プロダクトラインの開発と進化：パターン指向コアアセット改善手法

位野木 万里[†,††] 深澤 良彰[††]

プロダクトライン型の開発を最適な状態で継続するには，市場，技術，組織の変化に合わせて，コアアセットを進化させることが重要である．複数のプロダクトラインを保持する企業は，とくに組織全体として，コアアセットの最適な進化を考慮する必要がある．本稿では，組織で共有することを考慮した，コアアセットの改善手法を提案する．本手法では，進化のよりどころとなる標準を定義し，同標準に基づき，コアアセットと標準そのものを改善する．本手法の標準は，コアアセットの状態を計測するメトリクスとコアアセットの状態のタイプ，望ましい改善ノウハウを示したパターン，継続的な改善を定義したプロセスから構成する．

Software Product Line Development and Evolution: Pattern Oriented Core Asset *Kaizen* Method

Mari Inoki[†,††] and Yoshiaki Fukazawa[††]

Continuing optimal product line development needs to evolve core assets in response to market, technology or organization changes. In addition, a company which has several product lines needs to evolve all product lines as a whole for total optimization. In this paper, we propose a core asset evolution method which takes the standardization in a company into consideration. The important points of our method are to prepare a work standard and continue to improve core assets and the standard on the basis of the standard. Our core asset evolution method provides a standard that includes core asset types based on simple metrics, evolution patterns representing expertise, and evolution processes for continuous improvement.

1. はじめに

1.1 プロダクトライン型の開発とは

ソフトウェア開発の生産性の向上，品質の安定，開発リードタイムの短縮をねらい，プロダクトライン型の開発が注目されている[1),2)]．プロダクトライン型の開発では，製品開発のロードマップを描いたうえで，対象製品を開発するためのコアアセットと呼ばれるソフトウェア資産をあらかじめ整備し，これらのコアアセットを再利用し，製品を開発する．コアアセットは，ソフトウェア開発のための再利用資産であり，アーキテクチャ，コンポーネント，仕様書，テストケース，ツール等，ソフトウェア開発のライフサイクルで作成または利用される，あらゆる成果物が含まれる[3)]．

1.2 コアアセットの進化の重要性と課題

プロダクトラインが対象にしている市場，利用技術，組織は，時間の経過とともに変化する．また，コアアセットを利用した製品開発を通して，コアアセットへの修正や拡張の要望も蓄積される．コアアセットの使いやすさや再利用性の向上には，このような変化や要望に対応し，コアアセットを改善し，進化させることが重要である．

ところで，複数の事業を扱うソフトウェア開発企業は，複数のプロダクトラインを保持する．このような企業は，新規にプロダクトラインを構築しながら，すでに利用中のプロダクトラインを進化させることになり，とくにコアアセットの進化・改善の方針の共有が重要である．なぜならば，既存のプロダクトラインの技術や要求が変化したために，予想外にプロダクトラインの進化のコストが必要となれば，新規のプロダクトラインへの投資に影響を与え，組織全体のゴールをスムーズに達成できないリスクが高まるためである．よって，企業内でコアアセットの進化・改善の手法を標準として定め，共有することは重要である．

Clements らは，プロダクトライン型開発のための29個のプラクティスエリアと，これらを実際に使う

† 東芝ソリューション株式会社
Toshiba Solutions Corporation
†† 早稲田大学
Waseda University

ためのプラクティスパターンを提供した[3]．前者は，組織がプロダクトライン型開発に熟練するために必要な能力についての定義であり，後者は，組織の状況やゴールに従ってプラクティスエリアを選択する方法を示している．プラクティスエリアやプラクティスパターンは，様々な組織に適用可能な汎用化された形式で定義されている．しかし，実際の組織のねらいとコアアセットの状態に対応した，きめ細かいノウハウは明らかではない．また，組織全体の観点から，コアアセットを評価する仕組みは提供されていない．

たとえば，対象とする市場が縮小した場合には，コアアセットの保守コストを下げる目的で，利用頻度が低いと予想されるコアアセットをプロダクトラインから削除することがある．また，コアアセットの保守要員を，熟練者から新人に交替する予定がある場合，組織は，コアアセットを再構成し，新人でも使いやすい工夫を施す場合がある．さらに，組織全体の観点から，プロダクトラインの進化・改善の投資を，特定のプロダクトラインを対象に集中する場合もあれば，プロダクトライン群すべてに均等に投資する場合もある．これらのケースは，組織のねらいとコアアセットの状況に応じて変化するが，現状のプロダクトラインの研究では，そのための適切な方法は明らかにされていない．

1.3 研究のアプローチ

本稿では，プロダクトライン型開発の経験に基づき，組織全体の最適化を考慮した，コアアセットの改善手法を提案する．我々の手法では，望ましい改善のノウハウを組織の標準として定義し，同標準に基づき，コアアセットの改善計画を立案し，改善を実行する．標準は，コアアセットの状態を計測するメトリクスと，同メトリクスによるコアアセットのタイプの定義，改善ノウハウを示した改善パターン，継続的な改善を定義したプロセスから構成する．

本稿では，メトリクスは，組織がプロダクトラインを評価するための“ものさし”の役割である．改善パターンは，組織のねらいとコアアセットの状況を考慮した，コアアセットの改善方法の経験的な知識である．改善プロセスは，コアアセットの改善の手順である．組織の標準により，コアアセットの改善ノウハウを共有しても，時間の経過により，ノウハウの価値が変化する場合があるため，ノウハウそのものも継続的な改善が必要である．そこで，改善プロセスにより，実際の活動で得たノウハウを標準にフィードバックし，コアアセットと標準の双方を改善する．

1.4 コアアセット改善手法の基本要素

図 1 に，本稿で提案するコアアセットの改善手法の

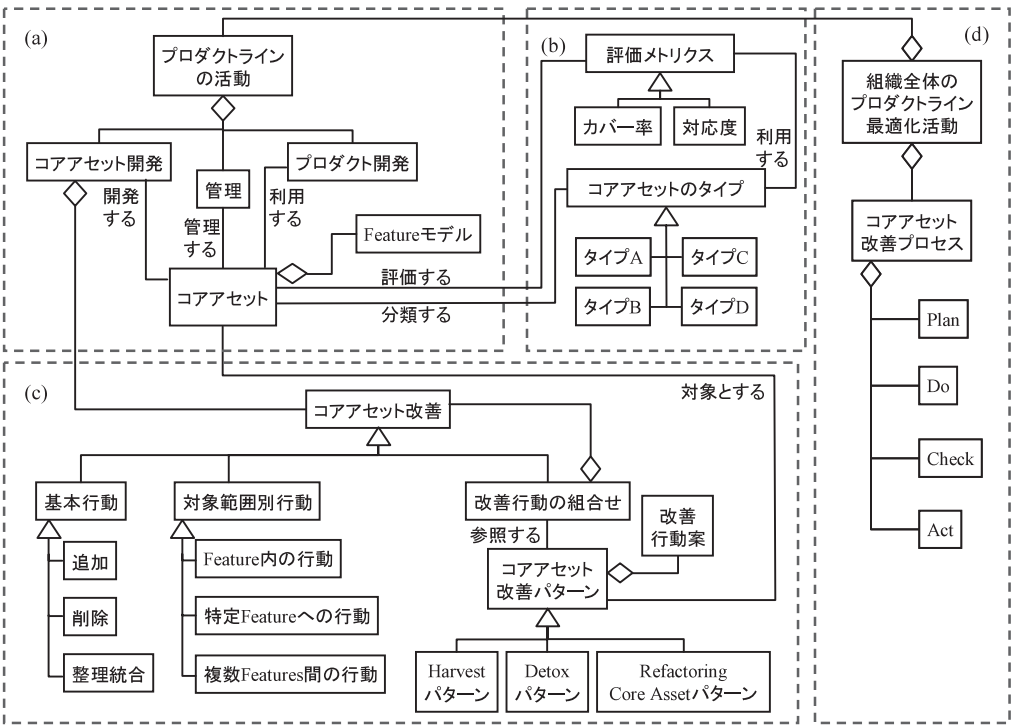


図 1 コアアセット改善手法の構成要素
Fig. 1 Elements of core asset *kaizen* method.

基本要素を，UML のクラス図により示す．

図 1 (b)〜(d) はコアアセット改善手法が提供する標準である．図 1 (a) は，プロダクトラインの活動が，コアアセットの開発，プロダクト（製品）の開発，コアアセットやプロダクトの管理から構成されることを示す[3]．また，コアアセットは Feature モデルを含む．Feature は，ユーザの視点で表現したドメインの特性であり，Feature を階層表現で記述した Feature モデルにより，ドメインの共通/可変性を明らかにする[4]．

図 1 (b) は，コアアセットに，評価メトリクスとコアアセットのタイプを導入することを示す．我々は，ドメインや利用技術に依存せずにコアアセットを評価するために，対応度とカバー率を定義し，コアアセットを 4 つのタイプに分類した．

図 1 (c) は，コアアセットの改善ノウハウに相当する．コアアセット改善行動をコアアセットの開発に位置づけ，さらに 2 つの観点から分類した．1 つは，基本的な行動の観点から，追加，削除，整理統合の行動に分類する．また，もう 1 つは，改善行動の範囲の観点から，Feature 内，特定 Feature，複数の Feature 間の行動に分類する．実際のコアアセットの改善活動は，基本的な行動と，対象範囲に関する行動の組合せとなる．我々の手法では，改善活動の経験から得た望ましい課題と解決方法の組合せを，改善パターンとして定義する．本稿では，Harvest，Detox，Refactoring Core Assets の 3 つのパターンを提案する．

図 1 (d) は，プロダクトラインの活動は，組織全体のプロダクトラインの最適化の活動の一部であり，継続的な改善活動は，Plan，Do，Check，Act のフェーズからなる PDCA サイクルに基づく改善プロセスによって定義されることを示す．

2. コアアセットのメトリクスとタイプ

複数のプロダクトラインを有する企業は，組織全体として，プロダクトライン型の開発を最適に維持するために，プロダクトラインの開発と進化のコストを考慮し，どのプロダクトラインへ投資をするかを判断する．そのためには，対象ドメインや利用技術の違いによらずに，コアアセットを評価する手段が必要である．そこで，プロダクトラインの開発経験に基づき，プロダクトラインの開発と進化のコストに影響するが，ドメインや条件に依存しないメトリクスとして，カバー率と対応度を定義し，これらに基づきコアアセットを 4 つのタイプに分類する（図 1 (b)）．

2.1 カバー率とは

カバー率の定義：

カバー率とは，どの程度の範囲の成果物を備えているのかという量を測るメトリクスのことである．カバー率は，組織の標準で定めたコアアセットの網羅項目数に対する，対象とするコアアセットが満たす項目数の割合により求める．

測定方法：

カバー率 =（コアアセットが満たす網羅項目数）
/（組織の標準で定めた網羅項目数）

組織の標準で定めた網羅項目数とは：

これは，成果物の種類の網羅項目数，成果物の種類別の網羅項目数の合計で構成される．

成果物の種類の網羅項目数とは，開発のライフサイクルで想定される工程と作業の分類別に，あらかじめ組織が定めた作成すべき成果物の種類数の合計とする．成果物の種類別の網羅項目数とは，上記で示した成果物の種類ごとに，ドメインの共通と特定製品に固有の成果物が準備され，製品開発に向けて成果物が網羅的かどうかを確認するための項目の数の合計とする．

たとえば，ソフトウェア開発の工程の例として，分析，設計，実装，テスト等がある．作業の分類の例として，データ，機能，環境，GUI 等がある．分析工程，データの作業の分類に対する，成果物の種類の例としては，概念データモデルの構造図と概念データモデルのデータ項目定義が考えられ，この場合，組織の標準で定めた成果物の種類数は 2 となる．その他の工程と作業の分類の組合せに対して，成果物の種類数を合計し，組織の標準で定めた成果物の種類を算出する．

成果物の種類として，概念データモデルの構造図と概念データモデルのデータ項目定義を定義したとする．これら 2 つの成果物の種類別の網羅項目として以下が考えられ，この場合の成果物の種類別の網羅項目数は 6 となる．分析工程，データの作業分類に関する成果物のみを組織が定めるコアアセットの種類とする場合は，組織の標準で定めた網羅項目数は，$2 + 6 = 8$ となる．

《成果物の種類別の網羅項目》

(a) 概念データモデルの構造図

(a-1) ドメインの共通データが，エンティティとして漏れなく構造図に定義されているか？

(a-2) 主要製品の固有データが，エンティティとして構造図に定義されているか？

(a-3) 開発完了/未定の製品の固有データが，エンティティとして構造図に定義されているか？

(b) 概念データモデルのデータ項目定義

(b-1) ドメインの共通データに対して漏れなくデータ項目仕様が作成されているか？
(b-2) 主要製品の固有データに対してデータ項目仕様が作成されているか？
(b-3) 開発完了/未定の製品の固有データに対してデータ項目仕様が作成されているか？

コアアセットが満たす網羅項目数とは：

これは，組織の標準で定めた網羅項目のうち，対象とするコアアセットが満たす網羅項目の数である．

分析工程，作業の分類がデータのコアアセットとして，概念データモデルの構造図のみを作成する場合には，コアアセットが備える成果物の種類数は 1 となる．その他の工程と作業の分類の組合せに対して，今回のコアアセットとして作成する成果物の種類数を合計し，コアアセットが備える成果物の種類数を算出する．

組織の標準で定めた網羅項目において，概念データモデルの構造図に関しては，上述した (a-1)〜(a-3) が設定されているとする．対象のコアアセットが，共通データについての概念データモデルの構造図を定義するが，特定製品に関する固有のデータについては定義しない場合には，(a-1) のみを満たすものとする．したがって，分析工程，作業の分類がデータについては，コアアセットが満たす網羅項目数は，成果物の種類数が 1，成果物の種類別の網羅項目数が 1 で合計 2 となる．組織の標準で定めた網羅項目数は 8 であるので，カバー率は，$2/8 = 0.25$ となる．

コアアセットは，あらゆる成果物が対象となる．しかし，現実のプロダクトライン開発では，投資対効果を考慮して，あらゆる成果物を作成することは困難であり，組織のねらいやプロジェクトの条件によって，カバー率を調整する必要がある．

2.2 対応度とは

対応度の定義：

対応度とは，コアアセットの成果物間で，どの程度の品質が考慮されているのかという質を測るメトリクスとする．対応度は，組織の標準で定めた品質確認項目数に対して，コアアセットの任意の要素間の対応関係が満たす，予想合格項目数の割合によって求める．

測定方法：

対応度 =（予想合格項目数）
/（対応関係に関する品質確認項目数）

対応関係に関する品質確認項目数とは：

これは，成果物の種類の任意の組合せに対する，品質に関する考慮すべき確認項目の数の合計とする．カバー率で示したように，組織の標準として，成果物の種類を定義しておくと，任意の複数の成果物間の関係に対して，品質確認のポイントをあらかじめ定義しておくことができる．

予想合格項目数とは：

予想合格項目数は，上記の対応関係に関する品質確認項目のうち，対象とするコアアセットで合格することを目標とする確認項目の数を表す．

対応度の算出例：

組織の標準で定めた成果物の種類として，設計工程，機能の作業の分類の設計モデルの構造（クラス図）と設計モデルの振舞い（シーケンス図），実装工程，機能の作業の分類のコンポーネントを取り上げる．

これらの 3 つの成果物の種類間での品質確認項目として以下が考えられ，この場合の品質確認項目数は 3 となる．

- 設計モデルの構造の仕様に基づいて，コンポーネントが実装されているか？
- 設計モデルの振舞いの仕様に基づいて，コンポーネントが実装されているか？
- 設計モデルの振舞いには，コンポーネントに実装された例外処理も仕様化されているか？

上記の 3 つの品質確認項目を組織が設定していると仮定する．注文管理ドメインのコアアセットが，注文処理の設計モデルと注文処理のコンポーネントを含む場合を取り上げ，次の 2 つのケースを考える．

ケース 1：注文処理の設計モデルに基づいて，注文処理のコンポーネントの検証が完了していない．たとえば，コンポーネントのある例外処理が，設計モデル上では実際の振舞いとは異なる仕様で記述されている．

ケース 2：注文処理の設計モデルに基づいて，注文処理のコンポーネントの検証が完了している．設計モデルの構造と振舞いの仕様に基づいて，コンポーネントが実装されており，コンポーネントの振舞いは，例外処理も含めて設計モデル上に仕様化されている．

設計工程，実装工程，機能の作業の分類の成果物間の対応関係に関する，品質確認項目の予想合格項目数，対応度は下記のようになる．

ケース 1：予想合格項目数 0
対応度 $= 0/3 = 0$

ケース 2：予想合格項目数 3
対応度 $= 3/3 = 1$

上記の 2 つのケースを比較すると，ケース 2 の対応度が高くなる．コアアセット内で高い整合性を保持するには，高コストを要する．組織のねらいやプロジェクトの条件により，整合性の度合いを調整する必要がある．

表 1 コアアセットのタイプ
Table 1 Types of core assets.

カバー率＼対応度	成果物間の対応がとれている（基準値との比較で2つに分類）	成果物間の対応は一部のみ（基準値との比較で2つに分類）
開発ライフサイクルで利用する成果物を多数含む（基準値との比較で2つに分類）	タイプA	タイプB
開発ライフサイクルで利用する成果物を一部含む（基準値との比較で2つに分類）	タイプC	タイプD

2.3 コアアセットのタイプ

カバー率と対応度に基準値を設定し，その基準値の前後で 2 つの範囲を設定すると，コアアセットの状態は次の 4 タイプに分類できる．タイプ間の関係を**表 1** に示す．

(1) タイプ A：上流から下流までの一連の成果物を備え，各々整合がとれた状態．投資に対し効果が期待でき，組織の強みとなるドメイン向けのプロダクトラインが該当する．

(2) タイプ B：上流から下流までの一連の成果物を備えるが，成果物間の整理統合が不完全な状態．すでに複数のアプリケーションの開発経験があり，様々な成果物が蓄積されているが，製品の利益率が低い等の理由により，プロダクトライン構築への高額の投資の決定が困難なドメイン向けのプロダクトラインが該当する．

(3) タイプ C：作成する成果物は限定するものの，成果物間の整合がとれている状態．新規参入スピードが必要なドメインに対するプロダクトラインが該当する．

(4) タイプ D：最低限のコンポーネントまたは設計仕様等が蓄積されているだけの状態．新規参入で，組織の強みとするかどうか，不確定要素の多いドメイン向けのプロダクトラインが該当する．

3. コアアセットの改善行動

3.1 改善行動の種類

コアアセットの改善行動を，基本行動と対象範囲別の行動に分類する（図 1 (c)）．基本行動は，2 章で定義したカバー率と対応度に基づくと，コアアセットへの要素の追加，削除，整理統合に分類できる（**表 2**）．これらは，順番に，コアアセットのカバー率を高くする/低くする，対応度を高くする行動である．

ところで，カバー率を低くする目的は，肥大化したコアアセットの保守コスト等を下げることと考えられる．なお，対応度を低くする改善行動はない．しかし，カバー率を高くするために，コアアセットを追加した結果，対応度が下がる場合がある．

また，改善のコストには，改善の範囲が影響する．そのため，改善の対象範囲の観点から，コアアセットの改善行動を定義する．本手法では，Feature に着目し，Feature 内，特定 Feature，複数の Feature 間に分類する．以下では，改善の範囲を考慮し，基本行動を定義する．

表 2 コアアセット改善行動の種類
Table 2 Kinds of activities for core asset *kaizen*.

行動の目的	行動
カバー率を高くする	追加
カバー率を低くする	削除
対応度を高くする	整理統合
対応度を低くする	該当なし

3.2 コアアセットの追加

コアアセットの追加とは，コアアセットに新たに要素を補充することである．組織は，特定製品向けに開発した要素を，他の製品で利用可能なものに汎化するか，コアアセット改善計画に基づき，新規に開発する．コアアセットの追加方法の例を下記に示す．

(1) Feature 内のコアアセットの追加
これは，既存の Feature を実現するための新たなコアアセットを補う行動のことである．たとえば，顧客の登録管理 Feature に対するビジネスコンポーネントがすでにコアアセットに含まれる場合に，本 Feature に対するテスト環境を新たに追加する．

(2) 特定の Feature の新規追加
これは，新規 Feature と同 Feature を実現するコアアセット一式を追加する行動のことである．たとえば，顧客管理ドメインにおいて，新規に顧客の購買履歴の統計分析 Feature と，同 Feature の設計モデル，コンポーネント，テストモデルを追加する．

(3) 複数 Feature 間に関連するコアアセットの追加
これは，複数の既存の Feature 群に関連する，コアアセットを追加する行動のことである．たとえば，顧客管理，販売管理，在庫管理の Feature 群に対して，共通して利用する開発・実行環境を追加する．

3.3 コアアセットの整理統合

コアアセットの整理統合は，コアアセットがカバーする範囲や役割の変更なく，コアアセットを手直しすることであり，ソースコードのリファクタリング（Refactoring）と類似の行動である[5]．整理統合には，コアアセットの品質を高め，利用しやすくする行動も含まれる．コアアセットの整理統合の例を以下に示す．

(1) Feature 内のコアアセットの整理統合
これは，既存の Feature の特定のコアアセットに対して，不具合修正や，利用者の使いやすさを向上させるため，表現を改善する行動のことである．たとえば，業務ロジックの仕様を，自然言語による記述から，UML のダイアグラム（アクティビティ図，クラス図，シーケンス図等）に改める．

(2) 特定の Feature のコアアセットの整理統合
これは，既存の Feature を実現するためのコアアセット群の間で，整理統合を行う行動のことである．たとえば，特定の Feature の設計モデルのシーケンス図上に表現されたメッセージ名と，設計モデルのクラス図上の操作名を対応づけることや，ソースコードと対応づけられるように設計モデルの可読性を高める．

(3) 複数 Feature に関連するコアアセットの整理統合
これは，複数の Feature 群に関連するコアアセットを整理統合する行動のことである．たとえば，Feature の追加や削除にともなって，システム総合テストに関するテストモデルを改訂する．

3.4 コアアセットの削除

コアアセットの削除とは，既存のコアアセットから類似または利用されていない成果物を排除し，コアアセットを精錬することである．コアアセットの削除方法の例を以下に示す．

(1) Feature 内のコアアセットの削除
これは，特定の Feature を実現するコアアセットの一部を削除する行動のことである．たとえば，Feature を実現するコンポーネントが，類似した多数の検証シナリオを備える場合，代表的な検証シナリオに限定し，残りは削除する．

(2) 特定の Feature そのものの削除
これは，特定の Feature を実現するコアアセット一式を削除する行動のことである．たとえば，大規模ユーザ向けに用意していた Feature を，市場ニーズの減少により削除する．

(3) 複数の Feature に関連するコアアセットの削除
これは，複数の Feature 群に関連するコアアセットを削除する行動のことである．たとえば，ある機能の検証環境等を，プラットフォームの陳腐化等を理由に削除する．

4. コアアセット改善パターン

パターンは，課題と解決策を組にして知識を表現する方法であり，ソフトウェア開発の様々な分野で活用されている[6]．我々のコアアセット改善手法では，コアアセットの改善ノウハウを，パターンの形式で蓄積し（図 1 (c)），コアアセットの改善計画立案に利用する．

以下，改善パターンの記述形式を定義し，改善パターンの例を示す．なお，4.1〜4.3 節の各パターンの改善行動案には，その一部を示し，詳細付録の**表 8** に示す．

改善パターンの記述形式

- **背景：**組織の状況
- **課題：**組織が直面する改善が必要とされる問題
- **ねらい：**コアアセットの改善の目的
- **解決策：**カバー率と対応度を用いた改善の方針
- **改善行動案：**解決策に基づく改善のシナリオ
- **変形：**上記の背景〜改善行動案を基本記述とし，上記以外で想定可能な背景や課題等の変形．

4.1 Harvest（収穫）パターン

背景：
コアアセットを利用して製品開発を進めてきたが，複数の製品開発の経験から蓄積された業務ノウハウが，コアアセットとして可視化されていない．

課題：
複数の製品向けに毎回類似した追加が発生しており，価格競争力が低下している．

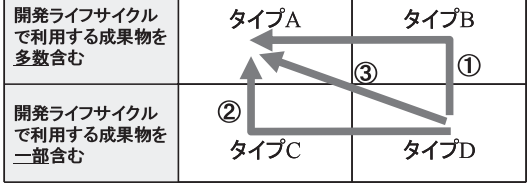
ねらい：
製品の開発コスト削減およびリードタイムの短縮．

解決策：
製品開発で蓄積された業務ノウハウを収穫（Harvest）し，コアアセットに追加し，コアアセットのカバー率を上げ，製品開発で作成する部分を削減する．単純にコアアセットを増加させただけでは，既存のコアアセットとの整合性，コアアセットの使いやすさ等が低下し，製品の品質劣化を招くリスクがある．したがって，コアアセットの改善行動の方針では，カバー率と対応度の両方の向上が重要である．

改善行動案：
現状のコアアセットがタイプ D の場合，タイプ A に向けた改善が望ましい．**表 3** に，タイプ D の場合の改善行動のシナリオを示す．

表 3 コアアセット改善のシナリオ (1)
Table 3 Scenario for core asset *kaizen* I.

カバー率＼対応度	成果物間の対応がとれている	成果物間の対応は一部のみ
開発ライフサイクルで利用する成果物を多数含む	タイプA	タイプB
開発ライフサイクルで利用する成果物を一部含む	タイプC	タイプD



カバー率と対応度の向上には，①〜③ のシナリオが考えられる．

① コアアセットの追加 → 整理統合

今後の製品開発で再利用されるコアアセットを追加した後，他のコアアセットとの整合をとる．

② コアアセットの整理統合 → 追加

既存のコアアセットの要素の整合をとり，整理統合した後，新規にコアアセットを追加する．

③ 整理統合されたコアアセットを追加

当初タイプ A のコアアセットを想定していたが，投資費用が確保できず，一部の断片的なコアアセットのみ開発し，タイプ D となった場合等において，同計画にのっとった整理統合された成果物を追加する．

①〜③ の選択は，投資対効果（ROI）分析，リスク分析等の結果に依存する．たとえば，タイプ D は，対応がとれていない状態のため，タイプ A への改善に向けては，① のコアアセットの追加を先に実行するよりも，② の整理統合後，コアアセットを追加する方が，リスクが低い．整理統合されていない状態にコアアセットを追加すると，コアアセットの対応度が低下し，コアアセットを利用した製品開発では，テストのコストが増加する．また，タイプ D は，対応度が低いため，タイプ A へ一足飛びの改善は困難であり，③ のシナリオのように，適切な改善計画が整備されている場合は稀である．したがって，② のシナリオが安全である．

変形：

新規のプロダクトの要求が発生した場合も，本パターンは適用可能と考えられる．ただし，開発するコアアセットが，別のプロダクトにも適用可能であると判断できることが条件である．ドメイン分析によって必要と考えられるコアアセットを開発し，新規に追加することになる．

4.2 Detox（浄化）パターン

背景：

長年にわたりコアアセットを管理してきた．製品開発で利用されないコアアセットが存在している．

課題：

長年の利用にともない，コアアセットが肥大化し，製品開発において，適切なコアアセットの選択が困難になり，さらに，仕様変更対応時，膨大なテストを実行する必要がある等，保守コストが増加している．

ねらい：

保守コスト削減．

表 4 コアアセット改善のシナリオ (2)
Table 4 Scenario for core asset *kaizen* II.

カバー率＼対応度	成果物間の対応がとれている	成果物間の対応は一部のみ
開発ライフサイクルで利用する成果物を多数含む	タイプA	タイプB
開発ライフサイクルで利用する成果物を一部含む	タイプC	タイプD

解決策：

不必要で製品開発に問題をもたらすコアアセットを浄化（Detox）し，製品開発時のコアアセットの選択をしやすくし，保守コストを下げる．その結果として，コアアセットのカバー率は下がる．コアアセットの削除後，残されたコアアセット内の整合性と，コアアセットの使いやすさの向上が重要である．

改善行動案：

現状がタイプ B の場合，タイプ C に向けた改善が望ましい．**表 4** に，タイプ B の場合の改善行動のシナリオを示す．

カバー率を下げ，対応度を上げるには，①〜③ のシナリオが考えられる．

① コアアセットの削除 → 整理統合

製品開発では不必要なコアアセットを削除し，その後，残されたコアアセット間で整合をとる．

② コアアセットの整理統合 → 削除

必要なコアアセットを識別し，それらの整合をとり，不必要なコアアセットを削除する．

③ 整理統合されたコアアセットの削除

あらかじめ計画していた改善計画にのっとって，整理統合された成果物を削除する．

①〜③ の選択は，ROI 分析，リスク分析等の結果に依存する．たとえば，タイプ B は，対応度が低い状態である．4.1 節と同様に，はじめにコアアセットを削除するやり方よりも，整理統合により，不必要なコアアセットを明確にし，影響範囲等を整理したうえで，削除を実行する ② のシナリオが，製品開発時の品質劣化のリスクを低減させるために安全である．

変形：

なし．

4.3 Refactoring Core Assets（コアアセットのリファクタリング）パターン

背景：

コアアセットの品質や再利用率等に問題はあるもの

表 5 コアアセット改善のシナリオ (3)
Table 5 Scenario for core asset *kaizen* III.

カバー率＼対応度	成果物間の対応がとれている	成果物間の対応は一部のみ
開発ライフサイクルで利用する成果物を多数含む	① タイプA ② タイプB	
開発ライフサイクルで利用する成果物を一部含む	タイプC	タイプD

の，開発メンバの努力によってコアアセットを利用した製品開発が行われている．

課題：

組織変更による開発メンバの刷新を予定している．現状のコアアセットに精通していない新メンバが，製品の開発が可能かどうかが懸案事項である．

ねらい：

コアアセットの品質（使いやすさ）の向上．

解決策：

コアアセットを整理統合し，コアアセットの可読性，使用性を向上させる．

改善行動案：

表 5 に示す下記の ① ② の 2 つのシナリオが考えられる．現実の組織では，様々な変化をふまえた適切な改善計画が存在しているケースは少なく，① のシナリオの実行は困難である．よって，まず，整理統合が必要な箇所を特定し，改善計画を明らかにしたうえで，コアアセットを整理統合する ② のシナリオが妥当である．

① 改善計画に従った整理統合

あらかじめ計画していた改善計画にのっとって，コアアセットの整理統合を行う．

② 整理統合 → 整理統合

最初の整理統合で，整理統合する箇所を特定し，改善計画を立案する．続いて，計画に基づいて，成果物間の整合性を高める．

変形：

なし．

5. コアアセット改善プロセス

4 章において，望ましいコアアセットの改善行動ノウハウをパターンで定義し，これらを組織で共有することを示した．しかし，このようなノウハウは，組織にとって完全である保証はなく，ノウハウそのものも継続的に改善することが重要である．そこで，図 1 (d) で示したように，我々の提案するコアアセット改善手法では，Plan-Do-Check-Act（PDCA）サイクルの繰返しによって，コアアセットおよび，コアアセットの改善ノウハウを集約した標準そのものも改善する．

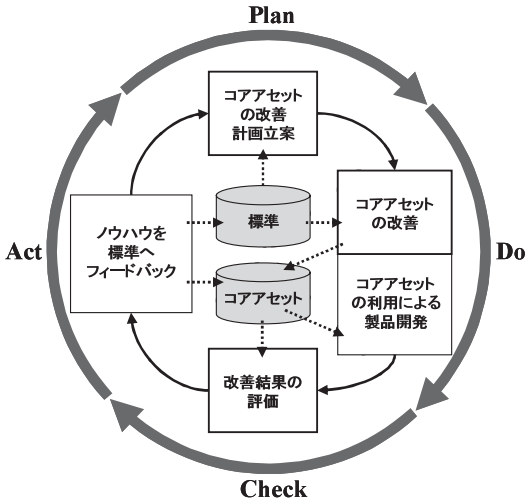


図 2 コアアセットの改善プロセス
Fig. 2 Processes for core asset *kaizen*.

PDCA サイクルによるコアアセットの改善プロセスは次のようになる（**図 2**）．

Plan フェーズ：

(a) 組織は，組織全体としてのコアアセット改善計画を立案する．

(b) (a) において，改善が決定したプロダクトラインを担当する組織は，次の手順により改善計画を立案する．

(b-1) 対象コアアセットの対応度とカバー率を特定し，現状のタイプを決定する．

(b-2) めざすタイプを決定する．

(b-3) 改善パターンを参照し，現状とめざすタイプから，改善行動案を選択する．

(b-4) 改善行動案のシナリオを分析し，分析結果から，改善行動のシナリオを決定する．

(b-5) 改善行動のシナリオに基づき，改善の範囲や年度別等での改善計画を立案する．

Do フェーズ：

組織全体のプロダクトライン改善計画と，さらに Plan フェーズでの各プロダクトラインのコアアセットの改善計画に基づき，プロダクトラインの担当組織は，改善，すなわち，コアアセットの追加，削除，整理統合を行う．また，並行して，改善されたコアアセットを用いて，製品を開発する．

Check フェーズ：

コアアセットの改善を実行した結果，改善したコア

アセットを利用して製品開発を行った結果，組織全体としてプロダクトライン群の改善の結果を分析・評価する．標準として蓄積していた改善パターンや，タイプの考え方と，実行結果との差分を抽出し，新規に共有すべきノウハウの候補を洗い出す．

Act フェーズ：

Check フェーズでの分析結果に基づき，得られた新たなノウハウを，コアアセットまたは組織の標準へフィードバックする．また，新たな改善計画の立案に進み，継続的に改善を行う．

6. 適 用 例

コアアセットの改善をめざす組織を想定し，改善パターンを適用した改善計画立案の試行例を示す．本章での改善手順および各データは，実際のプロダクトライン開発案件を対象に，関係者へインタビュを行い，その結果に基づいて著者らが作成した．

6.1 ドメインの説明

対象ドメインは，特定分野向けの大規模な情報編集支援システムである．市場には，約 20 年前から参入し，複数の顧客に製品を導入した．コアアセットは，合計で数百万行のコード量のコンポーネントを含む．顧客は，従来は手作業で情報を編集していたため，顧客ごとにやり方が異なる．コアアセットを開発する組織は，顧客要望を反映するため，機能拡張を続けてきた．

長期にわたるプロダクトラインの維持の間，技術や顧客の要望の変化にともない，問題が発生している．たとえば，利用されなくなったコアアセットが，製品に含まれる場合があり，製品の保守で無駄なコストが発生している．また，コアアセットには，複数の類似するコンポーネントが蓄積され，製品開発時に，適切なコンポーネントを選択することが困難となっている．

6.2 改善パターンの選択と適用手順例

組織全体として，当該プロダクトラインの改善への投資を決定したとする．以下，改善パターンによる，当該プロダクトラインのコアアセットの改善計画の立案手順を示す．組織の標準で定めた網羅項目数，対応関係に関する品質確認項目数，対応度，カバー率の基準値は組織の標準において定義されているものとする．

(1) 改善パターンの選択とシナリオの絞り込み

コアアセットの現状とめざす姿のタイプを明らかにし，改善パターンを選択しシナリオを絞り込む．組織の標準で定めた網羅項目に対し，対象とするコアアセットが満たす網羅項目を確認すると，本事例のカバー率は組織で定めた基準値よりも高くなった．その理由は，本事例では，製品開発のつど，コアアセットを拡充し，各工程別の成果物の種類が揃っていたこと，成果物の種類別にもドメインの共通と，開発完了済みの製品も含む主要製品に固有の成果物が網羅的に作成されていたこと等による．

しかし，対応関係に関する品質確認項目数に対して，対象コアアセットは，コアアセット間の整合がとれていない項目が多く，対応度は基準値よりも低くなった．よってコアアセットの現状はタイプ B となった．組織の方針検討により，利用されなくなった特定製品に固有のコアアセットをスリム化し，保守コストを削減したいため，めざす姿はタイプ C とした．

このケースでは肥大化したコアアセットをスリム化したいことと，パターンの背景と課題の合致性から，Detox パターンが選択された．改善行動案により，タイプ B からタイプ C へは，3 つのシナリオが考えられるが，現状，肥大化したプロダクトラインであるため，③ のシナリオは適用困難である．そこで，① 削除 → 整理統合のシナリオと，② 整理統合 → 削除のシナリオが候補となる．Detox パターンでは，② のシナリオが，開発リスクが少ないとしている．ROI 分析によって，Detox することの効果，Detox が妥当な場合には ① と ② のシナリオの効果を確認する．

(2) 改善行動のシナリオの効果分析

Böckle らの方式[7] を応用し，下記により効果を分析した．

コアアセット改善の効果

$$= \sum C_{old\_way} - \sum C_{new\_way} - C_{del} - C_{ref}$$

$\left(\sum \text{は，製品数の総和}\right)$

C_{old_way}：改善前コアアセットで製品を開発するコスト

C_{new_way}：改善後コアアセットで製品を開発するコスト

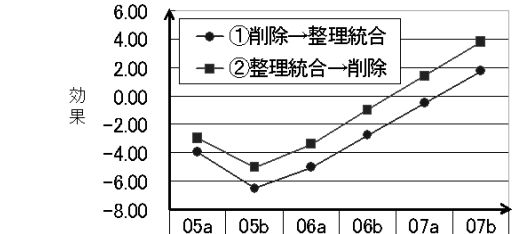
C_{del}：コアアセットを削除するコスト

C_{ref}：コアアセットを整理統合するコスト

表 6 に，① 削除 → 整理統合のシナリオを例にとり，上記の計算式の効果を半期ごとに算出するスプレッドシート例を示す．表 6 は，削除と整理統合の進化・改

表 6 効果見積りのためのスプレッドシート例
Table 6 Example of spreadsheet for estimating effect.

	05a	05b	06a	06b	07a	07b	計
プロダクト数	2	2	2	3	3	3	15
$\Sigma C_{old\ way}$	24.50	24.50	24.50	36.75	36.75	36.75	183.75
$\Sigma C_{new\ way}$	24.50	25.00	23.00	34.50	34.50	34.50	176.00
C_{del}	4.00	0.00	0.00	0.00	0.00	0.00	4.00
C_{ref}	0.00	2.00	0.00	0.00	0.00	0.00	2.00
効果	−4.00	−2.50	1.50	2.25	2.25	2.25	1.75



	05a	05b	06a	06b	07a	07b
①削除→整理統合	-4.00	-6.50	-5.00	-2.75	-0.50	1.75
②整理統合→削除	-3.00	-5.00	-3.40	-1.00	1.40	3.80

図 3 2 つのシナリオの効果見積り累計例
Fig. 3 Example of cumulative effects for two scenarios.

善に投資できる費用が 6.00 かつ，05 年度にのみ投資実行するという条件の下，① 削除 → 整理統合のシナリオを実行する効果見積りの一例を示したものである．実際には，削除と整理統合の割合を様々に変えて，検討する試行錯誤が必要であり，表 6 は，そのような試行錯誤を経て得た効果見積り案の一例である．

表 6 と同様にして，② 整理統合 → 削除のシナリオの効果も算出する．**図 3** には，2 つのシナリオの効果の累計を示す．図 3 において，横軸は半期ごとの時間を示し，縦軸は表 6 に示した効果の累計を示す．効果の単位は設定せず，相対値とした．投資額累計は両シナリオで同一とした．

図 3 に示すように，両シナリオとも 07b の時点で効果はプラスとなるため，Detox は有効と見なせる．ただし，組織が，07b では時期が遅いと見なせば，Detox は有効ではないと判断することもある．07b の回復で問題ないと判断した後，① および ② を比較する．効果額がプラスとなる時期は，② の方が若干早く，効果の値も ② の方が大きい．① のシナリオの場合，整理統合前の製品の開発では，削除にともなう影響範囲の確認のため，検証コストを通常より多く見積もることが安全である．また，削除によって，コアアセットの利用のコストが余分にかかるリスクを見込む必要がある．したがって，不確定部分が多くなることから，先に整理統合を実行することが安全である．

C_{old_way} および C_{new_way} の見積りでは，作業量を WBS（Work Breakdown Structure）のコストの総和から算出した．WBS は，列：データ系，ロジック系，環境系，GUI 系，行：分析，変更修正特定，変更修正実行，検証を設定し，合計 16 個に分けた．

図 3 の 05b のシナリオ ① の場合の C_{old_way} および C_{new_way} の分析例が**表 7** である．表 7 は表 6 の効果見積り案を作成する試行錯誤の過程で得られたデータの一例である．改善パターンを適用する場合，改善の前後の作業工数を見積もる必要がある．表 7 は，

表 7 開発コストの見積り例
Table 7 Example of estimation of development costs.

(1) 改善前（現状）の作業量見積り

開発作業	Data	Logic	環境	GUI	計
分析（仕様閲覧，理解）	0.25	0.25	0.25	0.5	1.25
変更修正特定（ソース閲覧，理解）	0.5	0.5	0.5	1	2.5
変更修正実行（修正実行）	0.5	1.5	0.5	2	4.5
検証（テスト）	0.5	1	0.5	2	4
合計	1.75	3.25	1.75	5.5	12.25

(a) (d)

(2) 改善前（現状）に対する割合

開発作業	Data	Logic	環境	GUI
分析（仕様閲覧，理解）	0.9	0.9	1	0.8
変更修正特定（ソース閲覧，理解）	0.9	0.9	1	0.8
変更修正実行（修正実行）	1	1	1	1
検証（テスト）	1.2	1.2	1	1.2

(b)

(3) 改善後の作業量見積り

開発作業	Data	Logic	環境	GUI	計
分析（仕様閲覧，理解）	0.225	0.225	0.25	0.4	1.1
変更修正特定（ソース閲覧，理解）	0.45	0.45	0.5	0.8	2.2
変更修正実行（修正実行）	0.5	1.5	0.5	2	4.5
検証（テスト）	0.6	1.2	0.5	2.4	4.7
合計	1.775	3.375	1.75	5.6	12.5

(e) (c)

WBS 単位による見積り方法を一例として示したものである．

表 7 (1) は，C_{old_way} の見積りである．コストは表 7 (d) となる．表 7 (2) はコアアセットの削除後の各 WBS 作業の割合予測である．① の場合，整理統合せずにコアアセットを削除した場合，データ，ロジック，GUI の検証コストは，1.2 倍になると見積もった．表 7 (3) は C_{new_way} の見積りである．各 WBS の改善後のコスト (c) は，下記の式から得る．

改善前の作業コスト (a) ∗ 作業割合 (b)

C_{new_way} は，表 7 (3) の 16 の WBS の和 (e) である．

(3) 改善行動の方針

選択するシナリオは ② 整理統合 → 削除とする．ところで，コアアセットの削除方法の例によれば，3.4 節の 3 案がある．本事例の場合は，3.4 節 (2)「特定の Feature そのものの削除」に限定することが望ましいと考えられる．(1)，(3) も含めた様々な削除を行うには，事前に十分に整理統合する必要があるが，投資額に限度があり，対応困難なためである．また，対象コアアセットには，不必要なコアアセットが存在する徴候があり，利用されなくなった Feature の識別に限定すれば，整理統合も削除も効率的である．整理統合により，使用されていない Feature を特定し，独立性を高める改善を行い，その後，同 Feature を削除する方針を設定する．

6.3 標準へのフィードバック例

コアアセットの計画立案ならびに改善の実行後，PDCA サイクルに基づいて，コアアセットの改善手法の標準を改善する．なお，実際のコアアセットの改善，製品開発での結果のフィードバックは，本試行の

範囲外とする．

今回の改善計画立案では，次のようなノウハウが得られたので，標準へフィードバックできる．

– コアアセットを削除する場合，リスクを回避して効率的に改善するには，改善の範囲を特定の Feature から始める（ノウハウの追加箇所：Detox パターンの改善行動案）．
– 改善コスト見積りには WBS による分割が有効である（ノウハウの追加箇所：改善プロセス）．
– 整理統合せずにコアアセットを削除すると，リスク回避のため，検証コストは，20%以上多く見積もる（ノウハウの追加箇所：改善プロセス）．

7. 考　　察

適用例に基づき，提案したコアアセット改善手法の有用性を，プロダクトラインの開発者の視点と，組織全体の視点から考察し，今後の課題を整理する．

(1) 各プロダクトラインの開発者の視点：

本手法は，コアアセットの進化・改善のノウハウを，改善パターンとして提供した．改善パターンにより，コアアセットの改善行動にはコアアセットの追加に加えて整理統合と削除もあることを示し，これらの行動の望ましい組合せを定義した．適用例では，保守コストを削減したいコアアセットに対して，単純にコアアセットを削除するのではなく，整理統合後に削除することが望ましいことを確認した．また，整理統合を事前に実行するかどうかで，コアアセットの追加や削除後の品質に影響がでること，不要なコアアセットを削除する改善をしないと，製品開発で無関係な部分も含めた余分なテスト等の作業が発生するリスクがあることを抽出できた．改善の実行前に，複数のシナリオの効果を比較することでリスクを洗い出し，プロダクトラインの改善計画の品質向上に貢献できた．

(2) 組織全体の視点：

我々の手法では，対応度とカバー率から決定されるタイプによって，コアアセットの状態を，対象ドメインや利用技術によらずに分類できる．タイプと改善パターンを用いると，適用例のプロダクトラインの改善方針は，次のように記述できる．

– ドメイン：情報編集支援システム
– 現状タイプ：タイプ B
– ゴールタイプ：タイプ C
– 適用パターン：Detox パターン
– 選択シナリオ：整理統合 → 削除
– 投資効果見積り：07a に 1.4（図 3，表 6，表 7 等で説明）

上記のように統一化した形式で，組織全体のプロダクトラインの改善方針を蓄積でき，プロダクトライン間の比較がしやすくなる．また，投資見積り，実行結果，成功/失敗とその根拠等と組にして，上記情報を継続的に蓄積すれば，それらを参照・再利用することにより，組織全体の投資計画の効率化，統一化，品質向上に貢献できる．

また，適用例からは，改善範囲の決定や改善コストの見積り方法に関するノウハウが得られた．我々の手法では，改善プロセスによって，改善に関する標準の改善も含めているため，このようなノウハウを標準にフィードバックできる．そして，標準の継続的な改善により，プロダクトラインの進化・改善に関する，組織の知識継承に貢献できる．

(3) 今後の課題：

コアアセットの改善手法の適用によって，次の 2 つの観点で標準を充実させることが課題である．

第 1 に，コアアセット改善パターンの改善行動のシナリオの拡張が課題である．たとえば，改善の基本行動と対象範囲別の行動の望ましい組み合わせ方に関するシナリオを改善行動案に追加することや，適用例で示した開発作業を WBS に分割した場合に各 WBS 単位に改善行動のシナリオを詳細化することが考えられる．

第 2 に，現行の改善パターンは，改善の計画立案（Plan）フェーズに関するノウハウが中心のため，改善の実行（Do），検証（Check），フィードバック（Act）についての活動ノウハウと，組織全体のプロダクトラインの最適化に関するノウハウを抽出し，パターンとして蓄積することが課題である．

8. 関 連 研 究

Clements らによるプロダクトライン型開発のためのプラクティスパターンにおいて，コアアセットの改善は，Evolve Asset パターンに定義されている[3]．Evolve Asset パターンは，コアアセットの進化（Evolution）の開始・管理・評価のために，組織が備える能力は，Technical Planning，Data Collection，Metrics, and Tracking，Tool Support，Process Definition，Testing，Configuration Management のプラクティスエリアであるとしている．本稿で提案した手法は，Technical Planning のプラクティスエリアに関連し，我々は本プランの立案に関する具体的なノウハウを示した．

Schmid らは，プロダクトラインの Evolution を，(a) Infrastructure-based evolution，(b) Branch-

and-unite，(c) Bulk に分類した[8]．(a) は，新しい要求に備えて，あらかじめコアアセットの基盤になる部分を追加するケースである．(b) は，特定の製品のリリースの後，蓄積されたノウハウをコアアセットの選択肢の 1 つ（Branch）として統合する．(c) は，(b) よりもさらに大きな選択肢に相当する別の基盤を，コアアセットに加える．(a)〜(c) は，コアアセットに何らかを追加するという観点で，本稿で提案した手法の「追加」行動に分類できる．我々の手法では，コアアセットの改善において，削除や整理統合の行動を加味し，Evolution の方法を詳細化した点に特徴がある．

Svahnberg ら[9] は，プロダクトラインの進化に関する 2 つのケーススタディから，得られた知見を提示している．その知見には，インタフェースの汎用化，関心事の分離，コンポーネントやパターンの再利用等のコアアセットの技術的な課題を解決するノウハウが示されている．このようなノウハウは有用であるが，我々の手法では，こうしたノウハウをパターンによって共有し，PDCA サイクルに基づくコアアセット改善プロセスによって，ノウハウそのものも進化・改善させることを含む点で，有用性が高い．

McGregor によれば，コアアセットの進化（Evolution）のプロセスは，進化の開始（Initiate Evolution），進化の計画の開発（Develop an Evolution Plan），変換の適用（Apply Transformations），進化したアセットの受容（Accept the Evolved Assets）からなるとしている[10]．このプロセスのうち，変換の適用が，コアアセットの改善行動である．改善行動を実現する技術として，Refactoring，Reconfiguration，Customization，Model Transformation，Change Impact Analysis，Incremental Consistency Analysis 等があげられている．本稿で提案したコアアセットの改善行動は，これらの技術に基づいて実行することになる．我々の改善パターンは，現状は，Plan フェーズに関して作成しており，Do，Check，Act に関するパターンは示していない．Do フェーズのノウハウをパターン化する場合には，McGregor の示す技術に関する使い方を具体的に述べることになる．McGregor の研究成果は，我々の改善パターンを拡張するための指針になる．

コアアセットの改善計画には，ROI 分析が必要である．簡便に技術者が ROI 分析を行える方法として，Böckle らは次の方式を提案している[7]．

プロダクトライン型開発の効果

$$=\sum C_{evo}-\left\{C_{org}+C_{cab}+\sum(C_{unique}+C_{reuse})\right\}$$

$\left(\sum\right.$ は製品数の総和$\left.\right)$

C_{evo}：従来手法で製品を開発する際のコスト

C_{org}：組織がプロダクトライン型に適合するコスト

C_{cab}：コアアセットを構築するためのコスト

C_{unique}：コアアセット非利用部分の開発コスト

C_{reuse}：コアアセットの再利用コスト

上記は，新規にコアアセットを構築する場合を想定している．適用事例で示したように，我々の手法では，C_{del} と C_{ref} のコストを考慮し，改善コストを算出した．

9. おわりに

本稿では，プロダクトラインに関わる市場，技術，組織の変化に対応して，プロダクトラインを進化・改善させるための手法を提案した．

本手法では，組織の改善のノウハウを組織の標準として定義することを提案した．そして，標準では，ドメインや利用技術によらずにコアアセットの状態を分類するために，対応度とカバー率というメトリクスを定義し，コアアセットを 4 つのタイプに分類した．また，改善の行動を定義し，望ましい行動ノウハウを改善パターンとして蓄積した．さらに，コアアセットの継続的な改善と，そのような改善を通して得られた知見をフィードバックし，標準そのものを改善するプロセスを定義した．

この手法によって，組織は，プロダクトラインの進化・改善のノウハウを継承する枠組みを標準として構築し，組織全体で一貫性のあるプロダクトラインの改善計画を立案できる．組織全体が標準にのっとって立案した改善計画は，比較・検証しやすく，組織全体のプロダクトラインの最適化に貢献できる．

今後は，改善の基本行動や範囲別の行動の組合せ方法，WBS に範囲を分割した場合の改善方法，Plan フェーズ以外の Act，Do，Check の活動方法に関するノウハウの抽出と標準への取り込みが課題である．そして，改善の実績と標準との比較，標準の精錬を通して，組織全体のプロダクトライン群の継続的な改善と知識継承に貢献する．

謝辞 研究の機会を与えてくださった東芝ソリューション（株）遠藤直樹技監，IT 技術研究所の落合正雄所長，IT 品質ラボラトリの栄光宏室長に感謝します．

参考文献

1) Nord, R. (Ed.): Software Product Lines, 3rd International Conference, SPLC 2004, p.335, Boston, MA, USA, August/September 2004, Proceedings, Springer (2004).

2) Obbink, H. and Pohl, K. (Eds.): Software Product Lines, 9th International Conference, SPLC 2005, p.235, Rennes, France, September, 2005, Proceedings, Springer (2005).
3) Clements, P. and Northrop, L.: *Software Product Lines: Practice and Patterns*, p.608, Addison-Wesley (2001). 前田（訳）：ソフトウェアプロダクトライン，p.652，日刊工業新聞社 (2003).
4) Kang, K., et al.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, CMU/SEI-90-TR-21 (1990).
5) Fowler, M., et al.: *Refactoring: Improving the Design of Existing Code*, p.431, Addison-Wesley (1999).
6) Gamma, E., Helms, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, p.416, Addison-Wesley (1995).
7) Böckle, G., et al.: Calculating ROI for Software Product Lines, *IEEE Software*, Vol.21, No.3, pp.23–31 (2004).
8) Schmid, K. and Verlage, M.: The Economic Impact of Product Line Adoption and Evolution, *IEEE Software*, Vol.19, No.4, pp.50–57 (2002).
9) Svahnberg, M. and Bosch, J.: Evolution in Software Product Lines: Two Cases, *Journal of Software Maintenance: Research and Practice*, Vol.11, No.6, pp.391–422 (1999).
10) McGregor, J.: The Evolution of Product Line Assets, (CMU/SEI-2003-TR-005), Carnegie Mellon, Software Engineering Institute (2003).

付　　録

4 章の 3 つの改善パターンの改善行動案の詳細を表 8 に示す．現状とめざすタイプの組合せにより，改善行動案を定義した．めざすタイプとして，タイプ D と，現状から対応度を下げるタイプ（例：現状のタイプ A からタイプ B をめざすこと）は，取り除き，合計 15 個の行動案と推奨案を示す．

表 8 改善パターンの改善行動案の詳細
Table 8 Detailed scenarios for core asset *kaizen* patterns.

No.	改善パターン	タイプ		改善行動案	補足説明	推奨案
		現状	ゴール			
1	Harvest (収穫) パターン	A	A	①コアアセットの追加 ②コアアセットの追加→整理統合 ③コアアセットの整理統合→追加	対応が取れている状態であるから，コアアセットを直接追加しても問題ない．追加後は対応度が下がる可能性がある．しかし，整理統合を行うことで，余計なコストをかけないことが望ましい．	①または②
2		B	B	①コアアセットの追加 ② コアアセットの追加→整理統合 ③コアアセットの整理統合→追加	対応がとれていない状態であるから，③が望ましい．	③
3		C	A	①コアアセットの追加 ②コアアセットの追加→整理統合	追加すると，コアアセット間のバランスが崩れることが予測できるため，整理統合することが重要．	②
4		C	C	①コアアセットの追加 ②コアアセットの追加→整理統合 ③コアアセットの整理統合→追加	カバー率はほとんど変化させない微量のコアアセットを追加する．対応が取れている状態であるから，コアアセットを直接追加しても問題ない．むしろ，整理統合を行うことで，余計なコストをかけないことが望ましい．	①
5		D	A	① コアアセットの追加→整理統合 ② コアアセットの整理統合→追加 ③ 整理統合されたコアアセットの追加	本文4.1で示した内容と同様．②が安全である．	②
6	Detox (浄化) パターン	A	A	①コアアセットの削除 ②コアアセットの削除→整理統合 ③コアアセットの整理統合→削除	対応が取れている状態であるから，コアアセットを直接削除しても問題ない．しかし，削除後は，対応度が下がる可能性がある．しかし，整理統合により，余計なコストをかけないことが望ましい．	①または②
7		A	C	①コアアセットの削除 ②コアアセットの削除→整理統合 ③コアアセットの整理統合→削除	上記よりもコアアセットを大幅に削除することになるため，対応度が上記よりも下がることが予測される．よって，整理統合のコストを多めに見積もる必要がある．	②
8		B	B	①コアアセットの削除→整理統合 ②コアアセットの整理統合→削除 ③整理統合されたコアアセットの削除	対応が取れていない状態のため，直接のコアアセットの削除は難しい．タイプBというゴールでも，対応度が改善されるように削除を行う．	②
9		B	C	①コアアセットの削除→整理統合 ②コアアセットの整理統合→削除 ③整理統合されたコアアセットの削除	本文4.2で示した内容と同様．②が安全である．	②
10		C	C	①コアアセットの削除 ②コアアセットの削除→整理統合 ③コアアセットの整理統合→削除	カバー率が低い状態からさらにコアアセットを削除することで，プロダクトラインとしての価値がなくならないよう留意する．削除すると，コアアセット間の対応度が崩れるので，整理統合を実施した方が良い．	①または②
11	Refactoring Core Assets (整理統合) パターン	A	A	①改善計画に従った整理統合 ②整理統合→整理統合	対応度がとれている状態からさらに整理統合して対応度を安定させることは良い改善である．しかし，必要以上にコストがかかることのないように考慮する．	①
12		B	A	①改善計画に従った整理統合 ②整理統合→整理統合	4.3で示した内容と同様．②が現実的である．	②
13		B	B	①改善計画に従った整理統合 ②整理統合→整理統合	目指すゴールは現状維持でも，少しで対応度が向上し，現状の課題を改善する方向になれば良い．②が現実的．	②
14		C	C	①改善計画に従った整理統合 ②整理統合→整理統合	対応度がとれている状態からさらに整理統合して対応度を安定させることは良い改善である．しかし，必要以上にコストがかかることのないように考慮する．	①
15		D	C	①改善計画に従った整理統合 ②整理統合→整理統合	カバー率は現状維持でも，少しでも現状の課題を改善する方向になれば良い．②が現実的である．	②

(平成 18 年 11 月 29 日受付)
(平成 19 年 5 月 9 日採録)

位野木万里（正会員）

1989 年早稲田大学理工学部数学科卒業．1991 年同大学大学院修士課程修了．同年（株）東芝入社．現在，東芝ソリューション（株）IT 技術研究所に所属．ソフトウェア生産技術の研究・開発，開発部門への適用・普及業務に従事．早稲田大学大学院理工学研究科博士後期課程に在籍．IEEE，ACM 各会員．

深澤 良彰（正会員）

1976 年早稲田大学理工学部電気工学科卒業．1983 年同大学大学院博士課程修了．同年相模工業大学工学部情報工学科専任講師．1987 年早稲田大学理工学部助教授．1992 年同教授．工学博士．ソフトウェア再利用技術を中心としてソフトウェア工学の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．