. 5–40, 2006.

[9] Chiara Di Francescomarino, Alessandro Marchetto, and Paolo Tonella. Reverse engineering of business processes. In *Proceedings of 13th European Conference Software Maintenance and Reengineering (CSMR)*, pp. 139–148, 2009.

[10] A. Ghose, G. Koliadis, and A. Chueng. Process discovery from model and text artifacts. In *Proceedings of International Workshop on Service- and Process-Oriented Software*

Engineering (SOPOSE), pp. 167–174, 2007.

[11] OBJECT MANAGEMENT GROUP. *Unified Modeling Language: Superstructure Version 2.1.1*. OMG formal/2007-02-05, 2007.

[12] Anneke G Kleppe, Jos B Warmer, and Wim Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[13] Maikel Leemans and Wil van der Aalst. Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. In *Proceedings of 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 44–53, 2016.

[14] D. Leffingwell and D. Widrig. *Managing Software Requirements: A Unified Approach*. Addison-Wesley Pub., 1999.

[15] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. *End-user development: An emerging paradigm*. Springer, 2006.

[16] Charles X. Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *Proceedings of 4th International Conference on Knowledge Discovery and Data Mining*, pp. 73–79, 1998.

[17] M.J. Mahemoff and L.J. Johnston. Principles for a usability-oriented pattern language. In *Proceedings of Computer Human Interaction Conference, 1998.*, pp. 132–139, 1998.

[18] Ryota Mibe, Tadashi Tanaka, Takashi Kobayashi, and Shingo Kobayashi. Business process recovery based on system log and information of organizational structure. In *Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 531–535, 2017.

[19] Gail C. Murphy, David Notkin, and Kevin Sullivan. Software reflection models: Bridging the gap between source and high-level models. In *Proceedings of 3rd Symp. Foundation of Software Engineering (FSE)*, pp. 18–28, 1995.

[20] Cowan N. *Working memory capacity*. Psychology Press, 2005.

[21] Shinpei Ogata and Saeko Matsuura. Evaluation of a use-case-driven requirements analysis tool employing web UI prototype generation. *WSEAS Transactions on Information Science and Applications*, Vol. 7, No. 2, pp. 273–282, 2010.

[22] B. Paradauskas and A. Laurikaitis. Business knowledge extraction from legacy information systems. *Journal of Information Technology and Control*, Vol. 35, No. 3, pp. 214–221, 2006.

[23] Ricardo Pérez-Castillo. Marble: Modernization approach for recovering business processes from legacy information systems. In *Proceedings of 28th International Conference Software Maintenance (ICSM)*, pp. 671–676, 2013.

[24] Luiz A. Pacini Rabelo, Antonio F. do Prado, Wanderley L. de Souza, and Luis F. Pires. An approach to business process recovery from source code. In *Proceedings of 12th International Conference Information Technology New Generations(ITNG)*, pp. 361–366, 2015.

[25] Marcela Ridao, Jorge Doorn, and Julio Cesar Sampaio do Pado Leite. Domain independent regularities in scenarios. In *Proceedings of 5th IEEE International Symposium on Requirements Engineering*, pp. 120–127, 2001.

[26] Atanas Rountev and Beth Harkness Connell. Object naming analysis for reverse-engineered sequence. In *Proceedings of 27th International Conference Software Engineering (ICSE)*, pp. 254–263, 2005.

[27] Shinobu Saito, Jun Hagiwara, Tomoki Yagasaki, and Katsuyuki Natsukawa. Requirements model oriented system generation tool and its practical evaluation. In *Proc. COMPSAC2014*, pp. 535–540, 2014.

[28] Shinobu Saito, Jun Hagiwara, Tomoki Yagasaki, and Katsuyuki Natsukawa. ReVAMP: Requirements validation approach using models and prototyping — practical cases of requirements engineering in end-user computing. *Journal of Information Processing*, Vol. 23, No. 4, pp. 411–419, 2015.

[29] Douglas C Schmidt. Model-driven engineering. *IEEE Computer*, Vol. 39, No. 2, pp. 25–31, 2006.

[30] Carolyn Snyder, (黒須正明訳). ペーパープロトタイピング. オーム社, 2004.

[31] Stephen L Squires, Martha Branstad, and Marvin Zelkowitz. Special issue on rapid prototyping. *ACM SIGSOFT Software Engineering Notes*, Vol. 7, p. 5, 1982.

[32] Stéphane S.Somé. Supporting use case based requirements engineering. *Information and Software Technology*, 2006.

[33] Richard H Thayer, Sidney C Bailin, and M Dorfman. *Software requirements engineerings*. IEEE Computer Society Press, 1997.

[34] Jenifer Tidwell, (ソシオメディア株式会社監訳, 浅野紀予訳). デザイニング・インタフェースパターンによる実践的インタラクションデザイン. オライリー・ジャパン, 2007.

[35] Wil van der Aalst. *Process Mining: Descovery, Conformance and Enhancement of Business Process*. Springer, 2011.

[36] Wil van der Aalst. *Process Mining - Data Science in Action*. Springer, second edition, 2016.

[37] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald

Brandtjen, Joos Buijs, et al. Process mining manifesto. In *Business Process Management Workshops(BPM)*, pp. 169–194, 2012.

[38] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[39] R. Vonk. *Prototyping: The Effective Use of Case Technology*. Prentice Hall, 1990.

[40] K. Watahiki and M. Saeki. Scenario patterns based on case grammar approach. In *Proceedings of 5th IEEE International Symposium on Requirements Engineering*, pp. 300–301, 2001.

[41] Jochen De Weerdt, Seppe vanden Broucke, Jan Vanthienen, and Bart Baesens. Active trace clustering for improved process discovery. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, No. 12, pp. 2708–2720, 2013.

[42] Richard L Wexelblat. *History of programming languages*. Academic Press, 1981.

[43] M. V. Wilkes and W.Renwick. The EDSAC (Electronic Delay Storage Automatic Calculator). In *Math. Comp.*, Vol. 4, pp. 61–65, 1950.

[44] Ying Zou, Terence. C. Lau, Kostas Knotogiannis, Tack Tong, and Ross MeKegney. Model driven business process recovery. In *Proceedings of 11th Working Conference Reverse Engineering (WCRE)*, pp. 224–233, 2004.

[45] ソフトウェア高信頼化センタ（SEC). システム再構築を成功に導くユーザガイド. 独立行政法人情報処理推進機構 (IPA), 2017.

[46] 熊澤壽. ERP 導入による効果と難易度の実際. 日本オペレーションズ・リサーチ学会論文誌「オペレーションズ・リサーチ: 経営の科学」, Vol. 49, No. 6, pp. 352–358, 2004.

[47] 総務省情報通信国際戦略局経済産業省大臣官房調査統計グループ. 情報通信業基本調査報告書. 総務省情報通信国際戦略局, 2017.

[48] 薦田憲久, 水野浩孝, 赤津雅晴. ビジネス情報システム. コロナ社, 2005.

[49] 大西淳, 阿草清滋, 大野豊. 要求フレームに基づいたソフトウェア要求仕様化技法. 情報処理学会論文誌, Vol.31, No.2, pp. 175–181, 1990.

[50] 田中淳. 金融機関向けソリューション・システムの全体像と今後の取組み. UNISYS TECHNOLOGY REVIEW 第 77 号, pp. 3–11, 2003.

[51] 武内淳. 推敲モデルを用いたソフトウェア設計ドキュメントの推敲方式の提案. 電気学会論文誌 C, Vol.123, No.10, pp. 1892–1900, 2003.

[52] 森雅俊. ERP パッケージ導入における評価とその課題. システム制御情報学会誌「システム/制御/情報」, Vol. 44, No. 1, pp. 21–24, 2000.

[53] 小形真平, 松浦佐江子. UML 要求分析モデルからの段階的な Web UI プロトタイプ自動生成手法. コンピュータソフトウェア, Vol. 27, No. 2, pp. 14–32, 2010.

[54] 青山幹雄. e-ビジネスを実現するソフトウェアサービス技術: ソフトウェアサービス技術へのいざない. 情報処理学会誌「情報処理」, Vol. 42, No. 9, pp. 857–862, 2001.

[55] 中谷多哉子, 玉井哲雄. ユースケース記述のためのフレームワークとメタモデル. オブジェクト指向 2000 シンポジウム論文集, pp. 141–148, 2000.

[56] 田中匡史, 伊藤秀朗, 中川雄一郎. 業務フロー仕様生成技術の開発. 電子情報通信学会技術研究報告 「信学技報」, Vol. 115, No. 249, pp. 1–5, 2015.

[57] 日本情報システムユーザ協会. 企業 IT 動向調査 2012. 日本情報システムユーザ協会, 2012.

[58] 日本情報システムユーザ協会. 企業 IT 動向調査 2015. 日本情報システムユーザ協会, 2016.

[59] 白銀純子, 深澤良彰. ユースケースの撚合せによる GUI プロトタイプ生成手法. 第８回日本ソフトウェア科学会ソフトウェア工学の基礎ワークショップ（FOSE）, pp. 71–82, 2001.

[60] 牧野友紀. ビジネス環境と実装システムを繋ぐ BPM と SOA. 情報処理学会誌「情報処理」, Vol. 46, No. 1, pp. 60–63, 2005.