情報処理学会論文誌 Vol. 49 No. 4 1669–1679 (Apr. 2008)

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[†1] 河合克己[†1] 竹内拓也[†1]
石川貞裕[†2] 福士有二[†2]

要求獲得は，システム開発の最も重要なことの1つであり，文書，モデル，プロトタイプなどをユーザにレビューしてもらうことによって行われている．しかし，限られた期間と工数の中で，あいまい性のない仕様を提示し，ユーザのイメージを喚起し有効なコメントを得ることが課題となっている．本稿では，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法と，その方法を効率的に行うための支援環境を提案する．本方法は，Webアプリケーションのドメインで典型的なシナリオのパターンと，そのシナリオを実現する典型的な画面構成を画面パターンとして用意する．これらのパターンを組み合わせて，ユースケースシナリオからプロトタイプを生成することで，短期間に整合性のとれたユースケースとプロトタイプを提示する．プロジェクト管理システムでの適用実験において，ユースケースシナリオとプロトタイピングによって，ユーザからの要求が従来方法より約3割多く得られたことを示し，特に，データ項目に関する要求については仕様が収束したことを示す．

A Method for Requirement Acquisition for Web Applications by Usecase Driven Prototyping

Ryota Mibe,[†1] Katsumi Kawai,[†1] Takuya Takeuchi,[†1]
Sadahiro Ishikawa[†2] and Yuji Fukushi[†2]

A Requirement Acquisition is one of the most important phase in system developments. They do it using documents, models and prototypes at user review. It is a problem how to propose specifications without ambiguity and to get good comments from users in limited time and cost. We propose a usecase driven prototyping method and support environment. In the method, we prepare scenario patterns for Web application domain and GUI patterns that are typical GUI implementations each scenario pattern. Developers can define usecase scenario as to select scenario patterns and to combine them. And they can generate prototypes from usecase scenario. We evaluated this method by adapting to develop a project management system. We show to get 30% more user comments, especially specification of data items converged.

1. はじめに

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[1),2)]．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」収集する方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．

(i) 自然言語と非形式的な図からなるドキュメント
(ii) UMLなどのモデリング言語で記述されたモデル
(iii) 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方法である．この方法は，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[2),3)]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．あいまい性を排除する試みとしては，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[4)]．

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[1)]．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

†1 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi, Ltd.
†2 株式会社日立製作所情報通信グループプロジェクトマネジメント統括推進本部
Information & Telecommunication Systems Project & Process Management Division, Hitachi, Ltd.

相互作用を表現する．また，ユースケースシナリオは，アプリケーションの利用者（以下，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[6]．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[7),8)]．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，データモデルや機能との関連づけを行う設計段階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 の記述方法は，ソフトウェアの専門家ではないユーザにとって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，結果的に自由な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[9),10)]．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，レビュー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，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[11)]．本稿の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，その結果から下流工程で活用できる仕様を作るところにある．

本稿では，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 1**）．

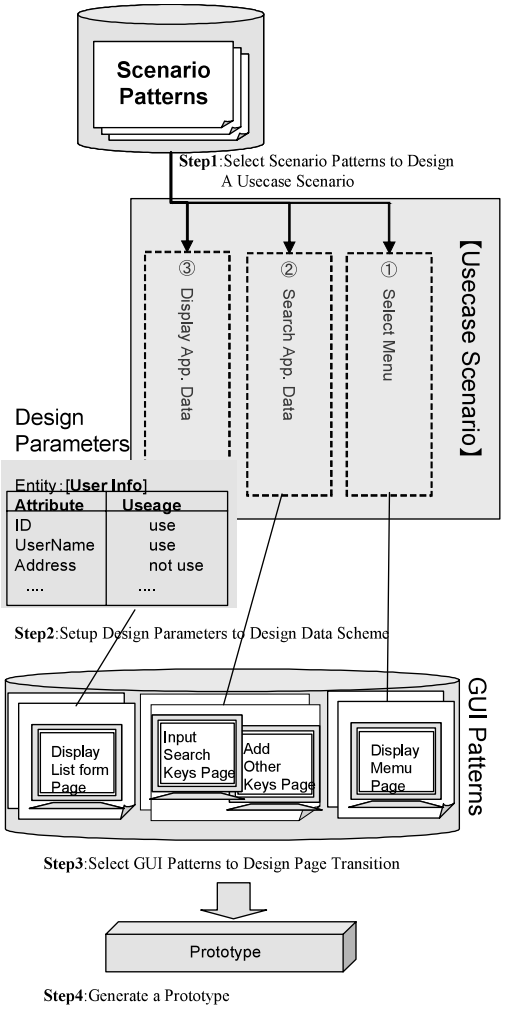

図 1　シナリオパターンと画面パターン
Fig. 1　Scenario patterns and GUI patterns.

（1）　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）することで定型化する（Scenario Patterns）．

(2)　(1) のパターンを組み合わせてシナリオを作成する（Step1）.
(3)　シナリオで使用するデータ項目（Design Parameters）を定義する（Step2）.
(4)　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）を選択する（Step3）.
(5)　(2)〜(4) の結果を用いてプロトタイプを生成する（Step4）.
(6)　(2)〜(5) 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．

ユースケースシナリオからプロトタイピングを生成する関連研究には，白銀らのものがある[12]．これは，ユーザの操作ステップ（入力，クリックなど）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アプローチである．

本稿の方法はこのアプローチと異なり，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．

以下 2 章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方法の基本コンセプトを，3 章では支援環境のアーキテクチャを，4 章では，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，5 章では実験の結果を，それぞれ説明する．

2. ユースケース駆動プロトタイプ環境

2.1 前提となる開発プロセス

本稿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 2** に示す．本開発プロセスは，開発を要求獲得フェーズ（Requirements Definition），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．

要求獲得フェーズでは，開発者はユースケースシナリオ，プロトタイプを作成する．これ

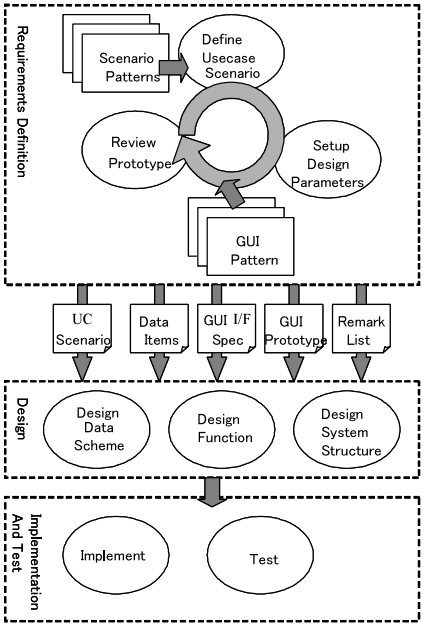

図 2　プロトタイプ利用型開発プロセス
Fig. 2　The development process using prototypes.

らをユーザに提示し，レビューすることで，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，システムの完成イメージを固めていく．

設計フェーズでは，要求獲得フェーズで得られた外部的な要求に対して，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，コンポーネント設計を行う．たとえば，要求獲得フェーズでは，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ることができるが，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・テストフェーズでは，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し，テストを行う．

本稿で提案する方法は，このプロトタイプ利用型開発プロセスを支援し，特に要求獲得

フェーズにおいて，ユーザの要求を早期に効率良く獲得することによって，後の設計フェーズ，実装・テストフェーズでの仕様変更を最小限にすることを目的とする．

2.2　要求獲得フェーズの要件

前節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を達成する必要がある．

(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，設計者のスキルや癖への依存性や，成果物間の矛盾を排除する．

(2)　ユーザのイメージを喚起できる情報を提示してレビューできる

　レビューで提示するユースケースシナリオとプロトタイプは，組み合わせることによってユーザに対して完成イメージを具体的にイメージできる必要がある．

(3)　本質的な仕様を中心にレビューできる

　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．下流工程でも修正が容易な仕様ではなく，データ項目や機能を中心にレビューを進められるようにする．

(4)　レビュー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 から 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 回のレビュー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，一般に複数の画面で実現されるものとする．

2.3　パターンによる定型化

前節の要件を達成するために，図 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．

シナリオパターンには，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．**図 3** で示すように，シナリオパターンを組み合わせて，対象システムのユースケースシナリオを作成する．ここでは『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となる．

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うえでアクセスするエンティティの属性，型を設定する．図 3 の例の 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様子を**図 4** に示す．ここで，ステップ 9 は単なるイベントを発生するだけなので，エンティティに対しての入出力は発生しない．

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

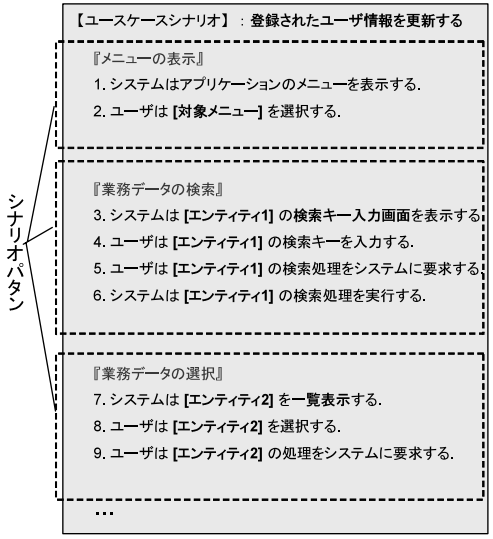

図 3　シナリオパターンの例
Fig. 3　Examples of scenario patterns.

	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	O	O	O				
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 4　設計パラメータの設定
Fig. 4　Setting up design parameters.

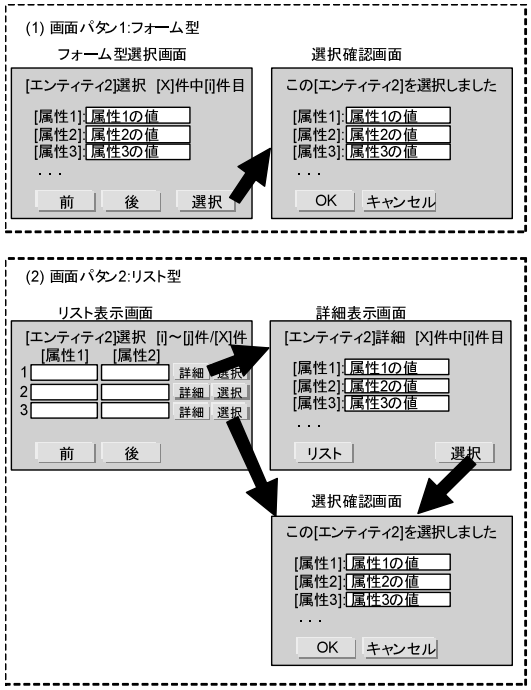

図 5　画面パターンの例
Fig. 5　Examples of GUI patterns.

張する仕掛けには<<extend>>や<<include>>がある[13]．これらは，ユースケースの単位で，共通部分をくくりだしたり，拡張部分を外だししたりすることは可能であるが，本稿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 つのシナリオは一般に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考えられるので，それぞれに対応する画面パターンを作成する．

たとえば，図 3 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**）．

(1)　1 レコードをフォーム形式でデータを表示し，その画面で処理対象を選択させる画面パターン
(2)　複数レコードをリスト形式でデータを表示し，フォーム形式の確認画面で詳細を確認した後に処理対象を確定させる画面パターン

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．

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存在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．

ここで，あらかじめ用意されたパターンを提示することで，本来のユーザの要求をゆがめてしまうのではないかという懸念がある．しかし，その一方，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．本稿では，要求をゆがめる危険性を回避するためには，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のやりたいことができるかどうかという視点でレビューを行うことで，ユーザの要求が獲得できるという立場をとる．

3. 支 援 環 境

3.1　支援環境の構成

前章で示した方法を支援するための開発支援環境を開発した（**図 6**）．開発支援環境は次の 3 つのコンポーネントからなる．次節で，コンポーネントの詳細を述べる．

(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も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．

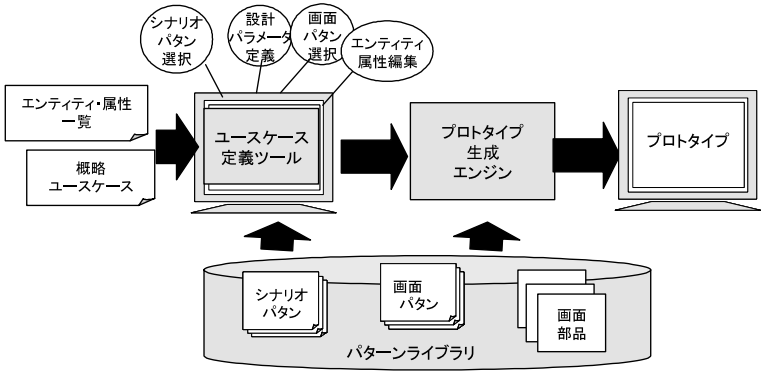

図 6　支援環境
Fig. 6　The support environment.

(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．

3.2　パターンライブラリ

開発支援環境では，あらかじめシナリオパターン，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．たとえば，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．

3.3　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 7**）．

ツールでは，以下の手順でユースケースを定義する．

(1)　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．
(2)　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，そのステップで扱う「エンティティ選択」を行う．
(3)　シナリオ詳細の各ステップに対して，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．
(4)　選択したシナリオパターンの「画面パターン定義」を行う．

その際，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応じた支援（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．たとえば，「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可能となる．

また，シナリオパターンと対応づけ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可能である．

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．

3.4　プロトタイプ生成エンジン

前節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

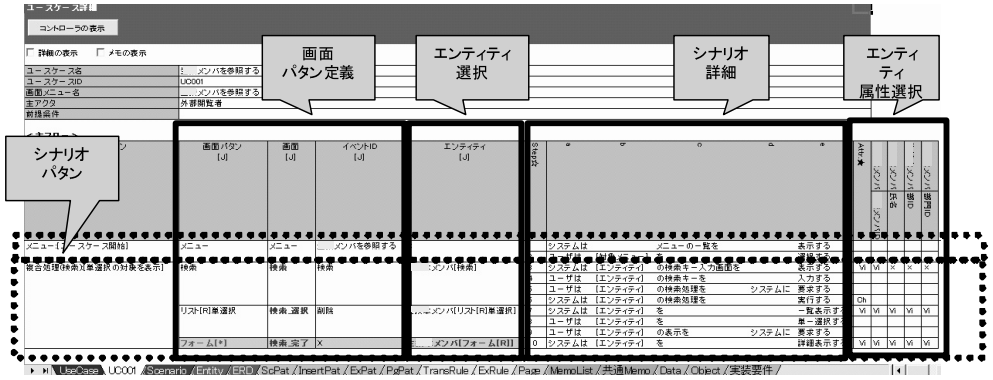

図 7　シナリオ作成ツール
Fig. 7　The tool for scenario writing.

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で，プロトタイプを実装のたた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．

4. 適 用 実 験

4.1　適用実験の対象システム

前章で示した，支援環境を用いて，実際のシステムを開発した．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．

4.2　開発プロセス

開発は 2.1 節で示したプロセスに従って行った．

(1)　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，レビュア（発注者）によるレビューを行い，レビュア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反映させる形式で合計 5 回レビューを繰り返した．レビューは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レビュー
(B)　画面構成を紙に印刷したもの（画面イメージ）を用いたレビュー
(C)　プロトタイプのレビュー

レビュアは，本システムの管理者，類似システム開発経験者，エンドユーザの 3 人に参加してもらった．レビューは，1 回あたり(A)〜(C)の合計で 2 時間かけて行った．

(2)　設計フェーズ

設計フェーズは，(1) 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，設計内容のレビューは行わなかった．

(3)　実装・テストフェーズ

設計フェーズで行った設計を基に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レビュアが 2 回レビューし，得られた指摘に対して修正を行った．このフェーズでのレビューは，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，あらかじめ下記をインプット情報として用意した．

- 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）

- 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
- 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. 実験結果

5.1 規模の推移

図 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．たとえば，「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レビューを示し，E1，E2 は実装・テストフェーズで行ったレビューを示す．当初 16 件あったユースケースが要求獲得フェーズでのレビュー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 で認証に関するユースケースを 1 件追加した．U2 では，「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，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，2 件，1 件のユースケースを追加した．

図 9 にテーブル数（エンティティ数），**図 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，そのフェーズ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 つ前のフェーズ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要望に関する指摘を受け追加した．

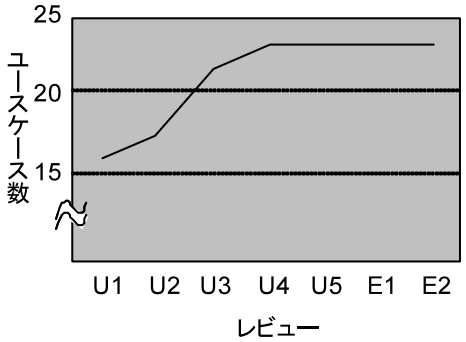

図 8　レビュー回数とユースケース数の推移
Fig. 8　The number of usecases for each review.

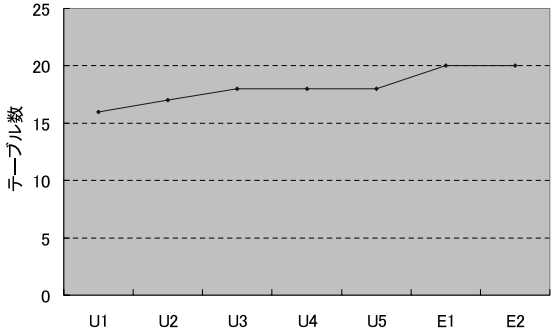

図 9　レビュー回数とテーブル数の推移
Fig. 9　The number of tables for each review.

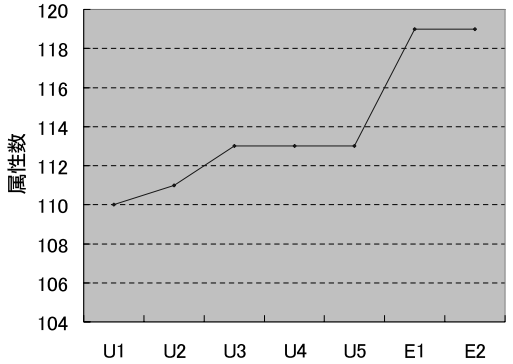

図 10　レビュー回数と属性数の推移
Fig. 10　The number of attributes for each review.

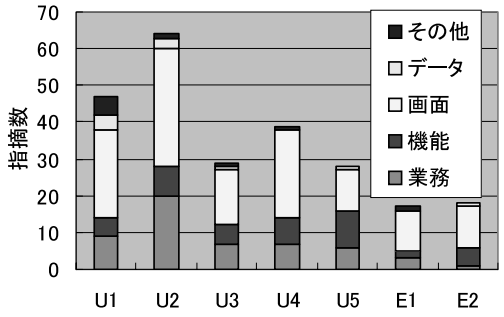

図 11　レビュー回数と指摘数の推移
Fig. 11　The number of pointing out for each review.

5.2　レビュー指摘数

図 11 に各レビューでの指摘内容の分布を示す．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．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，レビューを追うごとに収束していく様子が分かる．

図 12 に要求フェーズ（要求），実装・テストフェーズ（実装）でのレビュー指摘数を重要度別に示す．ここで，従来方法はユースケースと画面イメージによるレビューを想定し，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（約 27%．内重要な指摘 4 件）多くの指摘がなされている．

5.3　工　　数

レビュー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．

レビュー準備は，2 人で行った．要求獲得フェーズは 1 回のレビュー準備が平均 144 人時で，5 回分あわせて 721 人時かかった．

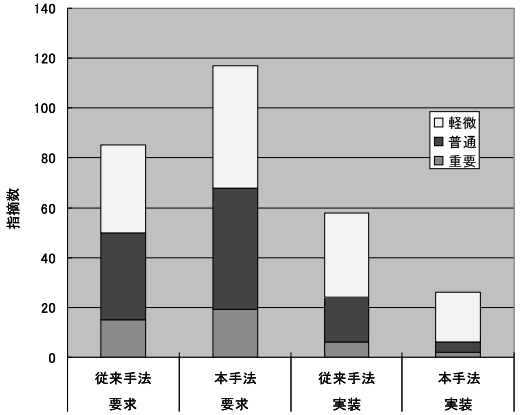

図 12　指摘重要度とフェーズ
Fig. 12　Degrees of importance and development phases.

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．

5.4　実験結果のまとめ

評価実験によって明らかになったことをまとめる．

(1)　**あいまい性**

本稿の方法では，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ほど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レビューすることにより，さらにあいまい性を排除できる．実験において実装・テストフェーズの指摘で仕様のあいまい性による誤解に起因するものは 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．

(2)　**イメージ喚起**

本稿で提案する方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ピングに対して，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．これは目論見どおり，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方法では，システム構築後になされる可能性が高い．

(3) **本質的議論への誘導**

一方，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レビューを進めていく際に，プロトタイプで生成できるレベルの画面仕様を中心にレビューを進め，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．その分限られたレビュー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．これにより当初目論んだ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．

(4) **プロトタイピングのコスト**

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，一般に両者を矛盾なくメンテナンスしていくことは多大な工数を要する．本稿では，ユースケースシナリオに画面パターンを対応づけることによりプロトタイプを生成する．このため，つね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，50 人時（約 1 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が大きい．

6. おわりに

本稿では，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法とその方法を効率的に行うための支援環境を提案した．本稿で提案する要求獲得方法では，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．本方法より，ユースケースとプロトタイプの一貫性を保ちながら，ユーザレビューを行える．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は，本稿で示す方法により，仕様が収束することを確認した．

今後の課題を以下に示す．実験により，本提案の実現性と，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，本稿では，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，設計展開する方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．

本稿では，機能的な要求を獲得する目的でパターンを設計した．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[14] 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[15),16)] 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．

参考文献

1) Leffingwell, D. and Widrig, D.: *Managing Software Requirements: A Unified Approach,* Addison-Wesley Pub. (1999).
2) IEEE Computer Society: IEEE Recommended Practice Software Requirements Specifications, IEEE Std830-1993, IEEE (1993).
3) 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).
4) 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).
5) Cockburn, A.: *Writing Effective Use Cases* (*Agile Software Development Series*), Addison-Wesley Pub. (2000).
6) 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).
7) Ridao, M., et al.: Domain independent regularities in scenarios, *Proc. 5th IEEE International Symposium on Requirements Engineering*, pp.120–127 (2001).
8) Watahiki, K. and Saeki, M.: Scenario patterns based on case grammar approach, *Proc. 5th IEEE International Symposium on Requirements Engineering*, pp.300–301 (2001).
9) Vonk, R.: *Prototyping: The Effective Use of Case Technology*, Prentice Hall (1990).
10) ACM SIGSOFT: Special Issue on Rapid Prototyping, *ACM SIGSOFT Software Engineering Notes*, Vol.7, No.5 (1982).
11) Snyder, C.（著），黒須正明（監訳）：ペーパープロトタイピング，オーム社 (2004).
12) 白銀純子，深澤良彰：ユースケースの縒合せによる GUI プロトタイプ生成手法，ソフトウェア工学の基礎ワークショップ（FOSE2001）(2001).
13) OBJECT MANAGEMENT GROUP: Unified Modeling Language: Superstructure. http://www.omg.org/technology/documents/formal/uml.htm

14) Mahemoff, M.J. and Johnston, L.J.: Principles for a Usability-Oriented Pattern Language, Calder, P. and Thomas, B. (Eds.), *Proc. OZCHI '98*, pp.132–139 (1998).
15) Tidwell, J.（著），ソシオメディア株式会社（監訳）：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).
16) Borchers, J.O.: A pattern approach to interaction design, *Proc. Conference on Designing Interactive Systems*, pp.369–378 (2000).

(平成 19 年 7 月 10 日受付)
(平成 20 年 1 月 8 日採録)

三部　良太（正会員）

1990 年電気通信大学電気通信学部卒業．1992 年東京工業大学大学院総合理工学研究科修士課程修了．同年（株）日立製作所システム開発研究所入社，ソフトウェア生産性の研究に従事．電気学会会員．

河合　克己

2000 年神戸大学工学部情報知能工学科卒業．2002 年名古屋大学大学院工学部計算理工学専攻修士課程修了．同年（株）日立製作所システム開発研究所入社，ソフトウェア生産性の研究に従事．

竹内　拓也

1999 年九州大学理学部卒業．2001 年九州大学大学院理学府修士課程修了．同年（株）日立製作所システム開発研究所入社，ソフトウェア生産性の研究に従事．

石川　貞裕

1984 年青山学院大学理工学部経営工学科卒業．同年（株）日立製作所入社．エンタープライズ系システムのアプリケーションアーキテクチャ，システム開発技術の整備に従事．

福士　有二

1990 年弘前大学理学部卒業．同年（株）日立製作所入社．アプリケーションソフト開発，およびソフトウェア生産技術の整備に従事．