ソフトウェアプロダクトライン適用事例に基づくソフトウェア資産の再利用性に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2022 年 1 月

長 峯　基

内容梗概

近年，家電製品や空調機，カーナビ等の組込みソフトウェアは，製品の高機能化や多様なユーザニーズへの対応のため，開発規模が年々増加している．

一方で，これらのソフトウェアの開発は，新規開発から保守（派生）開発へとシフトしており，生産性を高めるため，ソフトウェアの再利用が重要な課題となっている．

ソフトウェアを体系立てて再利用するための手法の一つにソフトウェアプロダクトラインエンジニアリングがある．ソフトウェアプロダクトラインエンジニアリングとは，ある製品群において，共通かつ管理された機能を共有するソフトウェア・システム群を活用して体系的な再利用をする開発手法のことである．

ソフトウェアプロダクトラインエンジニアリングは，多くの企業での実践事例が報告されている．

しかしながら，これらの報告事例は，導入直後の実績であることが多く，長期的に見た時の経過や活動のメンテナンスについての観点からの評価が抜けており，実際にソフトウェアプロダクトラインエンジニアリングを適用したい組織が必要とする情報が欠けている．

実際に，多品種小変更型開発の特徴を有する空調機（室外機）においてソフトウェアプロダクトラインエンジニアリングを導入した場合，①上流工程からの体系的な再利用を促すプロセス定義が不十分な場合，コア資産の再利用/開発の判断が誤りやすい，②仕様差異を生み出す特徴（フィーチャ）の増加に対して構成選択が難しくなる，③並行開発時の構成管理（ブランチ・マージ）が不十分な場合，コア資産が維持できない，の大きく 3 つの課題があることを自らの経験より特定している．

本論文では，空調機（室外機）におけるソフトウェアプロダクトラインエンジニアリング導入時の取組みおよび導入直後の実績を議論の前提として述べ，その後，継続的なソフトウェアプロダクトラインエンジニアリングの実践において，ソフトウェア資産を効果的に再利用するための上記 3 つの課題について，改善事例および改善前後の定量データをもとに議論する．

1 つ目の課題である「上流工程からの体系的な再利用を促すプロセス定義が不十分な場合，コア資産の再利用/開発の判断が誤りやすい」については，コア資産管理のアプローチ（プロアクティブ型/リアクティブ型）を研究テーマとし，コア資産保守・製品導出プロセスを改善した．リアクティブなコア資産開発をベースとしたプロセスの見直しによって，類似部品発生率を 66.7%抑制し，コア資産の再利用率向上に寄与することができた．

また，2 つ目の課題である「仕様差異を生み出す特徴（フィーチャ）の増加に対して構成選択が難しくなる」については，可変性管理（可変性の記述法と構成手段）を研究テーマとし，要求仕様書における可変性の記述方法と，ソフトウェアにおける可変性の実現手段の見直しを図っている．本改善によって，仕様共通率を約 2.5 倍，構成決定パラメータ数を 56%低減し，構成決定を容易化できた．

最後に，3 つ目の課題である「並行開発時の構成管理（ブランチ・マージ）が不十分な場合，コア資産が維持できない」については，マージの衝突回避とコア資産の品質保証をテーマに，開発開始時にコア資産のマージ計画を立てるプロセスと，マージ時に将来適用する製品への組込みと評価をあらかじめ実施するマージ試験の 2 つのプロセスを追加している．本改善手法の導入によって，マージコスト比率を 61%削減，コア資産の派生 0 本を維持できている．

コア資産管理，可変性管理，構成管理（ブランチ・マージ）の 3 つの管理プロセスを改善した結果，多品種小変更型開発の特徴を有するソフトウェア開発において，コア資産の再利用率を向上させることができた．

これらの取組み結果を踏まえ，多品種小変更型の組込みソフトウェア開発を対象とした SPLE フレームワークを提案する．本フレームワークは，多品種小変更型開発における SPLE 実践時の課題に対応したもので，①リアクティブなコア資産開発と後続するアプリケーション開発における製品導出の仕組み，②ソフトウェアを解さない製品部門からもわかりやすい可変性記述と，要求仕様書から直接的にソフトウェアの構成選択が可能な仕組み，③マージの衝突回避とコア資産の他製品展開を促す仕組みを有するブランチ・マージプロセス，から構成される．

論文一覧

主要論文

[1-1] 長峯基, 中島毅, “多品種小変更型開発におけるコア資産保守・製品導出手法の改善と実践”, デジタルプラクティス, Vol.10, No.3, pp.639-655, 2019.

[1-2] 長峯基, 中島毅, 井上克郎, “多品種小変更型 SPLE 開発の経験：可変性管理の課題と解決”, デジタルプラクティス, Vol.11, No.3, pp.609-623, 2020.

[1-3] 長峯基, 中島毅, 徳本修一, 井上克郎, “空調機ソフトウェアを対象とした SPLE 開発におけるブランチ・マージプロセスの改善と考察”,デジタルプラクティス, 採録決定

国際会議

[1-1] Motoi Nagamine, Tsuyoshi Nakajima, Noriyoshi Kuno, A Case Study of Applying Software Product Line Engineering to the Air Conditioner Domain, the 20th International Systems and Software Product Line Conference, p.220-226, Beijin, China, 2016.

謝辞

本研究の推進にあたり，常日頃より適切なご指導を賜わりました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上克郎教授に，心より深謝申し上げます.

本研究の推進にあたり，副査としてご助言を頂くとともに本論文の細部に渡りご指導頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻楠本真二教授，並びに，同専攻長原一教授に，深く感謝申し上げます.

本研究の推進にあたり，業務の調整等，便宜を図って頂きました，三菱電機株式会社高丸浩一部長に，心より御礼申し上げます.

目次

第1章

はじめに

2000 年以降，家電製品やエレベータ，カーナビ等の組込み製品においては，製品機能の高度化や多様なユーザニーズを実現するための製品バリエーションの拡充が強い事業要求となっており，これらの製品に搭載される組込みソフトウェアの開発規模が年々増加している[1].

IPA の調査によれば，組込みソフトウェアの開発形態は，図 1 に示すように新規開発が 7%，改良（派生）開発が 93%となっており，ほとんどの開発が既存のソフトウェアをベースとした機能拡張や保守を主体とした開発となっている[2].

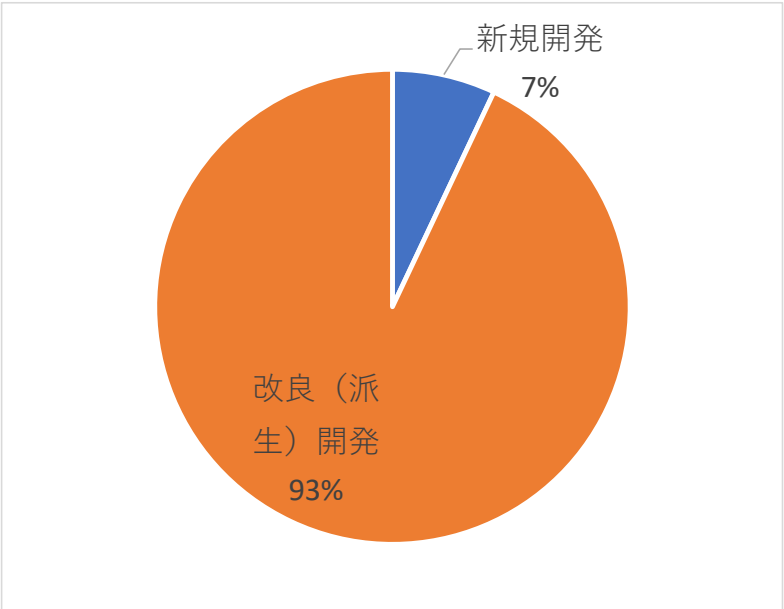

図 1 組込みソフトウェアの開発形態

このような派生開発においては，生産性を高めるため，既存のソースコードに極力手を加えず再利用することが重要であるとされている[3].

しかしながら，ソフトウェアの再利用はうまくいかないことが多い．なぜならば，必要な再利用資産を見つけ出し，再利用資産の仕様や設計意図を理解し，自分の開発するソフトウェアに組み込むことは非常に難しい作業だからである[3]．また再利用を想定した設計であっても異なるソフトウェアアーキテクチャ間では適合しないこともある [4].

以上のことから，ソフトウェア開発の再利用を成功させるには，再利用対象や再利用のさ

れ方に対する戦略を持ち，開発全体で体系的な再利用を行うことが重要であるといわれている[3].

このような課題を解決するための手法の一つにソフトウェアプロダクトラインエンジニアリング（Software Product Line Engineering，以降，SPLE と呼ぶ）がある[5].

SPLE とは，ある製品群において，共通かつ管理された機能を共有するソフトウェア・システム群を活用して体系的な再利用をする開発手法のことである．Clements らによれば，SPLE 成功のためには，29 の活動領域を繰り返し成功可能なレベルで実践できることが必要であるとされている[5].

SPLE は，世界中に数多くの実践事例が報告されているが，新規導入時や SPLE 導入直後の成果を報告したものが多く，その後の派生開発での取り組み報告は多くはない．また，報告されている事例は失敗事例が少なく，SPLE 成功において，どの活動領域がどの程度重要であるかは，明確ではない[6]．つまり，SPLE の各活動領域の実践の程度とそれが品質や生産性に与える影響についての評価や，長期的に見た時の評価が抜けているため，実際に SPLE を適用したい組織にとって必要な情報が不足している．

そこで本論文では，組込み製品の一つであり，多品種小変更型開発の特徴を有する空調機（室外機）への 5 年にわたる SPLE 適用経験（失敗原因の分析と各プラクティスの改善活動）から，長期的な SPLE 適用評価を行うとともに，SPLE の各活動領域の実践が不十分だった場合に生じる問題点とその解決策について定量的なエビデンスに基づいて議論する．また，これらに加えて，組込みソフトウェア開発（多品種小変更型開発）における SPLE フレームワークを提案する．

1.1. ソフトウェアプロダクトラインエンジニアリング（SPLE）

1.1.1. SPLE の定義

SPLE とは，ある製品群において，共通かつ管理された機能を共有するソフトウェア・システム群を活用して体系的な再利用をする開発手法のことである[3][5][7]（以降，SPLE の定義を本文献に基づいて概説する）．つまり，SPLE は，単一製品の開発を効率化するものでも，単なるコンポーネントベースの開発手法でもなく，ある製品群において再利用をあらかじめ計画し実行していくものとされている．

Clements らは，SPLE は 3 通の必須活動から構成されていると述べている．3 つの必須活動とは，①コア資産開発（ドメイン開発），②製品導出（アプリケーション開発），③管理プロセスを指す．ここでコア資産とは製品群に共通的に利用可能なソフトウェア資産（設計書やソースコード）を指す．

ドメイン開発では，SPLE のスコープ（対象範囲）を定義し，スコープにおけるコア資産を開発する．ドメイン開発においては①スコープを適切に定義し可変性を明確にすること，②製品群に共通利用可能かつ製品ごとの差異を吸収可能なアーキテクチャを定義すること，

③再利用可能なコア資産を開発すること，が重要であるとされている．

アプリケーション開発では，コア資産を用いて個々の製品を開発する．アプリケーション開発においては，①再利用資産を活用して個別の製品開発を行うこと，②要求－設計 - コンポーネント間のトレーサビリティが確保されていること，が重要であるとされている．

管理プロセスは，技術的もしくは組織的な管理を指し，SPLE を成功に導くための極めて重要な役割を担う．例えば，①再利用資産の変更を管理すること，②再利用資産の構成管理を行うこと，③並行開発が管理されていること，④再利用資産の保守・運用に関する管理がなされていること，が求められている．

これらの活動を，冷蔵庫を例に考えると次の通りとなる．冷蔵庫には冷蔵室や野菜室の冷蔵機能のような共通機能のほかに，自動製氷機能の有無やネットワーク接続機能の搭載有無（ネットワーク接続して利用できる機能にも機能差がある）といった差異がある．これらの差異を吸収可能なアーキテクチャを構築し，再利用可能な資産を開発することがドメイン開発に相当する．このドメイン開発で開発した資産をもとに自動製氷機能やネットワーク接続機能といった差異を可変性として実現し，各製品向けのソフトウェアを開発することをアプリケーション開発と呼ぶ．そして，これら開発した資産が再利用可能な状態で維持されるように管理することを管理プロセスと呼ぶ．

SPLE では，これら 3 つの必須活動を技術とビジネス両方の側面から融合させることが重要である．

Clements らによれば，SPLE を成功させるためには，3 つの必須活動を本質的なレベルで実行できるよう必要があるとされている．ここで，本質的なレベルとは，繰り返し成功可能であることを意味している．そのためには，ソフトウェア技術，技術管理，組織管理の 3 つに活動カテゴリに分類された 29 個の技術領域について習得する必要がある．

カーネギーメロン大学が定義した 29 個の技術領域について，活動カテゴリごとに分類すると以下のとおりである．

Software Engineering practice area（以降，ソフトウェア工学活動）は，適切な技術を使用してコア資産開発や製品導出を行うためのものであり，他の活動カテゴリから見た際に基盤となる活動カテゴリである．カーネギーメロン大学の定義に基づきソフトウェア工学活動の技術領域をまとめると表 1 の通りとなる．

表 1 ソフトウェア工学活動の技術領域

技術領域	内容
アーキテクチャ定義	製品群に共通利用可能，かつ製品固有の差異を吸収可能なアーキテクチャを設計すること
アーキテクチャ評価	チェックリストや定量値の測定によってアーキテクチャの品質を評価する
部品開発	適切な構成手段を選択し，再利用可能なソフトウェア部品を開発する
コア資産探索	SPLE 適用対象の進化に合わせて，システムの一部を見直し進化に適応させること
要求工学	製品群全体の仕様と，仕様差異を生み出す特徴（フィーチャ）に基づく製品固有の仕様を決定すること
ソフトウェア統合	プロアクティブ/リアクティブに開発したコア資産をマージし再利用可能とすること
テスト	コア資産と製品固有のソフトウェアのそれぞれを評価すること
ドメイン理解	製品群の共通性と可変性を決定するための最初のステップであり，対象製品の分析を含む
外部ソフトウェア利用	コア資産の一部や製品固有のソフトウェアに外部ソフトウェアを利用すること

Technical Management Practice Area（以降，技術管理活動）は，コア資産開発や製品導出を管理するための活動カテゴリであり，ソフトウェア工学の各技術領域の実行を管理・支援するものである．カーネギーメロン大学での定義に基づき技術管理活動の具体的な技術領域をまとめると表 2 のようになる．

表 2 技術管理活動の技術領域

技術領域	内容
構成管理	各製品ソフトウェアのバージョン管理だけでなく，各製品ソフトウェアの構成（どの部品から構成されているか）を管理する．可変性管理（製品群全体の共通性や可変性の管理）や並行開発管理（ブランチ・マージ）も含む.
自作/購入/委託分析	製品群の持つ制約や戦略に照らして検討すること，また中期的な品質・生産性を加味して決定すること
測定と追跡	3 つの必須活動を測定し，サポートするものでなければならない.（例 : 再利用性や製品群全体に対する戦略的な計画の評価）
プロセスの規律	ドメイン開発者やアプリケーション開発者が合意された方針に基づいて確実に協力しあうこと
スコープ定義	SPLE を長期的に実行可能とするためのスコープを決定すること
技術計画	コア資産開発やコア資産保守の方法，製品導出の方法を定めること
技術リスク管理	リスクは複数製品にわたることが多いため，個別の問題として扱うのではなく，組織の問題として扱う必要がある
ツール支援	コア資産管理，可変性管理，開発プロセス，組織管理等，SPLE の様々な場面でツールによる支援が必要である

Organizational Management Practice Area（以降，組織管理活動）は，SPLE 全体を調整するための活動カテゴリであり，具体的な技術領域をまとめると表 3 の通りである．

表 3 組織管理活動の技術領域

技術領域	内容
ビジネスケース定義	ビジネス的な意思決定をするためのツールであり，SPLE 導入を正当化すること，もしくは特定の製品をスコープに含めるかを決定するために用いられる
顧客 I/F 管理	単一の製品・単一の顧客の観点ではなく，複数製品・複数顧客の観点で管理する
買収戦略策定	SPLE 成功のため，コア資産やコア資産から生成された製品を買収すること
資金調達	SPLE のための組織準備，トレーニング，通常とは異なるプロセスや開発環境のために重要な活動である
立ち上げと制度化	組織の能力とニーズに応じて，他の技術領域の実践を取り入れていくことが重要である
市場分析	組織を単一製品の開発から SPLE に転換させる意思決定の際に重要な取り組みである
オペレーション	3 つの必須活動の実行を指し，これらをビジネス全体でみて調整することが重要
組織計画	SPLE を用いた組織への移行計画とコア資産開発および保守のための予算確保を含む活動である
組織リスク管理	個々のプロジェクトのリスク管理ではなく，組織の SPLE 成功に影響を及ぼすリスクを指し，組織全体で管理する必要がある
組織構築	コア資産開発と製品導出という役割を適切な組織単位に分割して割り当てることが重要である
技術予測	製品群の製品機能を実現する技術と，SPLE 適用を支援する技術の両面から継続的に予測・評価することが重要
トレーニング	初期の SPLE 導入と継続的な SPLE の活用の両面のために必要なものである

SPLE 成功のためには，これらの技術領域が個別に実行されるのではなく，重なり合いながら実行されることが重要とされている．

1.1.2. SPLE の適用事例

2000 年以降，SPLE の実践事例が数多く報告されている[8][9].

例えば，Cummins Inc.（カミンズ社）では，SPLE 適用によって生産性が 3.6 倍になったと報告されている[5][10]. カミンズ社では，燃料システムを取り扱っており，それぞれの燃料システムが異なるアーキテクチャに基づいて開発されていていた. カミンズ社では，燃料システムの気筒数や排気量の違い，プロセッサ等のハードウェア違いを可変性として定義し，SPLE を導入している．この結果，各燃料システムを個別に開発していたころと比べて開発人員を大幅に削減できている.

また，医療用画像処理装置を取り扱っている Philips Medical Systems（フィリップスメディカルシステム）では，増大する多様性への対応のため SPLE を導入している[5][11]. 社内の医療用ミドルウェアをベースにプラットフォームを構築し，SPLE への転換を図っている. プラットフォーム構築後は，プラットフォームの適用範囲を拡大し，個別の製品開発にかかる時間を削減している.

日本では，株式会社東芝にて発電監視制御システムにおける SPLE の実践事例が報告されている[12]. 東芝の SPLE では，プラントの共通部と固有部を明確に分離してコア資産を構築している. コア資産の組み合わせによってハードウェアや OS が異なるシステムも導出可能となっており，自由度の高いシステム構築を実現している.

1.1.3. SPLE の最新動向

2006 年ごろから，ソフトウェアの複雑化や短納期化に加え，不確定な要求事項へと対応するため，既存のウォータフォールモデルをベースとした開発プロセスだけでなく，アジャイル開発が取り入れられ始めている[13]. 平鍋らによれば，ウォータフォールモデルが分析・設計・実装・テストを順番に進める（設計したもの文書に残し，文書のレビュー・承認を経て次の工程へと進める）のに対し，アジャイル開発は，短い期間に分析・設計・実装・テストを並列に実行し，ユーザの求める一部の機能を開発しフィードバックを繰り返していくことに特徴がある.

さらに近年では，このアジャイル開発と SPLE を組み合わせた APLE（Agile Product Line Engineering）が研究・実践され始めている[14]. APLE とは，アジャイル開発と SPLE の共通のゴールであるビジネス価値を高めるために，アジャイルの持つ俊敏さと SPLE の持つ再利用性の融合を狙ったものである.

APLE においては，先述の通りアジャイル開発の持つ俊敏さと SPLE のもつ再利用性を両立させるため，コア資産や製品群のポートフォリオ管理，コア資産や製品ソフトウェアの開発をどのように実践していくか，というフレームワークの構築がさかんに研究・実践されている[14][15][16].

また，Marques らによれば，近年 SPLE とモデルベース開発との組み合わせが数多く研

究されている[17]．モデルベース開発とは，制御モデルをブロック図等で表記し，プログラム作成前にコンピュータ上でシミュレーションすることで制御仕様の検証を可能とする技術である[18]．近年ではこの制御モデルを活用し，製品のソースコードを自動生成する技術も活用されている[19]．

Marques らの調査によれば，モデルベース開発はコア資産の進化を推進する役割として重要な役割を果たしている．例えば，モデルを用いて可変性管理を行ったり，オートコードを含むモデルベース開発としての機能を SPLE に組み込んだり，SPL からの追加や削除のパターンカタログを作成したりしているものが多く研究されている[20][21]．

SPLE は従来からの方法論に加え，近年採用事例が増加しているアジャイル開発やモデルベース開発との融合が進んでいる．

1.2. 本研究の位置づけ

本研究は，組込みソフトウェア開発（多品種小変更型開発）における SPLE 実践事例に基づき，SPLE 成功に必要な技術領域の習得や実践が不十分だった場合に起きる問題を分析し，改善前後の結果を比較検証することで，SPLE の継続的な成功に必要な技術領域について定量的なエビデンスに基づき考察する．また，ここで得られた知見に基づいて，多品種小変更型ソフトウェア開発における SPLE フレームワークを提唱する．これによって SPLE 導入や運用の失敗を未然に防止し，組込みソフトウェア開発の生産性を向上させることを目的とする．

1.2.1. SPLE 適用対象

本論文では，業務用空調機（室外機）のソフトウェアを対象とする．この製品群は，日本国内のみならず世界各国へ出荷され，冷暖房能力や機能の搭載有無，ハードウェア構成の違いがあるため，様々な製品バリエーションを持っている．ハードウェア起因の小さな仕様の差異をいかにソフトウェアでカバーするかが開発戦略上重要であり，多品種小変更型開発として，以下の特徴を有する．

(ア)製品の機能・性能がハードウェアに大きく依存するため，製品群の仕様及び開発計画は，機械や電気部門（まとめて製品部門と呼ぶ）が決定する[31]．製品部門は用途や出荷地域が異なる固有の市場ごとに組織（部や課）が分かれており，互いの仕様変更を把握していない．

(イ)製品部門からの仕様は，コア資産に基づくものでなく，以前開発した製品に基づいたものとなる．例：前年製品をベースに，ある機能を変更する等．

(ウ)個々の仕様変更は小規模であり，短期に並行して実施される．

適用対象の製品群において，製品部門は市場ごとに組織が分かれているのに対し，ソフトウェア部門は，一つの部門が各製品向けにソフトウェアを開発し提供している．

1.2.2. SPLE 適用対象における課題

業務用空調機は，多品種小変更型の特徴を有しているため，SPLE 実践においては以下の課題がある．

課題①
特徴（ア）に示すように，多品種小変更型開発では，製品群への要求管理は顧客に近い製品部門が担っているため，ソフトウェア部門主導で要求事項の分析を進めることが難しい．また異なる市場に対応する多数のアプリケーション開発担当組織が，同時並行的に類似した要求仕様で開発を始めようとするため，実際にアプリケーション開発を実施するソフトウェア部門がソフトウェア部品を再利用するか新規に開発するのかの判断を誤る場合が増えてしまう．

課題②
特徴（イ）および特徴（ウ）から，コア資産に関する知識を持たない製品部門からの変更要求をもとにソフトウェア部門がコア資産の再利用/新規開発を判断することになるが，並行した開発から類似した仕様変更が数多く入力されるため，仕様差異を生み出す特徴（フィーチャ）の増加に対して構成選択が難しくなる

課題③
特徴（ウ）に示すように，同時並行の開発プロジェクト間で独立してコア資産に変更を加えるため，複数の開発が並行する中でマージの衝突が起こりやすい

1.3. 本論文の構成

本論文では，SPLE 適用の失敗事例の分析に基づき，SPLE 成功に必要な技術領域の実践の程度と生産性および生産性に影響を与える再利用性の関係について議論する．

2 章では，SPLE 成功の要件をまとめるとともに，三菱電機の空調機（室外機）における SPLE 導入後の実績に基づき SPLE 成功に必要な技術領域の不足と生産性に影響を及ぼす各指標の関係について示す．

3 章では，2 章で示した失敗原因の 1 つである「上流工程からの体系的な再利用を促すプロセス定義が不十分であり，コア資産の再利用/開発の判断が誤りやすい」という課題について議論する．再使用可能なコア資産の開発には，あらかじめ可変性を予測してコア資産を開発するプロアクティブなアプローチと，製品開発の中で開発したソフトウェア資産の中

から再利用可能なものをコア資産として採用するリアクティブなアプローチの 2 種がある．多品種小変更型開発においては，類似仕様が発生しやすく，詳細なソフトウェア仕様の決定が遅いという特徴があるため，コア資産をプロアクティブに開発することが難しい．本章では，リアクティブなアプローチをベースとしたコア資産保守・製品導出プロセスを提案し，類似仕様の発生率をもとに本改善手法の有効性を評価する．

4 章では，2 章で示した失敗原因の 2 つ目である「仕様差異を生み出す特徴（フィーチャ）の増加に対して構成選択が難しくなる」という可変性管理における課題について議論する．ソフトウェアの構成手段（可変性を実現するための方法）には様々な方法が提案されているが，本章では，可変性記述法および可変性の構成手段の違いによる構成選択のしやすさについて比較検証する．要求仕様書からビルドへ直結する構成手段を提供することで，フィーチャの増加によらず構成選択を容易にすることができた．

5 章では，2 章で示した失敗原因の 3 つ目である「並行開発時の構成管理（ブランチ・マージ）が不十分であり，コア資産が維持できない」について議論する．多品種小変更型開発では，開発が並行することが多く，コア資産のブランチ・マージ（構成管理）が重要である．本章では，マージの衝突を回避する方法およびマージされたコア資産を再利用するための品質保証を研究テーマとし，開発開始時にマージ計画を立てるプロセスと，マージ前に他製品開発での利用を想定したマージ試験を実施するプロセスを提案する．本改善手法の適用によって，マージの衝突回避およびコア資産の維持に成功した．

6 章では，3 章から 5 章の結果をまとめ，将来への課題を述べる．

第2章

SPLE の実践と 3 つの必須活動の必要性の検証

2.1. SPLE 導入の背景と目的

近年携帯電話やカーナビ等の組み込み製品においては，他社製品との差別化のための製品機能の高度化・複雑化，多様なユーザニーズへのタイムリな対応が求められている．このためこうした製品に組み込まれるソフトウェアの開発は，品質を確保しつつも，高い生産性と短納期化を達成することが重要となっている．

このような課題を解決するためには，既存のソフトウェアになるべく手を加えず，再利用をすることが求められる．そのための体系的な開発手法として，ソフトウェアプロダクトラインエンジニアリング（以下，SPLE）が提案され，広く実践されている[5].

しかしながらこれらの事例報告は，SPLE 導入直後の成果報告である場合が多く，長期的に見た時の経過や活動のメンテナンスについての観点からの評価が抜けており，実際に SPLE を適用したい組織が必要とする情報が欠けている．

本章では，業務用空調機（室外機）を対象とした SPLE の適用事例をもとに，①SPLE の 3 つの必須活動の実装の程度が，生産性と品質に与える影響と問題点，②長期的に見た時の適用結果を分析評価する．

2.2. SPLE 適用の流れ

表 4 に三菱電機の空調機（室外機）における適用の流れを示す．適用ステージは，準備，適用，評価改善に大きく分かれる．

表 4 SPLE 導入の流れ

項目	適用ステージ		
	準備	適用（第 1 次）	評価改善
時期	2006～2009 年	2010 年～2014 年	2015 年～
実施活動	ドメイン開発	アプリケーション開発 及びドメイン開発	適用評価 SPLE 改善

準備ステージは， 2006 年～2009 年の 4 年間をかけドメイン開発を実施し，コア資産を構築した段階である．実際の製品開発と開発が並行したため，製品開発において作り込んだ機能を，コア資産に取り込む作業を行った．

適用ステージ（第 1 次）は，2010 年～2015 年で，準備ステージで構築したコア資産を用

いて，個別機種を本格的した開発段階である．この段階において，各機種は年度ベースで開発が実施され，各年度で複数の機種が並行開発された．

適用ステージにおいて開発を重ねていくにつれ徐々に効果が低下したことを受け，評価改善ステージでは，これまでの成果と適用上の問題点を整理する目的で設けられた．本章は，このステージの活動成果をまとめている．

以下，2.3 節はドメイン開発で特徴的な事項と活動内容，2.4 節で適用ステージにおける成果と明らかになった問題点をまとめる．

2.3. ドメイン開発

2.3.1. 実施内容

本事例では，主に準備ステージで実施したドメイン開発を SPLE の開発プロセス例[22]に従い，① 製品群のへの要求事項および現状分析，②プロダクトラインアーキテクチャの構築．③製品群共通部品・機種固有部品の開発，の流れに沿って活動を行った．

(1)　製品群への要求事項および現状分析

このプロセスでは，既存の業務用空調機（室外機）全製品群を対象として，個別機種の製品機能や制御仕様を分析し，機種間の共通機能と機種固有機能を分類・整理し，可変性を定義した．

可変性の定義においては，製品機能の搭載有無やパラメータ差分等の機種間の差異を，以下の理由別に整理した．

① アクチュエータの相違：圧縮機，室外ファン等
② その他の H/W の相違： スイッチの有無等
③ 機種バリエーション： 上位機種，廉価機種

なお，分析を通じて，機能名称は同一だが制御方式が異なる機能や，目的は同一だが実現手段が異なる機能を多数抽出することができた．

この抽出結果をもとに，製品レベルで設計思想が異なる機能を含む製品シリーズや特殊用途向け機能をプロダクトラインの対象外とするなどの選択を行い，既存の業務用空調機（室外機）製品群をカバーするドメインをスコープに含めることができた．1.1.1 項で定義した適切なスコープを決定できたと考える．

(2)　プロダクトラインアーキテクチャの構築

このプロセスでは，製品群に共通利用可能かつ機種ごとの差異を吸収可能なプロダクトラインアーキテクチャを構築することを狙って，フレームワーク（以下，GAO フレームワーク）を設計・構築した．

GAO フレームワークの設計に当たり，各機種の代表製品の設計仕様書とソフトウェアを対象に，静的な構造とふるまいを分析し，違いを吸収するやり方を採った.

図 2 に GAO フレームワークの静的な構造を示す.

図 2 GAO フレームワーク

GAO フレームワークは，状態遷移を司る状態管理層，アクチュエータごとの制御を司る制御機能層，状態管理層と制御機能層で必要なデータを定義した情報層，アクチュエータを動かすためのデバイス層からなる．無秩序な参照を防止するために，階層間の依存関係を 1 方向に規定し，階層間は標準化されたインタフェースでのみアクセス可能とした．これにより，課題であった機能追加や変更による機能間の相互依存の増大について解決を図った．その根拠を後述する各層の説明にて示す.

GAO フレームワークは，スコープ内の機種共通で利用可能な，運転モードの状態遷移やアクチュエータ制御の基本的なふるまいを共有し，競合する機能間の調整を組み込んだものである．以下，状態管理層と制御機能層について，フレームワークの構造とその理論的根拠について示す.

■状態管理層

業務用空調機（室外機）の既存ソフトウェアの分析から以下の 3 点がわかった，

- 冷房中や暖房中等などの空調機全体としての制御の状態（運転モード）に沿って，制御機能群を選択し実行する.
- 運転モードの遷移や時間経過に合わせてアクチュエータの制御内容が変化する
- 運転モードは排他的であり，いずれか一つを実行する

これらの分析結果から，運転モードとアクチュエータの状態の 2 つの状態遷移を管理し，これらの組合せにより，起動される制御機能を決定するという設計を行った．すなわち，図 3 に示すように，運転モードとアクチュエータ状態の組み合わせテーブルを作り，各セルに 1 つの制御機能を定義するようにした．そして，これらの制御機能をソフトウェア部品（制御機能部品）として実装する.

制御機能を部品化する単位を標準化したため，部品のインタフェースが統一的に決定さ

れ，また粒度のばらつきがなくなった.

アクチュエータ / 状態 / 運転モード	起動中	停止中	・・・
冷房中	冷房起動制御	停止制御	・・・
暖房中	暖房起動制御	停止制御	・・・
・・・	・・・	・・・	・・・

図 3 ソフトウェア部品表

■制御機能層

代表的な機種について，制御機能に相当するモジュール群の現状を調査し，以下の 3 点がわかった.

- 空調機の機能を実現するモジュールが，4 つの処理カテゴリ：入力，基本制御，補正制御，出力に分類できること，各機能は，処理カテゴリ間のデータ処理のシーケンスとして表現できる.
- 基本制御と補正制御のモジュールは，アクチュエータ毎に異なり機種固有な部分が多いのに対して，入力と出力のモジュールは共通できる部分が多い.
- 補正制御間で競合する場合があり，補正制御は一定のルールによって優先度が決まる.

これらの性質を利用し，制御機能層は，図 3 に示す構造を採用した．図 4 において，制御順序とデータの流れを管理する 4 つのモジュールと，基本制御と補正制御モジュールから呼び出されるアクチュエータ固有の制御部品群とに分けた.

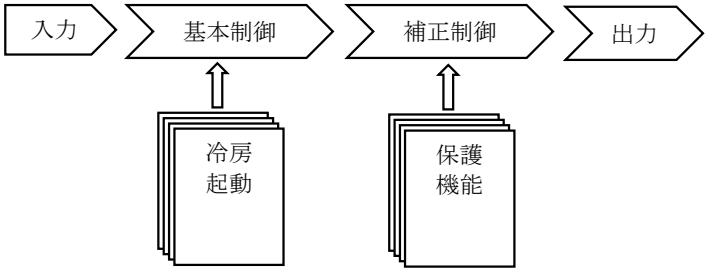

図 4 GAO フレームワークにおけるアクチュエータ制御

さらに各部品の出力を，統合・調整して空調機としての出力を決定することで，制御機能間の競合を回避する仕組みを組み込んだ.

プロダクトラインアーキテクチャは，これらのフレームワークを用い以下のように実現した．各階層のパッケージには機種間に共通な機能だけでなく，機種固有の機能を登録する.

新しい製品向けに開発する機種固有部品も含め，製品を構成するのに必要なすべてのソフトウェア部品を登録テーブルと呼ばれるリストに登録する仕組みとした．これにより，GAO フレームワークは，部品登録テーブルに登録されたソフトウェア部品を呼び出して実行する．

これにより，1.1.1 項で定義したように，製品群に共通な機能だけでなく，機種固有の機能の実現も容易なソフトウェアアーキテクチャを実現することが可能となった．しかし，分析の対象として，既存ソフトウェアの共通点と差異点にのみ注目したため，要求の視点から見たときの共通部および可変性の管理という観点が不十分だった．

(3)　　製品群共通部品・機種固有部品の開発

ドメイン開発 3 つ目の取り組みとして，製品群全体に共通利用可能，もしくは少ない変更で利用可能なコンポーネントの開発を行った．例えば，制御機能層に各アクチュエータの基本制御部品や製品の保護機能を補正制御部品として開発している．

また，再利用可能なコア資産開発のため，要求仕様書とソースコードそれぞれで機能と定義するものの粒度が同一となり，かつそれらが一対一に紐づくよう制御仕様モデル（図 5）を構築し，要求仕様書の目次や記載項目のひな型を作成，アプリケーション開発の中で運用することを試みた．

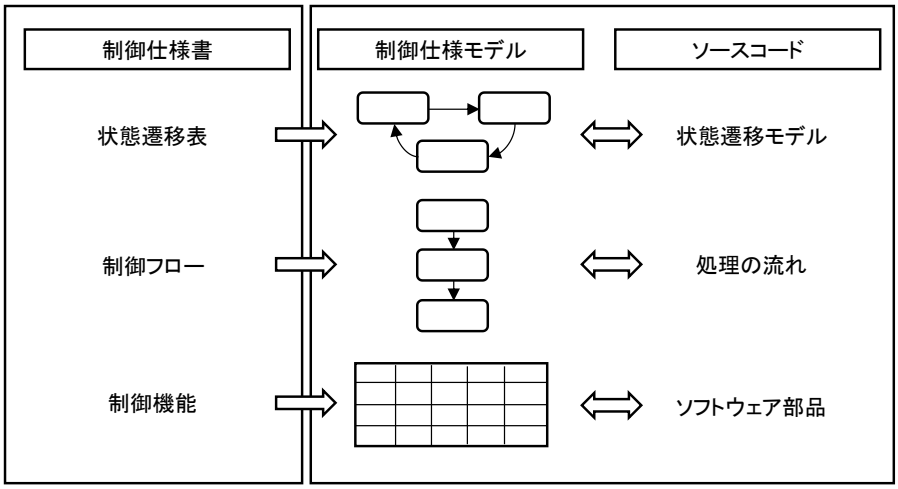

図 5 GAO フレームワークにおける制御仕様モデル

さらに要求仕様書の仕様記述に際しては，各部品の記載方法を定型化することで，漏れ・ムラのない仕様記述を可能にし，ソフトウェアを実装する際に標準化可能な仕組みとした．この要求仕様書をもとにソフトウェア部品を開発した．これにより 1.1.1 項に述べている再利用可能なコア資産が開発できたと考える．

2.3.2. ソフトウェア指標からのドメイン開発の評価

ドメイン開発結果の評価の一部として，プロダクトラインアーキテクチャのソフトウェア指標の面から評価する．表 5 は，従来ソフトウェアと本プロダクトラインを，関数の平均サイズ，複雑度（サイクロマティック複雑度)，影響度（1 か所修正した際に変更影響が及ぶ範囲）から定量的に比較する．

表 5 SPLE 適用後のソフトウェア指標に基づく評価

	改善後
関数平均サイズ	20.0%低減
平均複雑度	24.0%低減
影響度平均	40.7%低減

表 5 から，関数の平均サイズ，複雑度 24%低減する結果を得た．これは，機能制御部品の単位を，比較的小さく粒度の揃ったものに決めたことによると考える．影響度も 40.7%低減しているが，これは，従来にない階層の概念を導入したことと，従来部位間で双方向に依存関係が生じていたのに対し，GAO フレームワークが依存関係を一方向に限定したことによると考える．

2.4. アプリケーション開発と管理プロセス

個別機種の開発においては，従来の開発プロセスをベースに，トレーサビリティを確保するためのプロセスを定義して製品開発を行った．このとき，1.1.1 項に記載したアプリケーション開発における重要事項を意識してプロセス定義を行っている．

具体的には，個別機種の変更要求に基づき，制御仕様モデルをもとに作成した要求仕様書を作成した．この要求仕様書に基づき，一対一に対応したソフトウェア部品を開発し，ソフトウェア部品の組み合わせで機種開発を行うアプローチを採った．このように，個別機種の変更要求を，実現可能なコア資産の選択および組み合わせ，もしくは小改修で機種開発を行うよう試みた．

また，個別機種開発中の変更管理については，製品の変更要求とソフトウェアの変更点のトレーサビリティを確保し，変更の経緯や変更箇所の管理を行った．

ただし，構成管理については，SPLE に固有のものを定義する必要性を理解しておらず，従来のやり方をベースとしたプロセスとした．具体的には，バージョン管理は，ソフトウェア部品ごとではなく，プロダクトラインアーキテクチャを含むすべてのソフトウェアをまとめた管理，並行する複数機種の開発に対しては，従来通り緊密なコミュニケーションと日程調整によって開発時期をずらすことでブランチ・マージプロセスは不要と考え特段のプロセスを整備しなかった等，管理プロセスの実装の程度に不備があり，個別機種の開発に柔軟に対応できず（開発したソフトウェア部品をマージできず)，その後ソフトウェアを再び

派生させてしまう結果となった.

2.5. 適用結果まとめ

従来ソフトウェアを用いて開発していた時期と SPLE へと転換を図った直後およびその数年後の経過をもとに SPLE 導入による効果を検証する.

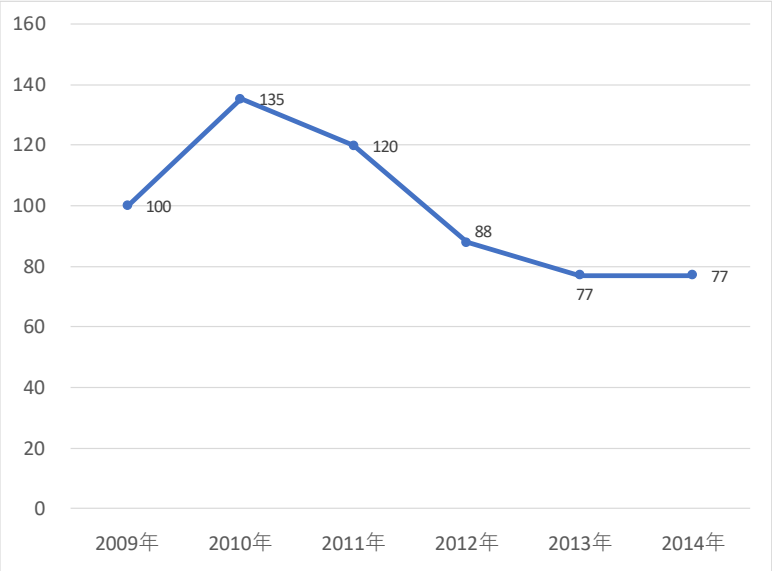

図 6 生産性 Pd の推移

図 6 に生産性 Pd の推移を示す. 生産性 Pd は, 1 人 1 か月で開発可能な開発規模と定義し, 2009 年を 100 とした相対値で示す. 図 6 に示すように, 従来ソフトウェアを用いた開発を長年継続し, また, 周囲を取り巻く事業環境の変化もあり, 準備ステージ 2006 年～2009 年（従来ソフトウェアによる開発を継続していた期間）は, 製品群全体における生産性は徐々に低下していた. 適用ステージ 2010 年～2014 年は, 当初製品群全体における生産性は 2009 年比 1.35 倍に向上した. これにより, 開発期間短縮の効果も得た. その後生産性は徐々に低下し, 2009 年比 0.8 倍まで低下した.

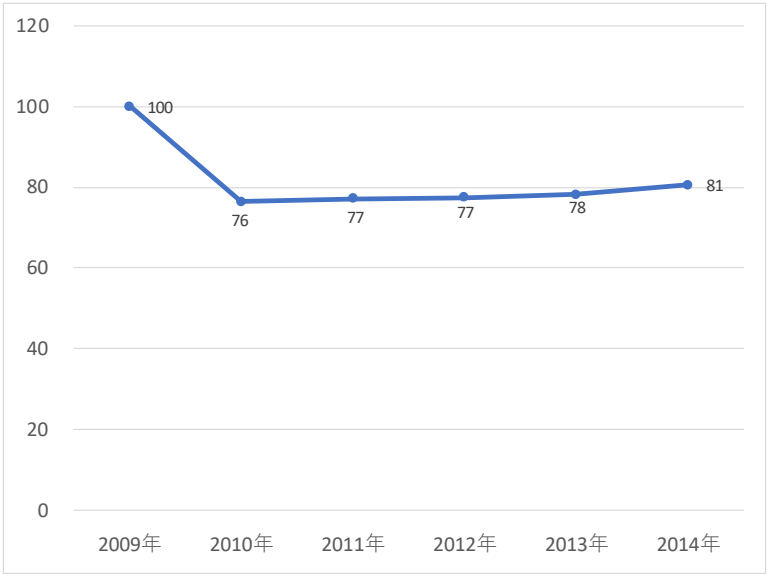

図 7 サイクロマティック複雑度 Cc の推移

図 7 にサイクロマティック複雑度 Cc の推移を示す．2010 年に SPLE を導入した時点で 2009 年比約 0.8 倍へと複雑度を低減し，その後も 2009 年比約 0.8 倍と 2009 年と比較して低い水準を維持している．このことから 2010 年以降も複雑度の点から見た保守性は高かったと考えるが，別の要因により生産性が低下したものと考える．

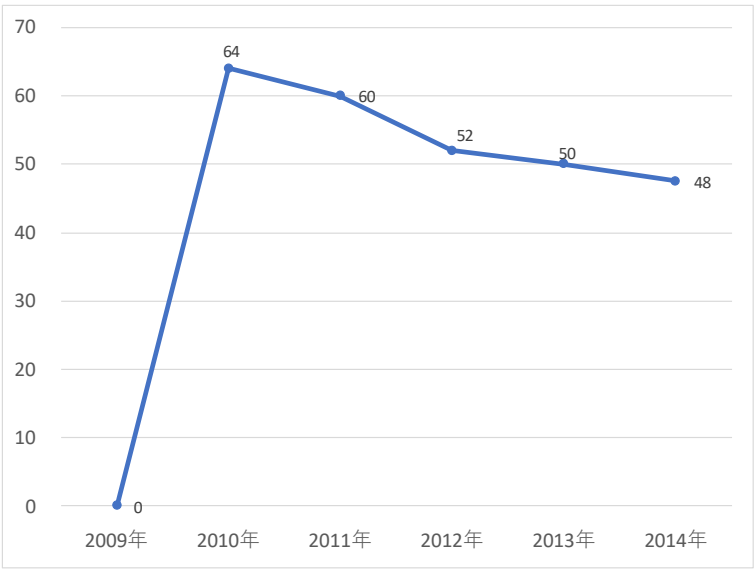

図 8 ソフトウェア部品再利用率 Rp の推移

図 8 に同じ期間のソフトウェア再利用率 Rp の推移を示す．ここで，ソフトウェア再利用率の定義は以下である．

Rp ＝ A/B （1）

A：再利用されたソフトウェア部品数

B：全ソフトウェア部品数

2009 年以前はソフトウェア系列が異なれば，同一仕様であっても再度開発しているため，再利用率は 0 である．2010 年以降からの再利用率の動きは，生産性と連動している．2010 年度は 60%まで向上し，その後は徐々に下がり，2014 年には 50%まで低下した．

適用ステージ当初は，SPLE の対製品カバー率が低いにもかかわらず目論見通り 2009 年比 135%の生産性の向上と，再利用率 60%という結果をもたらしたが，その後徐々にその効果は低下した．その原因を以下のように分析している．

① 類似部品発生による再利用率の低下

類似部品発生率 Rs を図 9 に示す．ここで，類似部品発生率 Rs の定義は以下である．

Rs ＝ 類似部品数／全ソフトウェア部品数 (2)

ここで，類似部品は，祖が同一であるソフトウェア部品のことである．

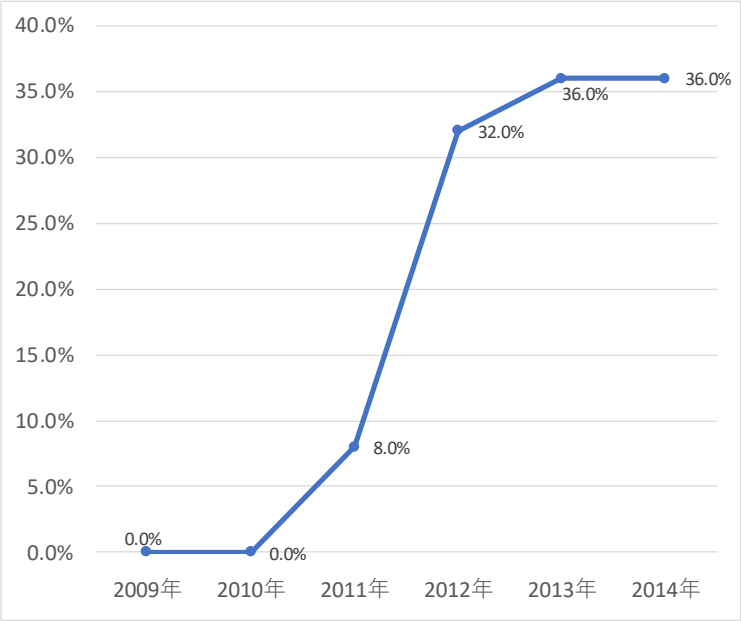

図 9 類似部品発生率 Rs

図 9 に示す通り，2010 年の SPLE 導入以降年々増加し，2014 年には 30%が類似部品となってしまった．この結果は，図 8 に示すソフトウェア部品の再利用率と負の相関関係があると言え，類似部品が多くなるにつれて，再利用率は低下していることがわかる．

これはソフトウェア部品のサイズが大きく，また，2.4 節で述べた通り，単一のソフトウェア開発時と同様の構成管理手法を踏襲し，ソフトウェア部品を個別に管理するという

ことを実施しなかったため，わずかな仕様差異を含む変更要求が発生するたびに，類似したソフトウェア部品が開発されてしまったこと（P1）が主要因であると考える．

② 仕様共通化率の低調による再利用率低下

仕様共通化率 Rc の推移を図 10 に示す．

Rc ＝ 1 - A/B （3）

A：単一機種でしか搭載されていない機能項目数

B：機能項目数

ここで，機能項目とは図 5 における制御仕様書中の制御機能を指す．2009 年を 100 とした相対値で示す．図 10 に示す通り仕様共通化率は適用ステージ 2010 年～2014 年の間に徐々に低下し，2014 年には 2009 年度比 0.8 倍まで低下した．SPLE を成功させるために戦略的な再利用が重要であることは広く言われているが，このグラフに示すように仕様共通化率は変化がなく低調であり，体系的かつ戦略的な仕様共通化プロセスが有効に機能していなかったと言える(P3)．

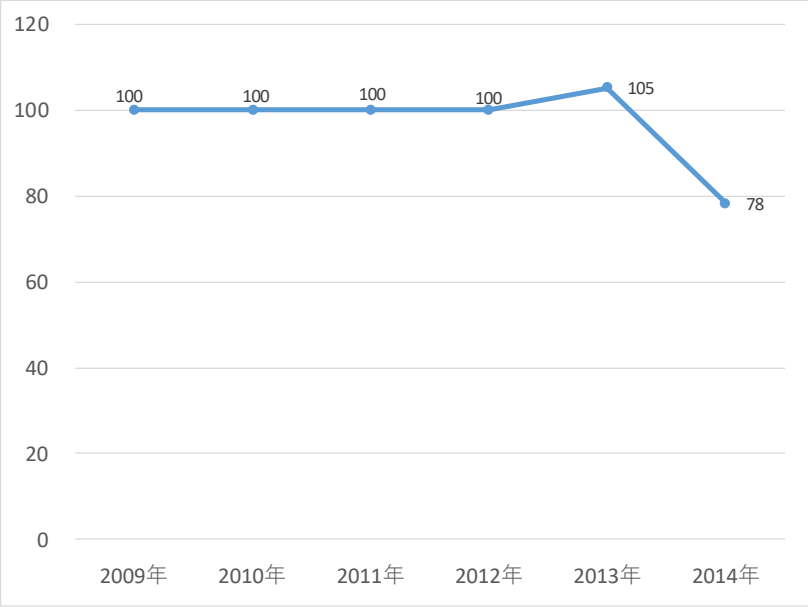

図 10 仕様共通化率 Rc の推移

③ アーキテクチャ派生による再利用率低下

図 11 はプロダクトラインアーキテクチャの違反率 Ra の推移を表す．ここで，アーキテクチャ違反率 Ra は以下と定義する．

Ra ＝ A/B （4）

A：アーキテクチャ違反ソフトウェア部品数

B：ソフトウェア部品数

ここで，ソフトウェア部品におけるアーキテクチャ違反とは，機種間の差異の吸収方法が当初設計思想と異なるものを指す．

図 11 に示すように，2010 年の SPLE 導入以降，毎年機種固有の差異を吸収する仕組みに対する実装違反が検出されている．

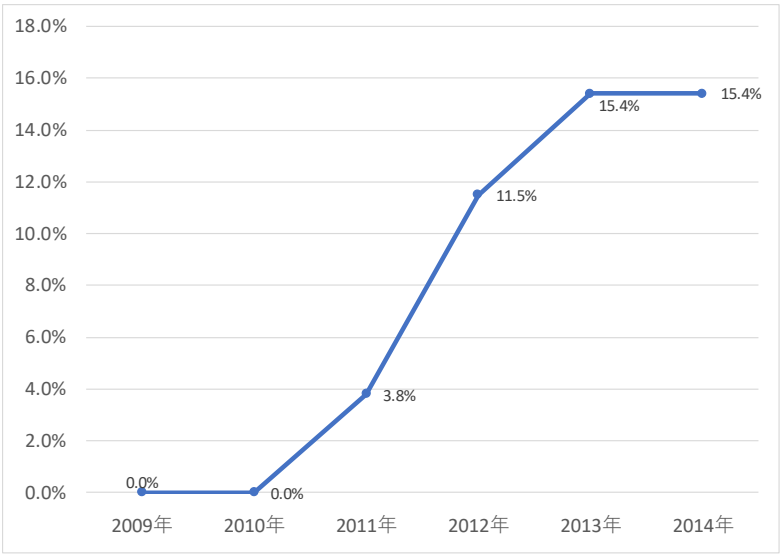

図 11 アーキテクチャ違反率 Ra の推移

図 12 は，マージされずにその後の開発のベースとして使用され続けたソフトウェア系列数を示すものである．Figure 12 において，同期間における SPLE のソフトウェア系列数が 1 年に 1 本増加し，適用ステージ 5 年間の間に 4 本になったことを示している．

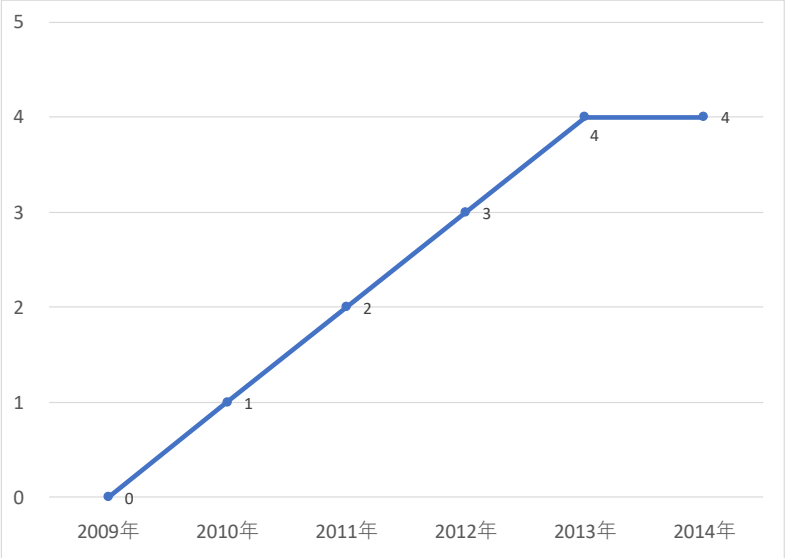

図 12 ソフトウェア系列数の推移

これは，機種固有の差異を吸収する仕組みが適切に使用されないまま開発が行われ，吸収できなくなった差異をソフトウェア系列として分けるという方策をとってしまったことを意味する(P4)．また，このほかにも並行開発に対応するためにソフトウェア系列を分けたというものもあった（P2)．このソフトウェア系列数の増加が，SPLE の製品に対するカバー率が上昇する一方で，製品群にまたがるソフトウェア再利用を難しくし，生産性低下を引き起こした一因であると考える．

ソフトウェア再利用率低下の 3 つの考察から，適用（第 1 次）ステージにおける生産性低下の原因を SPLE の 3 つの要件に対する実装の程度と照らして表 6 にまとめる．

表 6 SPLE 実装の程度と問題の対比表

重要取り組み	実装の程度	発生する問題
再利用資産の構成管理がなされている	プロダクトラインではない単一製品時の手法を踏襲	類似したソフトウェア部品が開発され再利用率が低下する（P1）
並行開発管理がされている	特段のプロセスを定義せず	ソフトウェア系列が派生する（P2）
再利用資産の保守・運用に関する管理がされている	①上流工程からの体系的な再利用を促すプロセス定義が不十分（仕様共通化率が低調） ②アーキテクチャの使い方や修正指針に関する浸透が不十分	・類似仕様が多発し，ソフトウェアの再利用率が上がらず効果が SPLE 本来の効果が得られない（P3） ・プロダクトラインアーキテクチャが派生する（P4）

以上の結果から，SPLE3 つの必須活動の実装の程度と生産性への影響について，以下のように考察する．

SPLE は製品群に対する開発手法であり，①製品間で要求仕様を共通化するプロセスの整備が有効に機能していないとソフトウェア部品の再利用率が低下する，②並行開発で生じたコア資産の分岐のマージを促進する構成管理プロセス，再利用を促進するために部品の選択と組立を支援するビルド環境ができていないと類似したソフトウェア部品が増え，再利用率が低下する，③製品間の差異を吸収するプロダクトラインアーキテクチャの使用方法，設計思想が徹底されていないとソフトウェア部品の再利用率が低下するばかりか，プロダクトラインアーキテクチャにまで影響を及ぼし，SPLE は機能しなくなってしまう．

本結果から，評価改善ステージでは具体的な改善対象として，以下の活動に取り組むべきという結論を得た．

- 仕様共通化プロセスを設け，個別機種開発において，仕様レベルで再利用可能な部品を準備すること
- ソフトウェアを部品単位で管理し，部品の選択と組立を支援する環境を開発するとともに，並行開発に対応可能な管理プロセスを設けること
- プロダクトラインアーキテクチャの使用方法，設計方針の共有だけでなく，機種間の差分を吸収するプロダクトラインアーキテクチャの違反を早期に検出し是正するアーキテクチャ監視を行うこと

第3章

コア資産保守・製品導出手法の改善

3.1. コア資産保守・製品導出プロセスの重要性

近年，空調機器やエレベータ，自動車等の組込み製品においては，他社製品との差別化のための製品機能の高度化，多様なユーザニーズへのタイムリな対応が求められている．そのため，これら製品向けのソフトウェア開発では，開発する機能を共有しつつ製品バリエーションに合わせて少しずつ異なる仕様を実現する複数の開発を，短期間に並行して行うことが多い（以下「多品種小変更型開発」と呼ぶ）．さらに，多品種小変更型開発は，厳しい企業間競争に勝ち抜くため，市場動向を見ながら柔軟に機能拡張することと，高品質を担保しつつ低コストで開発できることが求められている[23].

一般に，高品質と低コストを両立させるソフトウェア開発を実現するためには，なるべく既存のソフトウェアに手を加えず再利用することが肝要である[24][25]．このため，特定のドメインの製品群に対して，ソフトウェアの再利用を体系的に行う手法として，ソフトウェアプロダクトラインエンジニアリング（以下，SPLE）が提案され，広く実践されている[5][23][26].

SPLE は，3 つの活動 ①コア資産を開発・保守するドメイン開発，②コア資産をもとに製品を開発するアプリケーション開発，③これらを系統立って行うための管理プロセスから構成される．SPLE は，ドメイン開発とアプリケーション開発を分離することで，コア資産の再利用性の向上と，顧客向けアプリケーション開発の短納期化という 2 つの関心事を分離して実行・管理できる利点がある[24][27]．効率的にコア資産を保守し，製品を導出するためには，これら 3 つの活動を円滑に実施できる組織構造およびプロセス実装が成功の主要因であるとされている[28].

Klaus Pohl ら[22]は，これら 3 つの活動のプロセスの組み立て方や，組織構造に応じた実践方法を参照モデルとして紹介している．また，その成果は SPLE の参照モデルとして国際標準に採用されている[29].

筆者らは，Pohl らが紹介する参照モデルを参考に SPLE を導入し，2010 年から製品導出を実施した．その結果，一時的に製品群全体の生産性が向上したが，その後コア資産の再利用率が下がり生産性が低下傾向を示すようになった．この原因を 2 章および文献[6]に示した通り以下のように分析している．多品種小変更型開発では，製品群への要求は顧客に近いアプリケーション開発の担当組織がもっているため，ドメイン開発側主導で要求事項の分析を進めることが難しい．また異なる市場に対応する多数のアプリケーション開発担当組

織が，同時並行的に類似した要求仕様で開発を始めようとするため，実際にアプリケーション開発を実施するソフトウェア部門がソフトウェア部品を再利用するか新規に開発するのかの判断を誤る場合が増えてしまう．この判断誤りが，不必要な類似部品の発生を増やしコア資産を劣化させ，結果的に再利用率を落とす原因となった．

我々は，この原因分析に基づき，ドメイン開発の実施方法および，アプリケーション開発における要求仕様書の改善を図った．この改善手法は，①開発ロードマップからアプリケーション開発を 1 つの幹開発と複数の枝開発に分離，②幹開発の中で，個別製品の開発を行いつつ，後続する枝開発のバリエーションを吸収可能なソフトウェア部品とそのソフトウェア部品と関連付けた要求仕様書マスタを作成，③枝開発の中で，その要求仕様書マスタからのカスタマイズとソフトウェア部品の組立て，を実現するもので，上記の開発プロセスと要求仕様書の構成法からなる．ここで，ソフトウェア部品とは，コア資産の一部であり，ある機能単位にまとめられたソースコードを指す．

本開発手法を実開発に適用することで，判断誤りによる類似部品の発生を抑制し，結果，ソフトウェア部品の再利用率が 39.6 ポイント改善，製品群全体における生産性が 55.7 ポイント向上し，さらにそうした効果が持続できていることが確認できた．

3.2 節では，Pohl らが紹介しているコア資産保守・製品導出プロセスと，それに基づき実施した筆者らの第 1 期活動と課題について述べる．3.3 節では，2.2 節で述べた課題に対して，改善手法を提示し，効率的なコア資産保守・製品導出プロセス，及びそれを可能にする要求仕様書の構成法について説明する．3.4 節では，3.3 節で述べた手法の実践状況を述べるとともに，その効果や妥当性について検証する．

3.2. SPLE 参照モデルに基づく第 1 期活動と課題

3.2.1. SPLE 参照モデル

Pohl らは，図 13 に示すような，製品群に対する要求仕様から系統的にコア資産保守・製品導出するためのフレームワークを参照モデルとして紹介している[22].

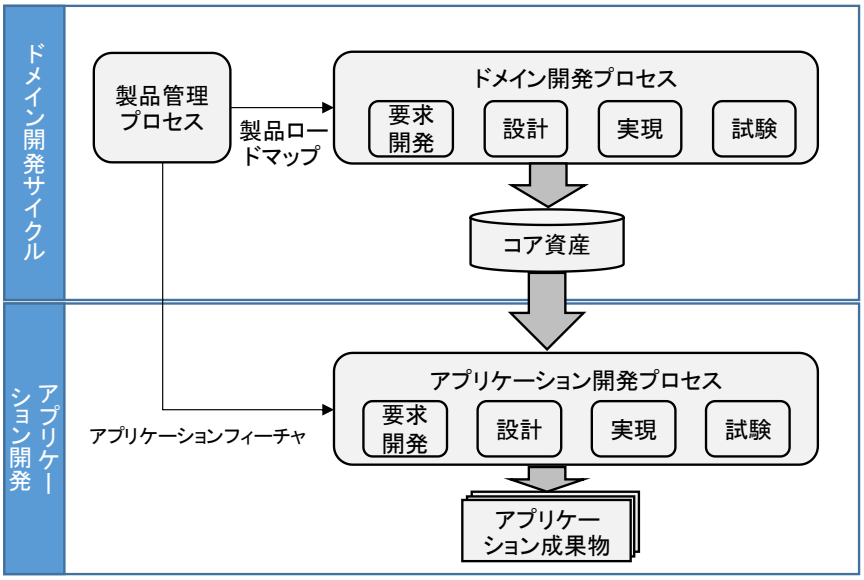

図 13 Pohl の SPLE フレームワーク

SPLE フレームワークは，製品群全体に対するコア資産を開発・保守するためのドメイン開発サイクル，及び特定の顧客や市場向けの製品を作るアプリケーション開発からなる．ここで，ドメイン開発サイクルは，製品管理プロセスとドメイン開発プロセスの継続的な繰り返しからなる．SPLE フレームワークは，ドメイン開発サイクルを個々のアプリケーション開発から分離することで，安定したコア資産を構築・保守し，製品群全体の開発コストを下げることを狙っている．

図 13 に示すように，製品管理プロセスは，製品群全体が対象とする顧客・市場の要求や経営戦略に基づき，製品ロードマップを定義する．製品ロードマップとは，製品群のリリース計画だけでなく，主要な共通フィーチャや可変フィーチャ，各製品がもつべきフィーチャ(アプリケーションフィーチャ)を含む．ドメイン開発プロセスでは，製品ロードマップをもとに，製品群の共通性と可変性を抽出し，個々の製品の開発に利用可能なコア資産を作り出す．アプリケーション開発では，ドメイン開発において開発したコア資産を活用し個別製品の開発を行う．

SPLE フレームワークに基づく組織構造およびプロセス実装の要点は以下の 3 つである．

①ドメイン開発サイクルへの開発組織の割当て

ドメイン開発サイクルへの組織の割当てについては，階層型組織，マトリクス型組織それぞれでいくつかのバリエーションが紹介されている．例えば，階層型組織構造における集中型ドメイン開発では，アプリケーション開発とは異なる独立した組織でコア資産を開発し，分散型ドメイン開発では，各製品開発組織内にドメイン開発を割り当てる．いずれにおいても，ドメイン開発と製品群全体に対する責任分担とを統合できることが重要である[22].

②ドメイン開発での共通性と可変性の組込み

ドメイン開発サイクルで製品の共通性と可変性の組込みを行う．プロセスの実装の仕方にはいくつかのバリエーションがあり，市場ニーズや技術の将来予測に基づいて先を見越して共通性と可変性を識別しコア資産の開発を行う方法[22]と，市場ニーズに基づく変更要求に受動的に対応しコア資産を発展させていく方法[30]とがある．

③コア資産の最大限の再利用

アプリケーション開発においては，ドメイン開発で開発したコア資産を利用することで，品質および生産性が高い開発を行う．このため，アプリケーション開発では，ドメイン開発であらかじめ開発したコア資産の利用率を高め，判断誤りによる類似部品の発生を抑制することが重要である．

3.2.2. 第 1 期活動の実践と課題

3.2.2.1. 多品種小変更型開発とその特徴

空調機器やエレベータ等の多品種小変更型開発の特徴を以下に示す．

①一定期間（例：3 か月～1 年）の開発サイクルで機能・性能向上のための製品群の更新を行う．その際ハードウェアとソフトウェアの開発が並行する．

②製品の商品価値がハードウェアにあるため，製品群の機能・性能の仕様及び開発計画は，機械（メカ）や電気（エレキ）部門が決定する[31]．ソフトウェア部門は仕様に従いソフトウェアを開発する．

③ハードウェア優先の製品開発であり，性能目標達成や市場動向の変化への対応のため，ハードウェアの仕様確定の遅れや変更が多い．その影響によりソフトウェアの仕様確定も遅れがちで，決定後も変更が多い．

④製品群は，ソフトウェアからみて，以前の開発サイクルの製品群の仕様の多くを共有し，また同一開発サイクルで開発する新機能・性能改善のための仕様も共有している．

⑤製品群は，上記共通仕様をベースにして，特定用途や特定の出荷地域向けにカスタマイズする必要があり，同一開発サイクル内におけるカスタマイズ開発数が多い(例：10～20 件)．

⑥個々のカスタマイズ開発は小規模であり，短期に並行して実施される．

3.2.2.2. 第 1 期活動における SPLE フレームワークへのマッピング

SPLE フレームワークに，多品種小変更型開発のメカ・エレキ部門及びソフトウェア部門を，割り当てようとすると，特徴②より以下の対応関係となる．

表 7 SPLE フレームワークと対応部門

SPLE フレーワーク		対応する部門（役割）
ドメイン開発サイクル	製品管理プロセス	メカ・エレキ部門
	ドメイン開発プロセス	ソフトウェア部門（要求・設計・実現・試験）
アプリケーション開発		メカ・エレキ部門（要求） ソフトウェア部門（設計・実現・試験）

メカ・エレキ部門が，製品管理プロセスを担い，製品群の個々の製品がもつべきフィーチャの決定と，その市場投入時期を決定する．

個々の製品開発に対して，メカ・エレキ部門から担当チームが割り当てられる．各担当チームは，製品管理プロセスで決定したアプリケーションフィーチャを実現すべく，個別に要求仕様をまとめる．ソフトウェア部門は，メカ・エレキ部門の担当チームより要求仕様を受け，アプリケーションの開発を担う．

ドメイン開発は，ソフトウェア部門がプロジェクトチームを立てて担い，製品計画に沿ってコア資産の開発や保守を行う．このとき，ドメイン開発はリアクティブにコア資産を開発・保守するアプローチを採用している．なぜならば，競合の多い製品ドメインにおいては，開発要求は変更されやすく，プロアクティブなアプローチでは開発したコア資産が利用されない可能性が高まるためである．

アプリケーション開発では，ドメイン開発で開発したコア資産をもとに，製品ソフトウェアを組み上げることを志向した．

3.2.2.3. 第 1 期活動における課題

第 1 期活動では，2 章および文献[6]で述べた通り 2 つの課題があった．以下にそれらの課題を整理して示す．

[課題 1] 要求仕様からソフトウェア部品再利用へ変換ミス

アプリケーション開発（要求開発）において，メカ・エレキ部門の担当チームは，ソフトウェア部門が保守するコア資産を構成する参照アーキテクチャとソフトウェア部品に関する知識を持たずに，要求仕様書を記述する．

ソフトウェア部門は，そうした要求仕様を解釈し，可変点の識別に基づき，利用可能なソフトウェア部品の開発をする．しかし，複数の開発が並行するため，要求仕様の解釈時に，可変点の見逃しが頻繁に起き，結果的に利用可能なソフトウェア部品を利用できずに不必

要な開発が増加してしまう.

[課題 2] ソフトウェア部品の仕様不適合

製品ロードマップ作成時点では，製品群のラインナップ，並びに各製品の持つアプリケーションフィーチャおよび可変点は決まっていても，可変点に対応するバリアントであるソフトウェア部品の具体的な仕様は，ハードウェアの詳細な仕様が決まるまで確定しない．極端な場合，その状態は対象となる新機能を盛り込んだアプリケーション開発の完了間際まで続く．

このため，アプリケーションの開発以前に，ドメイン開発（特に設計から試験）を実施しようとすると，仕様を仮決めして，ソフトウェア部品を作ることになり，作成したソフトウェア部品は，アプリケーションにそのまま利用できない場合が多くなる．すなわち，アプリケーション開発で個別に修正する事態が起こり易くなる．

要求変更が多い開発には受動的に対応するアプローチ[9]が適していると思われるが，特徴⑤により同一開発サイクル内におけるカスタマイズ開発数が多い場合，統制なくすべての要求変更に受動的に対応することは困難である．

課題 1 の要求仕様からソフトウェア部品への変換ミス及び課題 2 のソフトウェア部品の不適合が頻繁に発生することにより，コア資産であるソフトウェア部品の再利用率が低下し，類似したソフトウェア部品が増加する状況が生じる．

3.3. 第 2 期活動におけるコア資産保守・製品導出プロセスの改善

多品種小変更型開発に対する 2 つの課題を解決するためのコア資産保守・製品導出手法を改善した．改善手法は，開発プロセスおよび要求仕様書の構成法からなる．

3.3.1. 開発プロセス

改善した開発プロセスを図 14 に示す．本プロセスは，対象製品群の同一販売期間向けの開発サイクルに対応するもので，1 開発サイクルは 1 つの製品管理プロセス，1 つの幹開発プロセス，および複数のアプリケーション開発プロセス（枝開発）からなる．

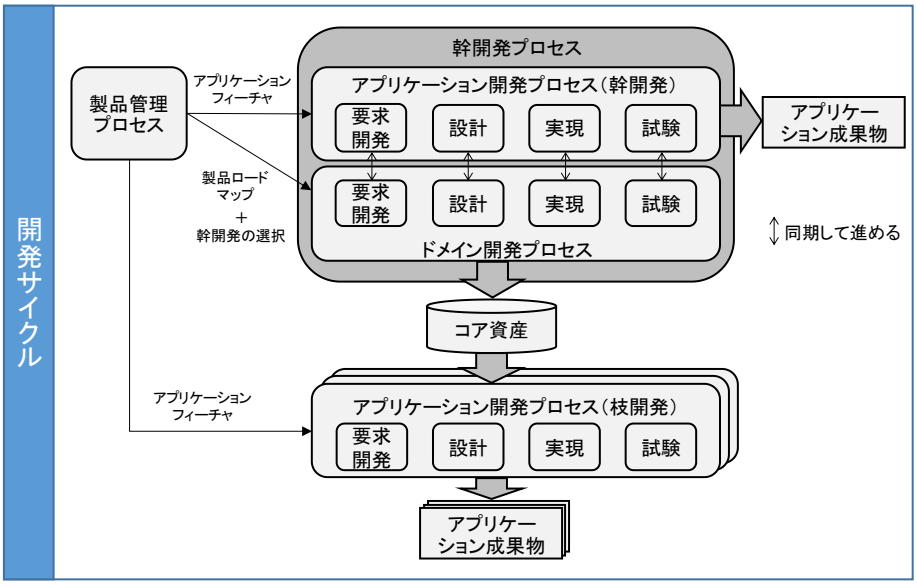

図 14 改善した開発プロセス

3.3.1.1. 開発プロセスの詳細

改善した開発プロセスの詳細と，多品種小変更型開発の 2 つの課題の解決について以下に示す.

(1) 製品管理プロセス

本プロセスの主要な目的は，開発サイクル（例えば 1 年間）の製品開発計画を定めることと，計画された製品の中から幹開発の選定を行うことである.

製品管理プロセスでは，メカ・エレキ部門が，市場のニーズや，他社動向，販売戦略等に基づき，長期の開発計画，及び当該開発サイクルの詳細な開発計画を定める．当該開発サイクルの詳細な開発計画には，製品ラインナップ，開発時期，主要な新機能や性能目標，ハードウェアの構成を記述する.

これらの情報に基づき，当該開発サイクルの開発対象製品群の中から，以下の条件に適合するものを幹開発対象として選定する：

①開発する新機能数が最多また次点のもの

②当該開発サイクルで新たなハードウェアを搭載するもの

③本来は枝開発であるが，幹開発と同時に開発ができるもの（ハードウェアが幹開発と同等であるもの）

幹開発対象製品以外は枝開発対象製品とする.

(2) 幹開発プロセス

幹開発プロセスの目的は，選定した具体的な製品のアプリケーション開発（幹開発）を進めることと，後続するアプリケーション開発（枝開発）のために当該開発サイクルで利用するコア資産の保守を行うドメイン開発を実施することである.

幹開発プロセスは，この目的のために，メカ・エレキ部門の当該製品担当者とソフトウェア部門との混成チームを構成し，ドメイン開発プロセス及びアプリケーション開発プロセスにおける要求開発・設計・実現・試験を，同時並行で進めていく．当該プロセスの入出力を以下に示す．

入力：製品ロードマップ，
　　アプリケーションフィーチャ，コア資産
出力： 幹開発対象製品（アプリケーション成果物），
　　保守されたコア資産（要求仕様書マスタ及び開発したソフトウェア部品を含む）

当該製品の具体的な要求仕様と製品ロードマップより分析・抽出した可変性に基づき，可変点に対応するバリアント(ソフトウェア部品)を識別する．例えば，可変点のフィーチャ「圧力保護」が，幹開発は圧力スイッチによる検出，別の枝開発では異なるハードウェア構成（圧力スイッチが付かない等）の製品の開発が枝開発で計画されているとする．この場合，スイッチありとスイッチなしの両方に対応したソフトウェア部品を識別する．

識別した可変点及びソフトウェア部品を盛り込んだ要求仕様書マスタを作成する．要求仕様書マスタは，3.3.2 項で詳述するように，アプリケーション開発（幹開発）の要求仕様書であると同時に，可変点と識別されたバリアントリストを組込み，当該開発サイクルの対象製品全体に対応した雛形となっている．

識別したソフトウェア部品の仕様の決定と実装は，アプリケーション開発の設計・実現時に並行して行う．

このように，改善後のプロセスでは，幹開発製品を対象とした具体的なアプリケーション開発と，ドメイン開発を同期して進めることができるので，ソフトウェア部品の仕様の確度が向上する．さらに，並行して開発するハードウェアの仕様の決定遅延や変更は，幹開発プロセスの中で吸収できる．これらから，提案プロセスにより，課題 2「ソフトウェア部品の仕様不適合」の問題を解決することができる．

(3)アプリケーション開発プロセス（枝開発）

アプリケーション開発プロセス（枝開発）の目的は，コア資産であるソフトウェア部品を再利用し，枝開発対象製品のアプリケーション開発を高品質かつ低コストで実施することである．

アプリケーション開発プロセス（枝開発）は，従来と同様，メカ・エレキ部門の担当チームが要求仕様書を作成し，その仕様書に基づき，ソフトウェア部門が開発を実施する．当該プロセスの入出力を以下に示す．

入力：アプリケーションフィーチャ，
　　コア資産（要求仕様書マスタ及び開発したソフトウェア部品を含む）
出力：枝開発対象製品（アプリケーション成果物）

アプリケーション開発用の要求仕様書は，コア資産の要求仕様書マスタをベースとして，設定された可変点からバリアントを選択し，必要なパラメータを設定することで作る．それにより，バリアントに紐づけられたソフトウェア部品の再利用が確実に実施できるようになる．

要求仕様書マスタにコア資産の再利用を考慮した可変性を直接織り込んであるので（詳細は 3.3.2 項），課題 1「要求仕様からのソフトウェア部品の変換ミス」が減り，コア資産を有効に再利用することができるようになる．

3.3.1.2. 開発プロセスの実施例

図 15 は，提案する開発プロセスを一実施例として時間軸に投影したものである．

製品開発プロセスで選択された幹開発が行われ，具体的な製品開発とともに，当該開発サイクルで利用可能なソフトウェア部品と要求仕様書マスタが開発され，コア資産として登録される．それに続いて，登録されたコア資産を用いた，特定用途向けや出荷地域ごとのカスタマイズを行うアプリケーション開発（枝開発）が，開発ロードマップに沿って，順次並行して行われていく様子を表している．

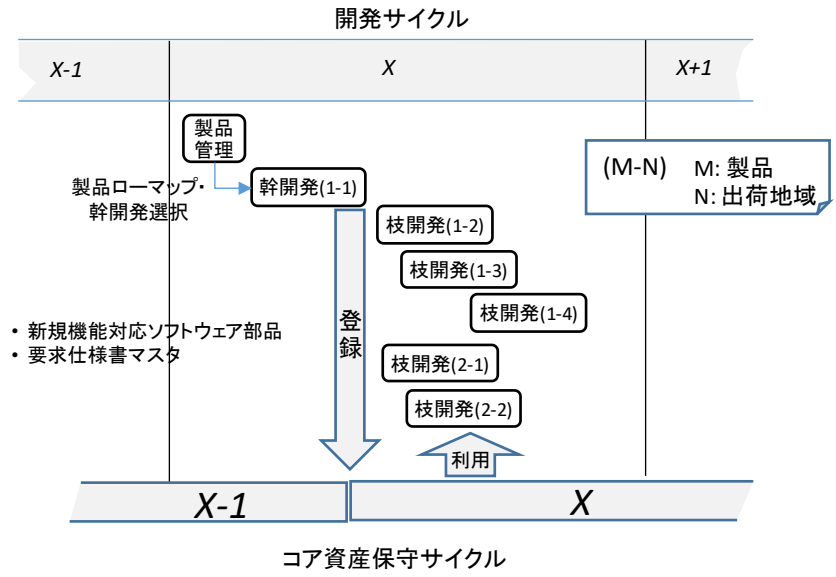

図 15 改善プロセスの実施例

3.3.2. 要求仕様書の構成および運用方法

3.3.2.1. 要求仕様書の構成に求められる要求事項

3.3.1 項の開発プロセスが有効に機能するためには，要求仕様書が以下 3 つの要求事項を

満たす必要がある.
R1)製品群で必要とされる可変性を表現できること.
R2)アプリケーション開発の要求開発を担当するメカ・エレキ部門の要員にとって理解しやすいものであること.
R3)要求仕様書とソフトウェア部品の間のトレーサビリティが確保できること.

改善手法は，これらの要求事項を満たすように，次に示す要求仕様書の構成法を用いる.

3.3.2.2. 可変性の分析

筆者らは，多品種小変更型開発に本改善手法を実装する以前に，5 年にわたる SPLE 実適用を経ている[6]. その適用結果の分析から，要求事項の変化は概ね以下の 3 つに分類できることがわかった.
①機能の有効／無効
②デバイス制御方式の差異（同一目的・同一機能だが，デバイス違いから検出方法や演算方法が異なるもの）
③ハードウェア・出荷地域違いによるパラメータ差分

これら 3 種類の要求事項の変化は，ソフトウェア的には以下の 2 種類の可変性によって吸収できることがわかった.
・ソフトウェア部品の選択（①，②に対応）
・パラメータ値の設定（③に対応）

3.3.2.3. 要求仕様書の構成

改善手法では，上記 2 種類の可変性を要求仕様レベルで選択できるような要求仕様書の構成することによって，要求事項 R1 の達成を図った.

図 16 に提案する要求仕様書の構成を示す. 要求仕様書は制御仕様書とデータ仕様書の 2 つの文書からなる. 制御仕様書は，製品群に搭載されるすべての機能について記述し，各機能の開始条件や制御内容等の仕様が記載された製品群全体で共有される文書である. 後述するように，すべての機能を記述したうえで，製品ごとに搭載される機能を明確化している. データ仕様書は，各製品における具体的なパラメータの設定値が記載された文書を指し，製品ごとに持つ.

(1) 制御仕様書

制御仕様書は，システム構成と製品機能の 2 つの章をもつ.「システム構成」は，製品－機能マトリクスによって表現される. 製品－機能マトリクスとは，縦軸に製品群に搭載されている全機能，横軸に各製品がそれぞれ列挙され，交点に各製品での機能の有効／無効が明示された表のことである. 機能を有効／無効をこのマトリクス上で選択することにより，

3.3.2.2 目に記載の要求事項の変化①に対応することができる．

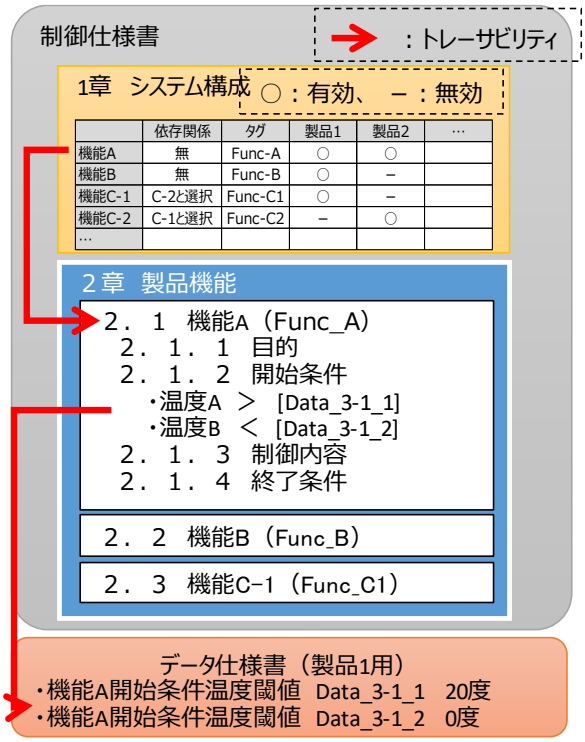

図 16 要求仕様書の構成

縦軸の機能一覧においては，制御方式が異なる類似機能を依存関係付きの別機能として扱う．類似機能とは，機能的な要求は同じでハードウェア構成等の事情により制御が異なるバリアントであり，3.3.2.2 目の要求事項の変化②に対応する．例えば，同じ「保護機能」でも，圧力センサで直接圧力を測るものと，圧力センサの代わりに温度を測定することにより圧力を推定するものとがあるなどである．こうした機能は，代替の関係にあるので，図 4 の製品－機能マトリクスの C-1 と C-2 のようにいずれかの選択であることを明示する．このようにメカ・エレキ部門には類似機能の存在とバリエーションを知らせている．

「製品機能」は，当該製品が持つ機能を，開始条件や制御内容等の項目について記述する．「システム構成」において一覧化されたすべての機能をここに記述する．これらの機能は（Func_A）のようなタグを有しており，下流工程の文書およびソースコードの該当箇所には本タグが付与されている．これによりソフトウェア部品と明示的に紐づけられるため，上流工程での変更を下流工程に誤りを混入させることなく伝達することが可能となる．

上記の章構成および記述方法により，開発済みの機能の存在とその仕様を知ることができる．

(2) データ仕様書

製品機能の中には，同一の仕様であっても製品ごとにパラメータが異なるものが多い．これらの差異は，データ仕様書に記述する．具体的には，制御仕様書の本文中には[Data_3-1_1]のようなタグを記述し，このタグと同一のタグをデータ仕様書に記述することで，トレーサビリティを確保する．外出しするチューニングデータは，能力帯，出荷地域，ハードウェア構成，製品シリーズを含む．

データ仕様書により，3.3.2.2 目の要求事項の変化③に対応する，製品別のパラメータ差分を吸収する箇所が特定でき，また製品ごとのデータ部品を自動生成することで，パラメータ違いのみでソースコードを修正するような無駄な開発を防ぐことができる．

3.3.2.4. 幹開発と枝開発における要求仕様書の利用方法

改善手法では，幹開発プロセスにおいて，提案する要求仕様書を作成する．これを要求仕様書マスタと呼ぶ．

幹開発プロセスは，①要求仕様書マスタの作成，②可変点とそのバリアントである「機能」に対応するソフトウェア部品の作成，③機能とソフトウェア部品のリンク付けを行う．要求仕様書マスタは，幹開発対象製品の要求仕様書でもある．要求仕様書マスタとソフトウェア部品は，コア資産に登録する(図 14)．これにより，要求事項 R3「要求仕様書とソフトウェア部品の間のトレーサビリティを確保」の達成を図ることができる．なお，この作業は，幹開発プロセスに割り当てられたメカ・エレキ部門の幹開発製品の担当者とソフトウェア部門との混成チームで行うことで，適切に行うことができる．

アプリケーション開発（枝開発）では，要求仕様書マスタをカスタマイズすることで，個別の要求仕様書を作成する．各アプリケーション開発プロセス（枝開発）において，要求仕様書マスタの「システム構成」の担当製品に対応する縦の欄を完成させ，当該枝開発用のデータ仕様書に能力帯や出荷地域ごとのパラメータ設定を記載する．

2 つの可変性を組み込んだ要求仕様書を利用することにより，各アプリケーション開発プロセス（枝開発）において，メカ・エレキ部門が，要求仕様書を個別に作らず，個別製品の要求事項を，選択式で行うことができる．これにより，要求事項 R2「メカ・エレキ部門の要員にとって理解しやすいものであること」の満足を図る．

R1-3 を満たす要求仕様書を用いることにより，提案プロセスを用いて，要求仕様から変換ミスなくソフトウェア部品をより確実に活用することが可能になる．

3.4. 適用と評価

3.4.1. 適用対象，経緯および方法

典型的な多品種小規模型開発である空調機（室外機）を対象として，提案するコア資産保守・製品導出手法を適用・評価を実施した．

(1)適用対象

空調機（室外機）は，日本国内のみならず世界各国へと輸出・展開している典型的な多品種小変更型の製品群である．製品群は，以下のバリエーションをもっている．

・能力帯（顧客層に対応）
・出荷地域別／能力帯別のハードウェア構成の違い
・特定用途向けの機能の搭載の有無

1 年を 1 開発サイクルとして，機能・性能の向上を実現する製品群のソフトウェア開発を実施する．上記のバリエーションを持つ製品群を 30 製品程度開発する．各製品の開発は，小規模でシステム試験まで含んで 3～6 ヶ月程度の工期が設定されている．これらの開発の多くは，前年度製品群からの機能の流用率が 90%と高い．

開発組織は，メカ・エレキ部門，ソフトウェア部門からなり，メカ・エレキ部門がハードウェアと各製品の開発，ソフトウェア部門がコア資産の開発・保守，及び各製品のソフトウェア開発を実施する．

(2)適用の経緯

適用対象に対して，2 章及び文献[6]で示した通り，以下の 3 段階で実施している．

①準備段階（2006～2009 年）：ドメイン開発を実施し，コア資産を構築した段階．プロダクトラインアーキテクチャを開発しソフトウェア部品をコア資産として整備した．

②第 1 期活動（2010～2014 年）：準備ステージで構築したコア資産を用いて，個別製品を本格的に開発した段階．Pohl らが紹介する参照モデル[7]をベースに，コア資産の保守・製品導出を行った．

③第 2 期活動（2015 年～）：第 1 期活動の結果，生産性と再利用率の低下があり，その原因分析に基づき改善手法を適用した段階．

適用対象ドメインは，SPLE 準備段階以前から国内に安定したビジネス基盤をもっており，製品群の品質もすでに高い状態であった．第 1 期から第 2 期にかけ海外市場への製品展開が増加していくことが予想されており，これに伴い，ソフトウェアの開発規模の増加，それに伴う開発人員の質と量の確保と同時に，従来と同様の品質を維持すること求められていた．

(3)適用方法

上記③第 2 期活動において，改善手法を実組織において以下のように実装した．

①メカ・エレキ部門とソフトウェア部門が，開発サイクルの最初にワーキンググループを作り，開発ロードマップに沿って幹開発となる対象製品を決定する．具体的には，国内市場向けの性能・機能における最上位製品を幹開発とし，国内市場向けの廉価製品や海外市場向けの製品を枝開発とした（図 15）．

②幹開発は，幹開発の担当者とソフトウェア部門のドメイン開発担当者がチームを組み，対象製品の開発を進めつつ開発サイクル内で共有する機能，およびバリエーションを実現するための要求仕様書マスタ＋ソフトウェア部品群を開発していく．幹開発を相対的に開発費の多い国内市場向けの最上位製品としたため，コア資産の開発や保守のコストおよび要員は，幹開発の担当部門が受け持つ．
③枝開発は，要求仕様書マスタを，メカ・エレキ部門の担当者がカスタマイズして枝開発対象製品の要求仕様書を作り，ソフトウェア部門がその仕様書に基づきソフトウェアを開発する．枝開発の途中で，想定してなかった仕様変更があり，新たな実装が行われた場合には，要求仕様書マスタとソフトウェア部品への追加を実施し，コア資産の保守を行う．

3.4.2. 適用前後の比較と評価

上記に示した適用対象，経緯，及び方法で，本手法を，2015 年度開発製品より適用を実施した．適用開始直前の 2014 年度の各指数を 100 として，各指標の推移を示し，適用の効果を示す．各年度のプロットは，各年度の開発で得られたデータの平均値とした．

3.4.2.1. ソフトウェア部品の再利用率

多品種小変更型開発への SPLE 適用上の課題 1「要求仕様のソフトウェア部品への変換ミス」，すなわち開発／再利用の判断ミスの解決を見るため，ソフトウェア部品の再利用率を評価する（図 17）．
ソフトウェア部品再利用率 Rp は次式と定義する．

$$Rp = Nr / Np \qquad (1)$$

Nr：複数製品で使用されているソフトウェア部品数
Np：コア資産に登録されているソフトウェア部品数

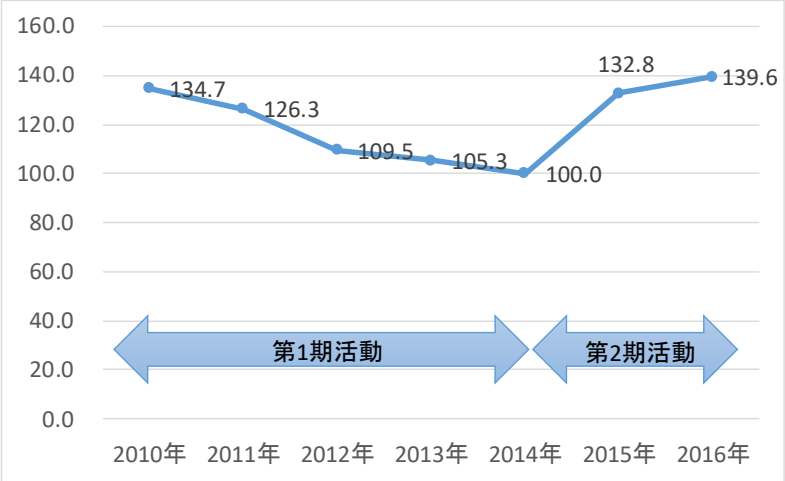

図 17 ソフトウェア部品再利用率の推移

(1)適用前と適用後の比較

図 17 に示すように，ソフトウェア部品再利用率は，第 1 期活動では，2 年目より，徐々に低下し 4 年間で 34.7 ポイント低下しているのに対し，本手法適用段階では，対 14 年度比で 15 年度に 32.8 ポイント，16 年度は 39.6 ポイントの改善が見られる．

(2)ソフトウェア部品への変換ミスの防止効果の評価

第 1 期活動時は，ソフトウェア部品の増加に伴い，再利用率の低下が見られた．本手法の適用後は，提案する要求仕様書の導入により，ソフトウェア部品の変換ミスが減り，再利用率が高まっていると考える．また，この時期開発人員が増加しており，慣れによる習熟度の向上が見込めない状況であったことを加味すると，本手法の有効性が評価できる．さらにこの効果が 2 年目も高い水準を維持できていることから，ソフトウェア部品数が増加しても，変換ミスを防止する効果が維持できる手法と言える．

3.4.2.2. 類似部品発生率

多品種小変更型開発への SPLE 適用上の課題 2「ソフトウェア部品の仕様不適合」，すなわちソフトウェア部品がそのまま使えない状況の発生という課題の解決を見るために，類似部品発生率（ソフトウェア部品 1 つ当たりのブランチ発生数）を評価する（図 18）．
類似部品発生率 Rs は次式と定義する．

$$Rs = Ns / Np \qquad (2)$$

Ns: 同一ソフトウェア部品からのブランチ数の総和
Np：コア資産に登録されているソフトウェア部品数

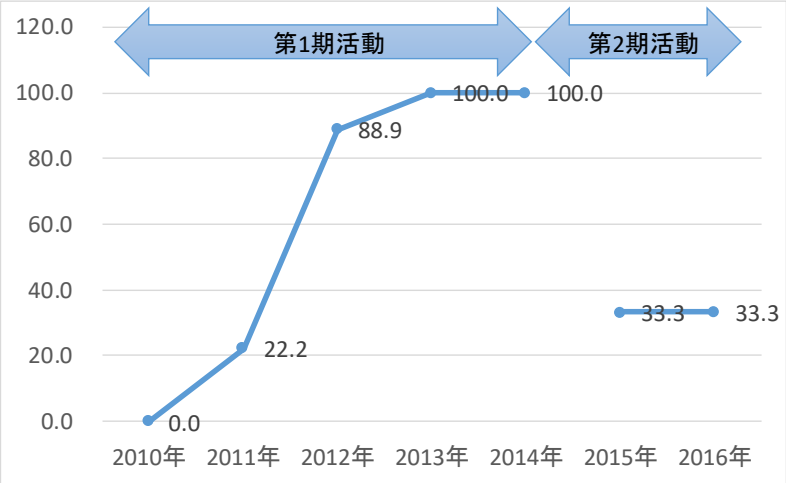

図 18 類似部品発生率の推移

(1)適用前と適用後の比較

図 18 に示すように，類似部品発生率は，第 1 期活動段階は，年々増大していった．本手法適用段階では， 15 年度から 16 年度はブランチ数の増加はなかった．なお，15 年度のブランチ数が 33.3 ポイントに減少したのは，活動の見直しに合わせて，類似部品を統廃合したことによるものである．

(2)ソフトウェア部品の仕様不適合の防止効果の評価

第 1 期活動時は，登録したソフトウェア部品をそのまま使えず，類似したソフトウェア部品を多数派生させてしまっていた．本手法適用後は，ソフトウェア部品の仕様の確度が上がったことにより，ブランチの発生数がなくなり，ソフトウェア部品をそのままの状態で利用する場合が増えていることを示しており， ソフトウェア部品の仕様不適合が軽減できる手法であることがわかる．市場の拡大に伴い，細かなバリエーションが増加しているなか，類似部品を発生させなかったことは，本手法の効果であると考える．

3.4.2.3. 製品群全体の生産性

本改善手法が製品群全体に与える効果をみるために，製品群全体の開発の生産性を評価する．

生産性 P は次式と定義する．

$$P = S / C \qquad (3)$$

S: 当該年度で開発した新規・改造コードの規模+

Σ（各ソフトウェア部品のコード規模×再利用数）

C: Σ（開発プロジェクトのソフトウェア開発費用）

図 19 に製品群全体における生産性の推移を示す.

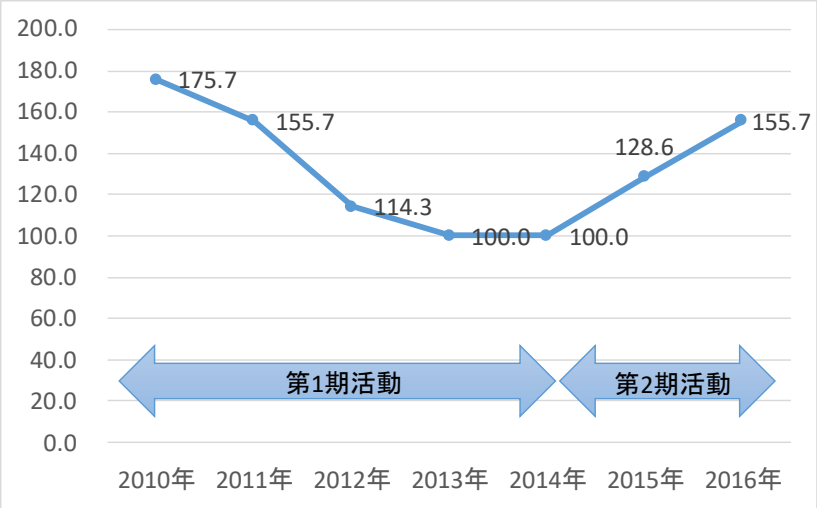

図 19 生産性の推移

(1)適用前と適用後の比較

図 19 には，第 1 期活動段階である 2010 年度からのデータも示している．この段階は，コア資産の保守や製品導出が適切に行われず，適用 2 年目より生産性の低下が見られ，4 年間で 75.7 ポイントの低下があった.

第 2 期活動段階においては，生産性が，1 年目の 2015 年度は 2014 年度比で 28.6 ポイント，2 年目の 2016 年度は 55.7 ポイントの改善があった.

(2)改善手法の生産性に与える効果の評価

第 1 期活動段階は，ソフトウェア部品の増加と修正によるコア資産劣化により再利用率が下がり，その結果生産性が下がり続けていた．本改善手法を適用することで，部品の再利用率が上がり，結果として生産性の改善の効果があった.

事業上の目標である製品群全体の開発コストの低減は，生産性の向上により達成することができた．第 1 期から第 2 期にかけ急激に年間開発数が増え，第 1 期の 10 件程度から 30 件以上に増加したが，開発リソースの調達に破綻なく柔軟に対応することができた．また，製品の品質については SPLE 導入以前から問題がない状況ではあったが，年間開発数の増加の中で同様の品質を維持することができた.

3.4.3. 考察

3.4.3.1. 改善手法の有効性の評価

3.4.2 項に示した適用結果より得た改善手法の有効性に関する評価を以下にまとめる.

・多品種小変更型開発の課題 1「ソフトウェア部品への変換ミス」については，ソフトウェア部品再利用率が向上・持続できていることから，提案する要求仕様構成法は，この課題を解決できていると評価する．
・多品種小変更型開発の課題 2「ソフトウェア部品の仕様不適合」については，類似部品発生率が抑制できていることから，提案する開発プロセスがソフトウェア部品の仕様の確度を上げる効果があり，この課題を解決していると評価する．
・製品群全体について生産性が改善し，2 年目も効果が持続していることから，本手法が多品種小変更型開発に対して持続性のある手法となっていると評価する．

3.4.3.2. 妥当性に対する脅威

本手法は，製品要求を決定するメカ・エレキ部門と，コア資産を開発するソフトウェアの部門が分離された組織を対象とすることを前提としている．その他の組織構造での有効性については未評価である．

また，適用対象は，開発規模の小さい，流用率が高い製品群の並行開発を対象としている．並行開発数，流用率や開発規模の範囲に関して本手法の適用上の限界がどこにあるか，不明である．

本改善手法の要求仕様書の構成法は，機能選択とパラメータ変更の 2 種類の可変性表現とその実装方式に強く依存している．この特徴は，2 章および文献[6]に示した通り，空調機分野において，2006 年から進めた SPLE の導入・適用活動の評価から発見した事項である．他の組込みドメインにおいても適用可能性は高いと考えるが適用評価は未実施である．

空調機の分野においても，新しい機能を作り出すアクチュエータの採用など大きな変更が入る開発の場合があり，その場合プロトタイピングなどの別の開発プロセスと組み合わせ対処する必要がある．

3.5. 関連研究

本章で課題としたコア資産の再利用や要求仕様書への可変性の組込みについて他研究論文との差異を整理する．

3.5.1. コア資産再利用のための組織的課題

組織が，SPLE 開発へ移行する際に生じるリスクを軽減するためのアプローチに関する研究や実践事例は数多く報告されている．Schmid らは，プロジェクト統合型というアプローチを提唱している[32]．プロジェクト統合型は，組織内で並走する複数のプロジェクト成果物をマージすることによりコア資産を開発するものである．また，Rubin らも先行製品からの差分開発によって SPLE 移行を進めている事例を報告している[33]．

これらのアプローチは，SPLE 導入移行時のコア資産の開発と利用を対象にし，移行期特

有のリスクやコストの低減の問題を扱っているのに対し，改善手法は，一定期間の開発サイクルに合わせ継続的に行うコア資産の保守と製品導出を対象としている点が異なっている．

Fogdal らは，調整部門を設け，コア資産の利用を促進する手法を提唱している[34]．調整部門は，並行する独立した開発プロジェクトに横断する形で設けられた組織であり，開発は行わずコア資産の周知・再利用の促進のみを行う．これらの取り組みによりコア資産の利用率が向上したことが報告されている．Fogdal らの取組みは，利用可能なコア資産の見逃しが多い部門において有効な手法と考えられるが，多品種小変更型開発においては，コア資産の利用可能性がアプリケーション開発の序盤で判断ができない（アプリケーション開発の完了間際までソフトウェア部品の仕様が確定しない）ため調整のみで再利用を促すことが難しい．

Adam らは，統合アプリケーション開発と呼ばれる手法を提唱している[35]．これは，各アプリケーション開発において，製品開発プロセスで作られた開発戦略を参照しコア資産を開発・登録するものである．並行する開発が多い多品種小規模開発には，コア資産の構成管理が複雑になるため，適さないアプローチと考える．

3.5.2. 要求仕様書への可変性の組込み

SPLE の製品導出において，各アプリケーションの製品機能を，段階を設けて構成していく段階的構成(Staged configuration)が有用な手法として知られている[36]．提案する要求仕様書の構成法は，機能の有効／無効及び制御方式の差異，並びにパラメータ差分の 2 段階による段階的構成を採っていると見ることができる．

SPLE では，共通性と可変性を，モデル図を使って表現し，ドメイン分析，コア資産設計，製品導出のそれぞれの局面で利用する事例が多い．モデル図としては，フィーチャモデル[37]やその発展形，それ以外のクラス図[38]などが提案されている．ステークホルダに応じて様々なモデルを活用する場合が多いが，多品種小変更型開発においては，直接の要求の出し手である製品部門が理解しやすいモデルであること，および直接的に可変性を選択し製品導出を実現できること，が重要である．つまり，フィーチャに基づきバリアントを選択するのではなく，バリアント（具体的な仕様）を直接選択するシンプルさが満たされる点に本改善手法の優位性がある．

表形式で可変性の表現する方法も多数ある．Stoiber らはフィーチャモデルと決定表の 2 つの表記を用いて製品導出を支援する方法を提唱している[39]．製品とフィーチャの星取表(Product-feature matrix)を用いて製品管理プロセスにおけるスコーピングを支援するもの[40]や，可変性を分析する手法[41]などがある．製品導出を行う際に利用する手法として Stoiber らのフィーチャモデルと決定表を用いる手法は，複数の表記法を活用して製品導出を行うためソフトウェアを解さない製品部門にとって扱いが難しいが，本改善手法では，製品と機能の星取表で直接的に表現しているためソフトウェアを解さない製品部門であって

も製品導出が容易である.

3.6. コア資産保守・製品導出プロセスのまとめ

本章では，多品種小変更型開発を対象に，要求仕様からソフトウェア部品への変換（開発／再利用の決定）誤りを軽減するため，コア資産保守・製品導出手法を改善し実践した.

この改善手法は，製品ロードマップに基づきソフトウェア再利用戦略上重要な個別製品開発を幹開発として選定し，当該製品開発におけるアプリケーション開発と同時に後続する製品（枝開発）の可変性を設計・実装することでコア資産を開発し，後続製品での再利用を促進する開発プロセスと，それを可能にする要求仕様書の構成法からなる.

多品種小変更型開発に改善手法を適用した結果，ソフトウェア部品の再利用率の改善，類似部品発生率の維持抑制が可能となった．この結果，高品質状態を維持したまま，事業上の目標である製品群全体における生産性を改善することができた．これらの結果から，改善手法の有効性を確認した.

第4章

可変性管理の課題と解決

4.1. 可変性管理の重要性

近年，家電製品をはじめとする組込み製品においては，他社との差別化のための高機能化，多様なユーザニーズに応えるための製品バリエーションの充実化，タイムリな市場投入が強い事業要求となっている[23].

このような事業要求に対応するため，ソフトウェアプロダクトラインエンジニアリング（以下，SPLE）[5] [23] [26]が広く実践されている．SPLE とは，ある製品群において，共通かつ管理された機能を共有する一式のソフトウェア・システム群のことであり，体系的な再利用を特徴とする開発手法のことである[5].

SPLE 成功のためのキープラクティスの一つに可変性管理がある．可変性管理とは，対象製品群の各製品間で変わりうる仕様を定義し，複数の製品開発に利用していく活動である[27]．可変性管理が重要とされる理由は，製品間の差異の分析や定義の良し悪しによって，コア資産（共有する設計書やソフトウェア部品）の再利用性が変わり，製品群全体の生産性に影響を及ぼすためである[5][42].

可変性管理の主要な技術に，可変性の記述法と構成手段がある[37][43]．可変性の記述法とは，製品間の差異の分析や定義を行うための表記法であり，可変性の構成手段とは，可変性を製品のソフトウェアとして実現する手段のことである．可変性管理の各技術については，様々な研究がなされる一方で，実践に基づく複数手法の比較検証や，特定ドメインに特化した実践が期待されている[44].

筆者らは，業務用空調機（室外機）を対象に 2010 年から 5 年間 SPLE を適用してきた（以降，第 1 期活動と呼ぶ）．第 1 期活動では，一時的に生産性に大きな改善があったが，時間経過とともにソフトウェア部品の再利用率が低下し，その結果製品群全体の生産性も低下した．この原因のひとつとして，2 章および文献[6]に示した通り，開発上流工程からの体系的な再利用を促す仕組みがなかったことがある．このような課題に対し，コア資産保守プロセスを，開発ロードマップに基づき，アプリケーション開発を一つの幹開発と複数の枝開発に分け，幹開発では当該製品のソフトウェア部品を開発しつつ，後続する枝開発で再利用可能なソフトウェア部品を開発するプロセスへと見直した[45]．しかしながらコア資産保守プロセスの改善だけでは，仕様差異を生み出す特徴（フィーチャ）の増加に対して構成選択が難しくなるという可変性管理における課題が残る．

本章は，第 1 期活動における可変性管理の問題点の分析結果に基づき，可変性管理の改

善手法と支援ツールを提案するとともに，第 1 期の活動とそれらを適用した第 2 期の活動を定量的に比較検証する．改善手法は，要求仕様書に製品-機能マトリクスとデータ仕様書を導入することで，可変性記述の一覧性と可読性を向上させるとともに，これら可変性記述から製品ソフトウェアへの実装・選択方法を仕様設計時点で明確にし，かつビルドへ直結する構成手段を提供することで，アプリケーション開発におけるコア資産の再利用率を向上させる．改善手法がコア資産の再利用性にどのような影響を与えるか考察し，製品ソフトウェアの構成決定を容易化することによってコア資産の再利用率が向上するという結果を得た．

4.2 節では，可変性管理に対する要求事項を整理し，4.3 節で適用対象の特徴を述べるとともに，可変性管理の改善手法を提案する．4.4 節では第 1 期活動からの改善ポイントを説明しつつ，開発における効果を定量的に比較検証する．

4.2. 可変性管理と技術

4.2.1. 可変性管理

可変性管理とは，対象製品群における可変性を定義し，複数の製品開発に利用してするための活動である．ここで，可変性とは，対象製品群の各製品間で変わりうる仕様のことである[29].

可変性管理の主要な技術に可変性の記述法と構成手段がある．可変性の記述法とは，製品間の差異の分析や定義を行うための表記法のことであり，可変性の構成手段とは，製品間の可変性をソフトウェアとして実現する手段のことである．

4.2.2. 可変性管理の記述法

可変性の記述法は，様々なものが考案されており，野田らが以下のように整理している[37][46].

(1)フィーチャモデル

フィーチャモデルは，製品群の持つフィーチャをツリー構造で階層的に表現する図式表現である[47]．直感的に理解しやすい利点を持つが，様々な情報が詰め込まれやすく現実の開発での適用が難しいとの指摘もある．

(2)直交可変性モデル

直行可変性モデルは，開発成果物と独立（直交）させて可変性自体をモデル化する．可変性と各開発成果物の関係を明示的に紐づける点に特徴がある[37]．複数の開発成果物に可変性情報が含まれる際に，関係性の理解や維持が容易となる．

(3)DM（Decision Modeling）

DM は，形式的なテキスト表現によるモデルで，可変点ごとに，決定すべき内容と選択肢を記載する[46]．フィーチャモデルや直交可変性モデルのようなグラフ表現に比べ一覧性に優れるが，直感性が損なわれる．

可変性記述は，アプリケーション開発者が製品の具体的な仕様を定義するために用い，その仕様が製品導出への入力となる．そのため，可変性記述法には，アプリケーション開発者が定義・レビューするための製品要求仕様書としての扱いやすさと，コア資産からソフトウェア部品を抽出しやすい情報形態であることが求められる．

4.2.3. 可変性の構成手段

可変性の構成手段とは，可変性の実現手段のことである．Gacek らは，開発言語に依存しない構成手段について，一般的によく使われているものを整理し以下のように分類している[44][48]．実際には，これらの構成手段を，実応用に合わせて組み合わせたり改良を加えたりして利用されている．

(1)条件コンパイル

条件コンパイルは，#ifdef（コンパイルスイッチ）によって，コンパイルする箇所を特定する方法である．条件コンパイルは，既存コードに影響なく追加できるため導入障壁が低い一方で，言語とマクロの 2 つの言語が混在し，頻繁なコード中断やレイアウト崩れが起き，コードの可読性が悪化しやすい[49]．

(2)フレーム方式

フレーム方式は，直接ソースコードを再利用するのではなく，パターンに基づいて再利用モジュールを生成しフレームワーク部分と結合する方法である．パターンの定義および適用に成功すれば再利用率が向上するが，適用対象であるドメインを十分に分析する必要があるため，導入障壁が高く，適用範囲が狭くなりがちである[50]．

(3)パラメータ方式

パラメータ方式は，セットされたパラメータによって振る舞いを変えるコンポーネントを用いる方法である．再利用性や仕様とのトレーサビリティに優れるが，パラメータに応じた処理をすべてコンポーネントに内包するためコンポーネント自体が複雑になりがちである．

(4)静的ライブラリ

静的ライブラリは，オブジェクトファイルからなるライブラリをコンパイル・リンク時に組み込む方法である．実装時に任意の静的ライブラリを選択することで，必要な部品のみから製品を構成できる．ただし，出荷後の部品の切り替えはできない．

(5)動的リンクライブラリ

動的リンクライブラリは，ライブラリをプログラム実行中に読み込む方法である．メモリ使用量や ROM サイズの節約などの利点があるが，実行時の組み合わせで振る舞いが変わるため品質保証が難しい．

可変性の構成手段は，プロダクトラインアーキテクチャおよびソフトウェア部品の形態を決定づける．また，構成手段の選択は，製品導出時のソフトウェア部品の組立て，製品ビルドの効率，ソフトウェア部品の保守のしやすさに影響を与える．従って，構成手段の選択では，適用するドメインの可変性や開発環境への適合のしやすさが求められる[44].

4.3. 可変性管理の提案

4.3.1. SPLE 適用対象

我々は，業務用空調機（室外機）の製品群に対して，SPLE の本格適用を 2010 年より開始した．この製品群は，日本国内のみならず世界各国へ出荷され，また能力帯や機能の搭載有無，ハードウェア構成の違いがあるため，様々な製品バリエーションを持っている．比較的小さな仕様の差異をいかにソフトウェアでカバーするかが開発戦略上重要であり，多品種小変更型開発の特徴を有している．多品種小変更型開発の特徴[31]を以下に記す

①製品の機能・性能がハードウェアに大きく依存するため，製品群の仕様及び開発計画は，機械（メカ）や電気（エレキ）部門が決定する．ソフトウェア部門は仕様に従いソフトウェアを開発する．メカ・エレキ部門（製品部門）は部門ごとに製品用途や出荷地域が異なる固有の市場に対応している．

②製品群は，ソフトウェアからみて，以前の開発サイクルの製品群の仕様の多くを共有し，また同一開発サイクルで開発する新機能・性能改善のための仕様も共有している．

③製品群は，②の共通仕様をベースにして，特定用途や特定の出荷地域向けにカスタマイズする必要がある．

④①および③の特徴から，製品部門からの仕様は，コア資産に基づくものでなく，以前開発した製品に基づいたものとなりがちである（例：前年度製品をベースに，ある機能を変更する等）

⑤個々のカスタマイズ開発は小規模であり，短期に並行して実施される．

4.3.2. 第 1 期活動における可変性管理の課題

第 1 期活動における可変性管理プロセスについて，各コア資産の可変性記述と構成手段およびソフトウェア部品のアプリケーション開発での使われ方を図 20 に示す.

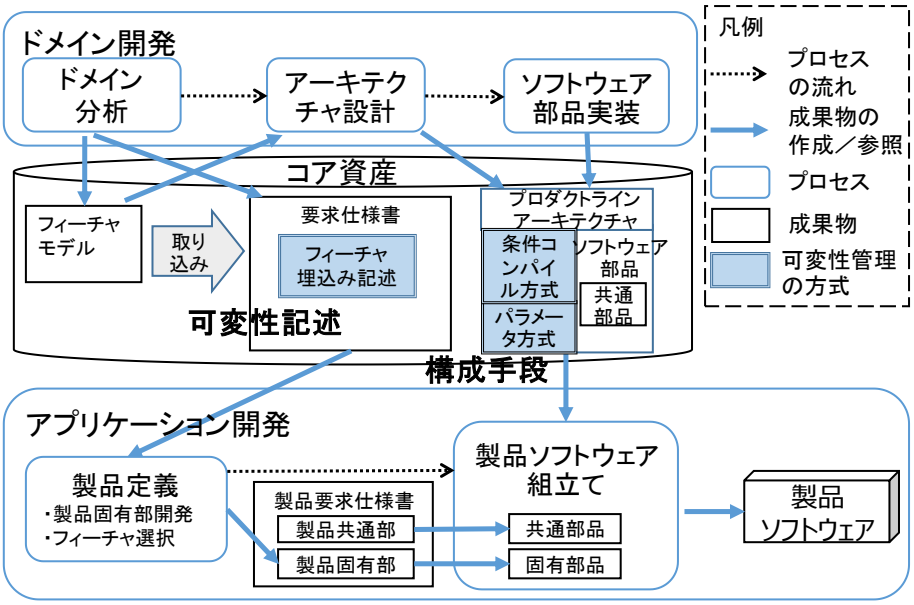

図 20 第 1 期活動における可変性管理手法

ドメイン開発では，コア資産である要求仕様書とプロダクトラインアーキテクチャおよび共通的なソフトウェア部品を開発する．要求仕様書は，フィーチャモデルに基づいて構成しており，製品機能内の仕様差異をフィーチャに基づく可変性として記載する（フィーチャ埋込み記述）．例えば，圧力保護機能の検出方式の差異である「圧力スイッチによる検出」と「温度推定による検出」の 2 つを，フィーチャに対応する検出方式の仕様（バリアント）として記載している．プロダクトラインアーキテクチャは，条件コンパイル方式を採用し，先述した製品機能内の仕様差異を，ソフトウェア部品内のコンパイルスイッチで実現している．

アプリケーション開発では，当該製品での固有制御を製品固有部として新しいフィーチャとともに製品要求仕様書に埋め込み，ソフトウェアの固有部品として実装する．この固有部品を登録テーブルに登録するとともに，条件コンパイルで有効にして固有の製品ソフトウェアを生成する．また，パラメータ差異はフィーチャの組み合わせによって具体的な値を決定する方式とした（パラメータ方式）.

製品定義（アプリケーション開発）では，コア資産に関する知識を持たない製品部門からの仕様をもとにソフトウェア部門がソフトウェア部品を再利用／開発する（特徴①）ことになるが，複数の開発が並行する（特徴⑤）ため，3 章および文献[45]に示した通り，仕様の解釈時に，可変点の見逃しが頻繁に起き，結果的に類似したソフトウェア部品（固有部品）

を開発してしまう．これは可変性管理における以下の 2 点の課題が原因となっている．
[課題①] 要求仕様書の目次構造により，内部に記載された開発済みの制御方式を共有できず，利用可能なソフトウェア部品を使わずに新たな仕様として要求されることが多い．
[課題②] 製品系列の増加に伴うフィーチャ数の増加によって，製品部門のフィーチャ選択に基づいて具体的なソフトウェア部品を#ifdef の組合せで決定するのが困難になった．

4.3.3. 可変性管理における提案手法

4.3.3.1. 第 2 期の可変性管理プロセスと提案手法

第 2 期活動における可変性管理プロセスについて，各コア資産の可変性記述と構成手段およびソフトウェア部品のアプリケーション開発での使われ方を図 21 に示す．

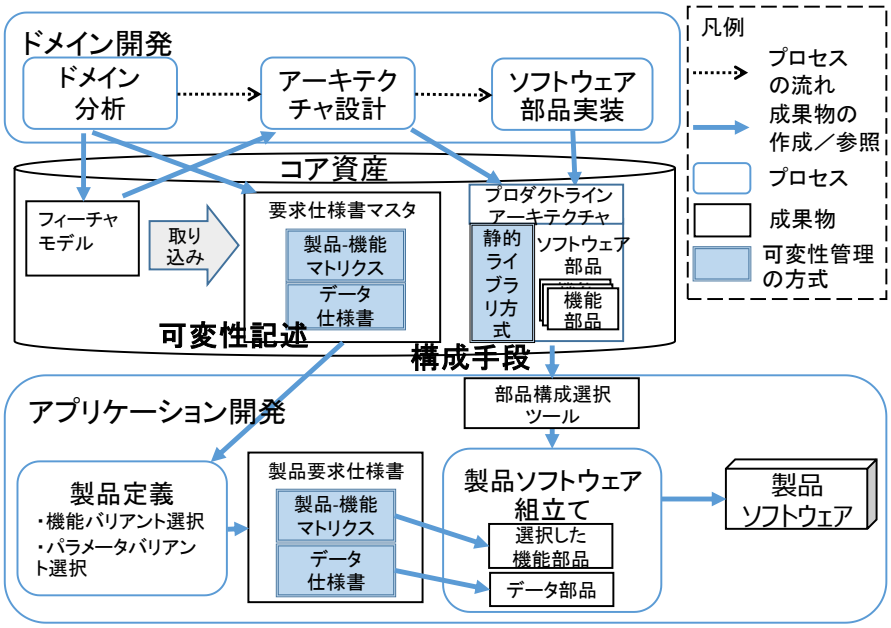

図 21 第 2 期活動における可変性管理

第 1 期活動からの主な変更点は，以下である．
①要求仕様書の可変性記述法をフィーチャ埋込み記述から製品 - 機能マトリクスおよびデータ仕様書へ変更
②プロダクトラインアーキテクチャの構成手段を条件コンパイル方式から静的ライブラリ方式に変更
③構成手段を支援する部品選択ツールを整備

本変更によって，製品部門と開発済みの製品機能を共有できること（課題①への対応）と，ソフトウェア部品の選択を容易にすること（課題②への対応）を狙っている．

4.3.3.2. 可変性記述法の提案

提案手法における可変性記述法（変更点①）を説明する．要求仕様書の構成を図 22 に示す．

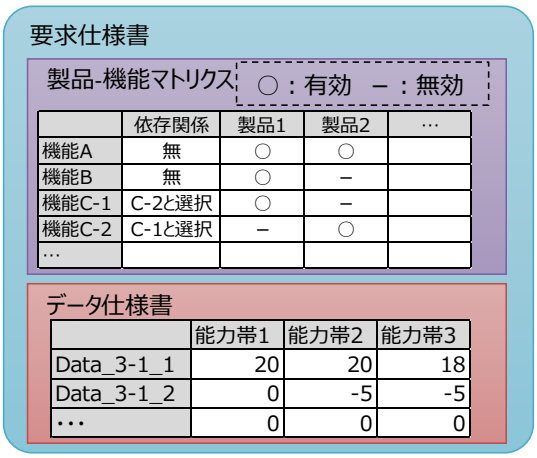

図 22 要求仕様書の構成

要求仕様書（要求仕様書マスタ）は製品 - 機能マトリクスとデータ仕様書からなる．製品ー機能マトリクスは，縦軸に製品群に搭載されている機能，横軸に製品として，交点に各製品での機能の有効／無効を明示した表である．縦軸の機能一覧においては，全機能を列挙するとともに，制御方式に差異がある類似機能を依存関係とともに明示している．図 22 においては，製品 1，製品 2 を含む製品群全体で機能 A，機能 B，機能 C-1，機能 C-2 を有する可能性があることを意味し，機能 C-1 と機能 C-2 はあるフィーチャによって機能が択一的に選択される機能であることを示している．製品 1 は機能 A，機能 B，機能 C-1 を有する製品であることを定義している．機能 C-1 および機能 C-2 とは，具体的には，圧力スイッチによる圧力保護と温度センサによる圧力推定を行う圧力保護をそれぞれ別の製品機能として定義し，製品機能の目次レベルで可変性があることを明示するとともに，択一的な関係であることを示す．

データ仕様書とは，製品個々のパラメータを一括して記載する文書である．チューニング可能なパラメータに対し，[Data_3-1_1]のようなタグを本文中に埋め込んでおき，タグの付いたパラメータの設定値をデータ仕様書に記載する．例えば，Data_3-1_1 は温度閾値を意味し，能力帯 1 と能力帯 2 では 20 度だが，能力帯 3 では 18 度であることを表す．この見直しは，パラメータ差異がフィーチャの組み合わせだけで決定せず，ハードウェア構成や出荷地域等の複合的な要因により決定するという分析結果に基づいている．

要求仕様書マスタを用いた開発例を図 21 に基づいて示す．ドメイン開発において作成した要求仕様書マスタには，開発済みの製品機能と製品機能のパラメータが記載できるデー

タ仕様書が含まれている．アプリケーション開発では，要求仕様書マスタをカスタマイズすることで，個別の製品要求仕様書を作成する．具体的には，各アプリケーション開発（製品定義）において，搭載する製品機能を選択することで製品－機能マトリクスの担当製品に対応する縦の欄を完成させ，当該製品用のデータ仕様書に能力帯やハードウェア構成の違いによるパラメータ設定を記載する．

表 8 に第 1 期と第 2 期の可変性記述法の違いを整理する．

表 8 可変性種別と要求仕様書の対応関係

可変性種別	第 1 期	第 2 期
機能有無	目次に登場する機能	製品 - 機能マトリクス
制御方式	機能内の可変点	
パラメータ		データ仕様書

第 1 期活動においては，制御方式やパラメータの差異は製品機能内の可変点として記述していたが，第 2 期活動では，制御方式が異なる製品機能を異なる製品機能として定義したうえで，すべての製品機能を製品 - 機能マトリクス上に列挙し，機能の搭載有無を表の交点に記載している．また，データの差異については個々に可変点として記述する方式から，データ仕様書に一括で記載する方式に切り替えている．すわなち，具体的なバリアント（制御仕様）を決めるために複数のフィーチャを段階的に選択する方式から，制御仕様を直接的に選択できる方式に切り替えた．

これらの方式の切り替えによって以下の 2 点が改善した．

①要求仕様書における可変性の一覧性向上

第 1 期では，500 ページ程度の要求仕様書の中から可変点を探し出し，当該製品に適合するバリアント（制御仕様）を選択する必要があったが，第 2 期では，5 ページ程度の製品 - 機能マトリクスの中から対象の製品機能を見つけ出し制御仕様を確認の上，選択することができるようになった．このように可変点およびバリアント（制御仕様）の検索が容易となったため，開発済みの製品機能の再利用率が向上した．

②要求仕様書における可変性の可読性向上

第 1 期では，目次に登場する機能の搭載有無を決定したのちに，機能内の可変点ごとに，フィーチャを組み合わせてバリアント（制御仕様やパラメータ）を決定していたが，第 2 期では，製品 - 機能マトリクスによって製品機能を取捨選択し，各機能のパラメータを当該製品用のデータ部品によって決定するという構成としたため，具体的なバリアントを決定するのが容易になった．製品 - 機能マトリクスは，製品機能を取捨選択する（搭載有無を決定する）というシンプルな表記法となっているため，熟達技術者でなくともその意味を理解することが容易であり，プロダクトラインアーキテクチャを理解していない製品部門でもコ

ア資産の確認が容易となる.

4.3.3.3. 可変性の構成手段の提案

可変性記述の見直しに基づき，仕様設計時点で明確にしたソフトウェア部品としてのモジュール性をそのままに設計・実装できるようにするため，表 9 に示すように，構成手段を静的ライブラリ方式に変更し，パラメータについてはフレーム方式を併用することとした.

表 9 可変性種別と構成手段の対応関係

可変性種別	第 1 期	第 2 期
機能有無	条件コンパイル方式	静的ライブラリ方式(パラメータはフレーム方式と併用)
制御方式差異		
パラメータ	パラメータ方式	

機能有無や制御方式差異に対する構成手段として，第 1 期活動では，実装のしやすさから条件コンパイル方式を採用していた．これに対して，第 2 期活動では，すべての機能が製品－機能マトリクスによって，機能の有無で表現できるようにするため，静的ライブラリ方式へと見直した.

パラメータ差異に対する構成手段ついても同様に，構成手段をすべて静的ライブラリ方式に統一した．これは，可変性記述法の見直しによって，パラメータ差異を，製品機能内のバリアントから製品ごとのバリアント（データ部品）として扱うことにしたためである．さらに，データ部品については，パラメータやソースコード自体を再利用対象とするのではなく，データ仕様書からソースコードへのパターン化した変換ロジックを再利用対象としている（フレーム方式）．具体的には，データ仕様書の各パラメータは，能力帯によらず一定の値をとる定数型,および能力帯ごとに設定値が異なる能力帯テーブル型の 2 種で記載し，パターンに応じて自動的にソースコードと取得関数に展開している.

これらの構成手段の変更によって，仕様からソフトウェア部品の実装方法が単純化しソフトウェア部品開発時の作業者の担当分けのしやすさや実装ミスの減少が効果として得られ，また製品との対応付けが明確になり，ビルド管理を含む構成管理も容易となった.

4.3.4. 可変性管理におけるツール支援

構成手段の見直しに合わせ，構成選択を効率化する目的で支援ツール（以降，部品構成選択ツール）を開発した．構成を図 23 に示す.

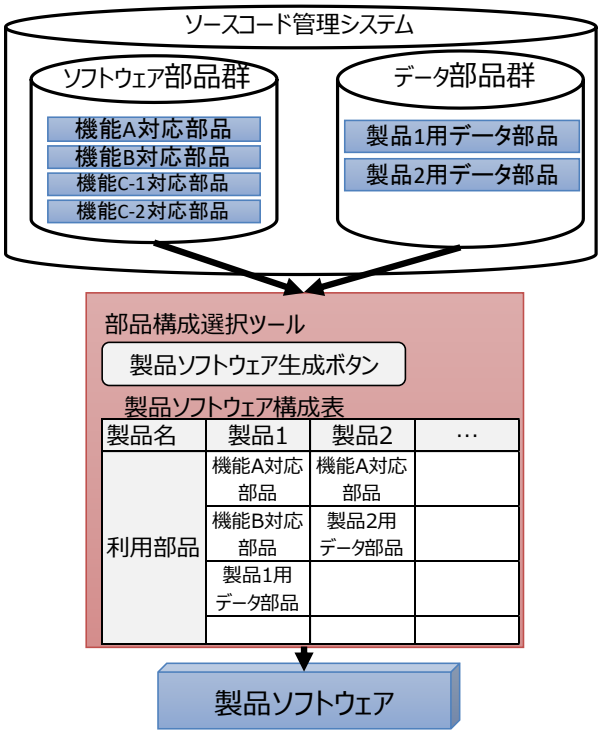

図 23 部品構成選択ツールの構成

部品構成選択ツールは，製品ソフトウェア構成表と製品ソフトウェア生成ボタンから構成されるツールで，ソースコード管理システム上にあるソフトウェア部品群とデータ部品群を入力として，製品ソフトウェアを出力する．製品ソフトウェア構成表は横軸に製品，縦軸に当該製品で利用するソフトウェア部品を列挙した表で，要求仕様書の製品－機能マトリクスに基づいて生成する．

製品ソフトウェア構成表の中からビルドしたい製品を選択して，製品ソフトウェア生成ボタンを押下することで，各ソフトウェア部品をソースコード管理システムから取得し Makefile を生成する．このとき，データ部品は，データ仕様書をもとにデータ形態に応じて取得関数を含むソースファイルに変換され，Makefile に組み込まれる．このように生成された Makefile を実行することで製品ソフトウェアを生成できる．

このような支援ツールを開発することによって，仕様設計時点で明確になったソフトウェア部品としてのモジュール性を活かして効率的に製品ソフトウェアを構成することが可能となる．

4.4. 提案手法の適用評価

4.4.1. 第 1 期からの改善ポイント

要求仕様書における可変性記述法においては，フィーチャ埋込記述から製品 - 機能マト

リクスおよびデータ仕様書に見直すことで，製品部門が開発済みの製品機能を把握・選択することが容易となる，つまり仕様が共通化されることを狙っている．

また，可変性の構成手段においては，条件コンパイル方式から静的ライブラリ方式に見直すことで，製品ソフトウェア組立て時に，各ソフトウェア部品のリンク有無のみを決定すればよくなる，つまり，製品構成決定のパラメータ数が低減できることを狙っている．

4.4.2. 開発における評価

4.3 節で述べた改善手法を，2015 年度開発製品より適用した．適用開始直前の 2014 年度の各指数を 100 として，各指標の推移を示し，適用の効果を示す．各年度のプロットは，各年度の開発で得られたデータの平均値とした．

4.4.2.1. 仕様共通化率 Rc

対応製品にあった仕様を見落としなく選択できてきることを評価するために，仕様共通化率を用いる．仕様共通化率 Rc は次式として定義する．

$$Rc = (A / B) \times 100 \qquad (1)$$

A:複数製品で使用される機能数

B:製品群全体での全機能数

なお Rc は比較のため，2014 年を 1.00 とした相対値 Rc-yy で示す．Rc-yy は次式と定義する．

$$Rc\text{-}yy = yy\text{ 年の }Rc \ / \ 2014\text{ 年の }Rc \qquad (2)$$

仕様共通化率は複数製品で利用された仕様の割合を示す指標となっている．本手法においては，4.3.3.2 目の要求仕様書において選択する仕様に対応して，粒度の揃った製品機能を部品化していること，各部品内にはパラメータ差異以外の可変点を埋め込まない方式を採用している．そのため，仕様共通化率を用いることにより，コア資産として作成した製品仕様がどれだけ予定通り再利用されているかを評価できると考える．

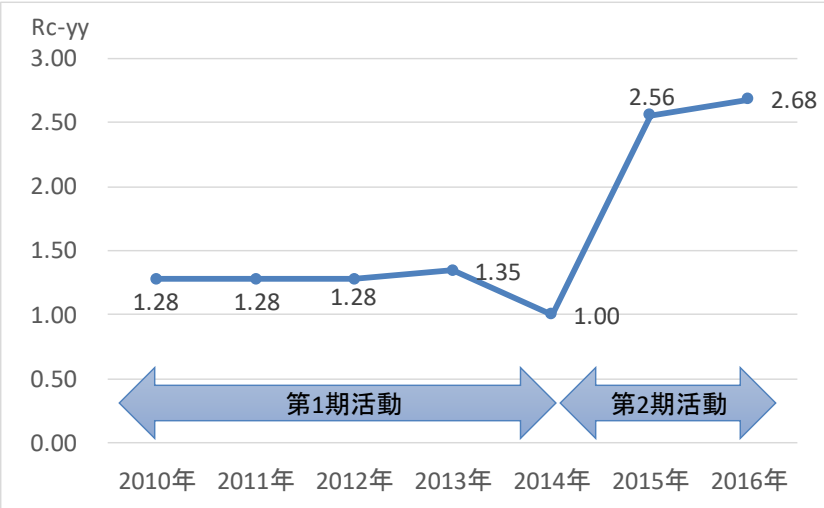

図 24 仕様共通化率

図 24 に示すように, 第 1 期活動では仕様共通化率は 1.00～1.35 倍と変動がなかったが, 第 2 期活動においては，15 年に 2.56 倍，16 年には 2.68 倍と，約 2.5 倍の仕様共通化率を達成している．

これは，第 1 期活動では，仕様書の目次構造の問題から製品部門と開発済みの制御方式が共有できておらず，類似した新たな仕様を開発してしまっていたが，第 2 期活動では，製品 - 機能マトリクスで一覧化できることによって 500 ページもの要求仕様書の中から可変点を探すのではなく，5 ページ程度のマトリクスにすべての可変点が表現されたことで，製品部門との共有が促進できた成果と考える．

4.4.2.2. 製品の構成決定パラメータ数 Pp

製品機能の具体的な仕様を効率的に決定できていることを評価するために，構成決定パラメータ数を用いる．パラメータ数 Pp は次式と定義する．

Pp = average(Nf / 製品機能) (3)

Nf:フィーチャ数（選択肢の数）

なお Pp は比較のため，2014 年を 1.00 とした相対値 Pp-yy で示す．Pp-yy は次式と定義する．

Pp-yy= yy 年の Pp / 2014 年の Pp (4)

構成決定パラメータ数は，一つの製品機能を実現するためにいくつの可変点を決定しなければならないかを示す指標となっている．本改善手法においては，ソフトウェア部品内にはパラメータ差異以外の可変点を埋め込まない方式としているため，目論見通り機能有無の選択とパラメータチューニングのみで製品機能を決定できているかを評価できる．

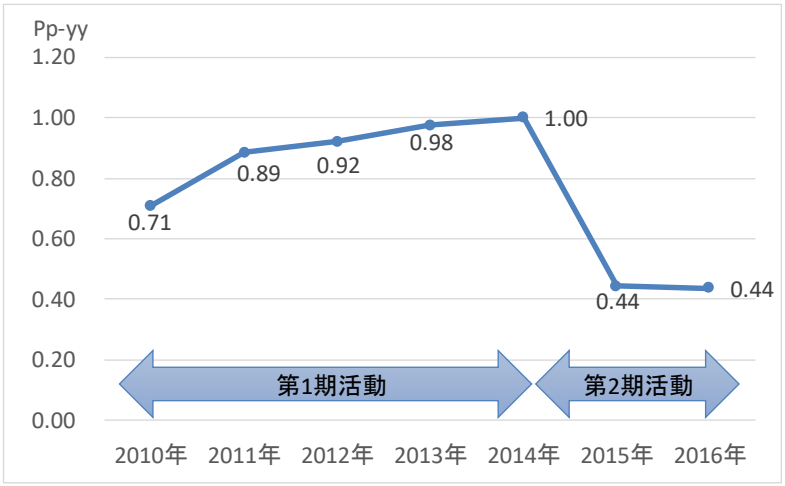

図 25 製品構成決定パラメータ数

図 25 に示すように，第 1 期活動では，Pp が年々増加しているのに対し，第 2 期活動では，Pp が 56%低減し，その後の増加はない．なお，第 2 期活動で Pp が減っているのは，本手法の導入に合わせて既存のコンパイルスイッチを削減するリファクタリングを実施したためである．

第 1 期活動では，製品系列の増加に伴いフィーチャ数が増加し，コンパイルスイッチが部品内に散在する結果となったが，第 2 期活動においては，部品のリンクおよびデータ部品の決定のみでよくなったったため，フィーチャ数が増加しても，構成決定が容易な状態を維持できた．

本手法はコア資産保守プロセスの見直しと合わせて適用することで，幹開発で開発したソフトウェア部品の枝開発における認知度が向上し，さらに，枝開発における構成決定が容易化できたことで，3 章および文献[45]にも示したようにソフトウェア部品の再利用率向上を実現している．

4.4.3. 妥当性に対する脅威

本改善手法における製品 - 機能マトリクスは，製品と具体的な仕様（バリアント）を直接結び付ける表であり，バリアント選択の根拠となるフィーチャが登場しない．そのため，一つの可変点に多くのバリアントが存在するような製品群の場合，なぜその仕様を選択しているのか，その根拠がわかりにくくなり，後述するディシジョンモデルと比較して，製品部門が正しくバリアント（制御方式）を選択できないという問題も発生しうるが，現時点ではバリアントの選択肢が多岐にわたる場合が少なくその選択理由が比較的自明であるため，特に大きな問題を生じていない．

データ仕様書およびデータ部品については，多くの製品群ではパラメータの違いで製品

間の差異を実現しているものも多いと考えられるので，個々のパラメータの違いを可変性とするのではなく，データ部品としてまとめて可変性としてしまう手法は他のドメインでも有効と考える．

4.5. 関連研究

4.5.1. 可変性記述法

可変性記述法に関する研究は数多く行われている．Czarnecki らは，レベルの異なるフィーチャが混在することによるフィーチャモデルの崩壊を回避するため，段階的構成を提唱している[51]．段階的構成では，フィーチャモデルを段階的に特殊化し，製品の構成を決定していくやり方を採る．Classen らも MLSC（Multi-Level Staged Configration）と呼ばれる複数レベルの段階的構成によって製品構成を実現している[52]．ソフトウェアを解さない製品部門と製品構成を共有するためには単一の表記法によってシンプルに製品構成を示すことが重要であるが，Czarnecki や Classen らの取組みでは，直感的に製品構成を理解させることが難しい，我々の手法は製品 - 機能マトリクスという単一の表記法によってどの製品にどの仕様（バリアント）を採択するかを示すことが可能である．

一つのフィーチャモデルと表によって製品導出を実現している事例もある．Czarnecki らは，フィーチャモデルとディシジョンモデルによって共通性と可変性の抽出および製品導出支援を実現している[46]．ディシジョンモデルは，縦軸にフィーチャ，横軸にフィーチャを決定するための質問やフィーチャの型や範囲が記述されており，フィーチャ決定を支援している．多品種小変更型開発の製品導出においては，ソフトウェアを解さない製品部門が直接的に製品構成を選択できることが重要であるが，ディシジョンモデルを用いた取り組みは，それらを直接的に行うことはできない．一方で，本改善手法では表を機能と製品の星取表とし製品導出に利用しているため直接的にバリアント選択が可能である．

4.5.2. 可変性の構成手段

Faulk らは最上位のクラス図で製品群の共通性を表し，サブクラスにより可変性を表現する手法を提案している[57]．Faulk らの提案は，サブクラスを切り替えることで製品固有のニーズを実現している点に特徴がある．Faulk らの取組みは，可変性を把握するためにはサブクラスまで確認しなければならないが，本研究は，類似機能を別機能として扱うため最上位のレベルで可変性を把握することが可能である．ソフトウェアを解さない製品部門にとっては，可変性が把握しやすいレベルで一覧化されていることが重要であり，本改善手法は多品種小変更型開発に適した手法と考える．

Greenfield らは，モデル駆動開発によって製品ソフトウェアの生成を実現している[53]．本取り組みは，OOA&D（object oriented analysis and design）をベースとしたモデル駆動

開発で，モデル内部で可変性を管理し，コードを自動生成している点に特徴がある．パラメータは，フィーチャの組み合わせだけでなく，ハードウェア構成や出荷地域等の複合的な要因により決定する可変性であり，モデル内部で可変性を管理することが難しい．パラメータ差異のような小さな可変性を製品ごとに一括管理し，変異体を増加させない点に本研究との差異がある．

4.6. 可変性管理のまとめ

本章では，空調機開発を対象に，コア資産保守プロセスの見直しと合わせて導入した可変性管理の改善手法を評価した．改善手法は，要求仕様書に製品-機能マトリクスとデータ仕様書を導入することで，可変性記述の一覧性と可読性を向上させ，仕様設計時点で可変点およびバリアントの選択方法を明確にするものである．さらにこれら可変性記述から製品ソフトウェアのビルドへ直結する構成手段を提供することで，アプリケーション開発におけるコア資産の再利用率を向上させる．

本改善手法によって，ソフトウェアへの理解の浅い製品部門とコア資産を共有することに成功し，仕様共通率を約 2.5 倍に高めている．さらに構成決定パラメータ数を 56%低減し，構成決定を容易化している．この結果，3 章および文献[45]にも示した通りコア資産の再利用率向上を実現している．これらの評価結果に基づき，可変性管理はステークホルダとのコア資産共有と構成決定を容易化することが重要であるという知見を得た．

第5章

ブランチ・マージプロセスにおける改善と考察

5.1. SPLE におけるブランチ・マージの重要性

近年，家電製品等の組込み製品においては，他社差別化のための高機能化，多様なユーザニーズに応えるための製品バリエーションの充実化，タイムリな市場投入が強い事業要求となっている．

このような事業要求に対応するため，ソフトウェアプロダクトラインエンジニアリング（以下，SPLE）[5][23][26]が広く実践されている．SPLE とは，ある製品群において，共通かつ管理された機能を共有するソフトウェア・システム群を活用して体系的な再利用をする開発手法のことである[5][54].

SPLE において，コア資産（再利用対象となる設計書やソースコード等のソフトウェア資産）の維持と発展のためには，構成管理を確実に行うことが重要である[55]．なぜならば，SPLE においては，コア資産への変更の影響は個々の製品の品質や生産性に多大な影響を及ぼすためである[56]．ここでコア資産の構成管理とは，コア資産自体のバージョン管理やコア資産の各製品への適用状況を管理することを指す[27][55].

複数プロジェクトによる並行開発でのコア資産の維持・発展を確実に行うための構成管理の技術領域の一つとしてソフトウェアのブランチ・マージがある．ソフトウェアのブランチ・マージとは，複数の並行した開発において互いの影響を受けずに開発するためにコア資産を派生させること（この作業を「ブランチを切る」と呼ぶ），および開発されたソフトウェア部品（再利用対象であるソフトウェアのモジュール）の変更差分を，コア資産として統合すること（この作業を「マージする」と呼ぶ）を指す[57]．ブランチ・マージの各作業をまとめた一連の活動をブランチ・マージプロセスと呼ぶ.

ブランチ・マージにおける主要な問題として，マージの衝突がある．マージの衝突とは，並行する開発で同じソフトウェア部品に，異なる開発者が異なる変更を別々に実施し，その変更をコア資産に戻す際に，2 つの変更が衝突することを指す．マージの衝突は，多大な設計のやり直しや修正誤りを引き起こす[57].

我々は，業務用空調機（室外機）を対象に 2010 年から SPLE を実践している．業務用空調機は，製品の並行開発が多く，ソフトウェア開発も並行しやすいためマージの衝突が生じやすいという特徴がある.

適用当初は，一つのトランク（コア資産を保管する場所）でコア資産を保守・開発し，個々のリリース時にコア資産のブランチを切り，製品固有の仕様変更や不具合修正があった場

合，開発後にマージする方法を採用していた．この方法では，同時並行の開発プロジェクト間で独立してコア資産に変更を加えるため，複数の開発が並行する中でマージの衝突が起こりやすくなっていた．このため，製品開発部門において，自部門の製品開発を最適化しようとしてコア資産の固有化（製品特化）が進んでいた．また，製品部門では他製品からの変更がマージされたコア資産の品質を懸念して利用を控える事態も生じていた．

この分析結果に基づき，マージの衝突回避ならびにコア資産の信頼性向上および再利用率の向上を狙い，①アプリケーション開発開始時に開発予定のソフトウェア部品のブランチ状況を確認しマージ計画を検討すること（部品開発計画策定），②開発終了後に将来の製品適用を見据え変更したソフトウェア部品を使用する他製品系列の動作保証を行うこと（マージ試験），をプロセスに組み込む改善を行った．

本章は，SPLE 適用経験から，複数組織による並行開発でのコア資産のブランチ・マージにおける問題点を明らかにし，その問題点を解決する改善手法を提案する．本改善手法を実開発に適用し，マージの衝突およびコア資産ブランチ数にどのような効果があるかを改善前後で比較検証することにより有効性の評価を行なった．その結果，並行開発時の部品開発状況を可視化し信頼性を確保することがコア資産の派生の抑制やマージにかかるコストの抑制に効果があることが分かった．

5.2 節では，適用対象の特徴および複数組織による並行開発でのコア資産のブランチ・マージの問題点を述べるとともに，ブランチ・マージプロセスの改善手法を提案する．5.3 節で，改善前後のプロセスを定量的に比較検証することで改善手法の有効性を評価し，5.4 節にて本評価に対する妥当性への脅威を示す．5.5 節では，本章で研究対象としたソフトウェアマージの衝突回避および SPLE の品質保証に関する関連研究を紹介し，5.6 節で，今回の実践によって得られた知見をまとめる．

5.2. ブランチ・マージの改善手法の提案

5.2.1. SPLE 適用対象

我々は，業務用空調機（室外機）の製品群に対して，SPLE の本格適用を 2010 年より実施してきている．この製品群は，日本国内のみならず世界各国へ出荷され，冷暖房能力や機能の搭載有無，ハードウェア構成の違いがあるため，様々な製品バリエーションを持っている．ハードウェア起因の小さな仕様の差異をいかにソフトウェアでカバーするかが開発戦略上重要であり，多品種小変更型開発として，以下の特徴を有する．

(ア)　製品の機能・性能がハードウェアに大きく依存するため，製品群の仕様及び開発計画は，機械や電気部門（まとめて製品部門と呼ぶ）が決定する[31]．製品部門は用途や出荷地域が異なる固有の市場ごとに組織（部や課）が分かれており，互いの仕様変更を把握していない．

(イ)　製品部門からの仕様は，コア資産に基づくものでなく，以前開発した製品に基づい

たものとなる．例：前年製品をベースに，ある機能を変更する等．
(ウ)　　個々の仕様変更は小規模であり，短期に並行して実施される．

適用対象の製品群において，製品部門は市場ごとに組織が分かれているのに対し，ソフトウェア部門は，一つの部門が各製品向けにソフトウェアを開発し提供している．

5.2.2. 改善前のプロセスとその問題点

5.2.1 項で述べた特徴を有する業務用空調機において，2010 年から 5 年間 SPLE 適用を継続した．この際，コア資産のブランチ・マージプロセスは，一つのトランクでコア資産であるソフトウェア部品を保守・開発し，個々のリリース時にコア資産のブランチを切って変更を加え，開発後にトランクにマージする方法を採用していた．このプロセスを 5 年間継続したことにより，2 章および文献[6]に示した通り，コア資産のブランチ数が増加するという問題が生じた．

改善前のプロセスにおけるブランチ・マージの進み方を図 26 に示す．

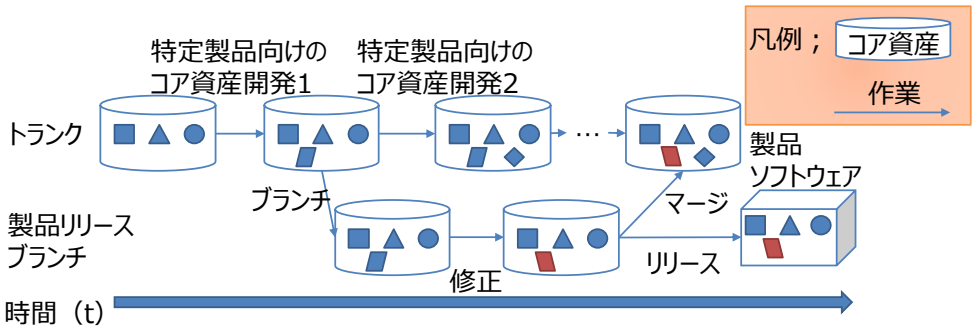

図 26 改善前のプロセスにおけるブランチ・マージの進み方

このプロセスの詳細手順は以下となる．

① コア資産はトランクに保管

② 特定製品向けに部品をトランクで開発

③ 部品開発完了後，トランクからブランチを切って製品リリースブランチを作成する．製品リリースブランチは，ある製品向けの評価やチューニングを目的に，トランクで開発したソフトウェア部品を固定化するためのものである．

④ 製品リリースブランチを使い，製品ソフトウェアを生成・リリース（リリース作業）．

⑤ 製品ソフトウェアに不具合や仕様変更が生じた場合は，製品リリースブランチ上で部品を修正する（チューニング目的の仕様変更は頻発する）

⑥ 製品ソフトウェアを出荷したのちに，確定した変更差分をトランクにマージする

改善前のプロセスは，ソフトウェア部品をトランクで開発する．これは，マージの衝突がほとんど生じないことを前提としている．この当時，並行開発を実践するソフトウェア開発者は 5 人程度であり，緊密なコミュニケーションやスケジュール調整によって同一トランクでソフトウェア部品を保守・開発することが可能であると考えていた．

しかし実際には，製品の並行開発の多さゆえに，同一のソフトウェア部品に同時に異なる仕様変更が発生することが多かった．この場合，並行に行われた他製品向けの部品開発との衝突を避けるため，ある製品のために切ったブランチ（製品リリースブランチ）がトランクにマージされず，継続的に製品固有のコア資産として維持される問題が生じた（図 27）．この派生したコア資産を製品メインブランチと呼ぶ．製品メインブランチをベースに開発が進んでしまうことで，製品群全体の再利用性が低下する，変更が累積しトランクに戻すためのマージコストが生じるという問題がある．

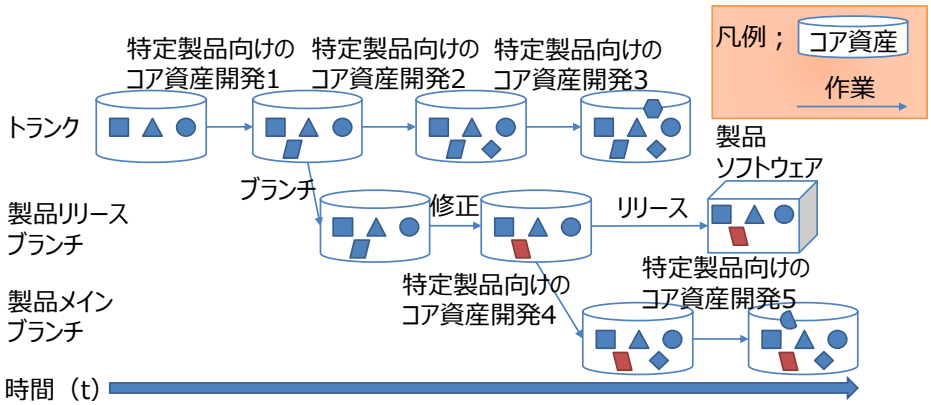

図 27 改善前のプロセスにおける製品メインブランチ

製品メインブランチが発生する原因は，以下である．

問題①：製品部門によるコア資産の固有化（製品特化）

多品種小変更型開発においては，同一ソフトウェア部品に対する開発の衝突が生じやすい．さらに，特徴（ア）および特徴（イ）から，製品部門は，自部門の製品開発を最適化するため，コア資産を製品部門固有にする，つまり製品メインブランチを積極的に作ることで他の製品開発の影響を最小化しようとする．製品メインブランチ上で開発を継続することで，変更が積み上がっていき，益々マージが困難になる．

問題②：未評価部品の利用回避

製品部門は用途や市場ごとに異なる開発組織が担当する．ある製品開発は，ベースとなる以前開発した製品に基づき実施される．その際，以下が問題となる．

・ベース製品は，その製品系列における評価済のコア資産，すなわち製品リリースブランチを選択するのがベストである．対してトランクには，その製品系列における前回開発での後，他の製品系列の部品開発による変更もマージされており，他の製品系列の開発で変更された部品は，当該製品系列では未評価である．

この問題のため，製品部門は，コア資産を自部門で開発している製品固有のものとしたがる（製品メインブランチを積極的に作りたがる）．

問題①および②を解決するためには，他製品からの部品変更の影響を受けずに開発しつつ，コア資産をマージする仕組み，およびマージしたコア資産が他製品系列に組み込んでも動作することを保証する仕組みを構築することが課題である．

5.2.3. 改善プロセスの提案

5.2.3.1. 改善後のプロセスの概要

5.2.2 項で述べた課題を解決するためにブランチ・マージプロセスの改善を行った．改善後のプロセスにおけるブランチ・マージの進み方を図 28 に示す．

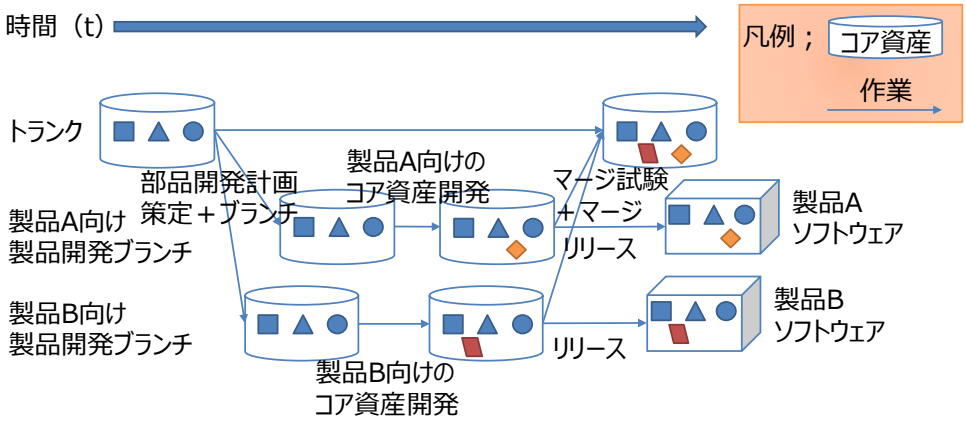

図 28 改善後のプロセスにおけるブランチ・マージの進み方

改善後のブランチ・マージプロセスは，製品開発ごとに製品開発ブランチと呼ぶブランチを切って保守・開発し，製品開発完了後に開発したソフトウェア部品をコア資産にマージする方式を採用している（問題①への対応）．さらに，ソフトウェア部品をトランクからブランチする際に「部品開発計画策定」を実施する（問題①への対応）点と，開発したソフトウェア部品をマージする際に，「マージ試験」を実施する（問題②への対応）点を改善前のプロセスに追加している．このプロセス変更によって，マージの衝突を回避することと，開発したソフトウェア部品が他製品でも利用できることを狙っている．以下に改善プロセスを詳述する．

5.2.3.2. 製品開発ブランチでの部品開発

改善前のプロセスでは，すべてのソフトウェア部品を一つのトランク上で保守・開発するプロセスを採用していたため，複数の製品の開発が重なると製品メインブランチが作成されやすく変更が累積しやすい（マージがされにくい）という問題があった．この問題を解決するため，各製品開発のプロジェクト開始時にトランクからブランチを切り，そのブランチ上で当該製品対応のソフトウェア部品の変更開発を行い，開発後にトランクへのマージを行うプロセスとした．

5.2.3.3. 部品開発計画策定

上記のプロセス変更は，複数の製品の開発が重なっても部品の開発は可能だが，マージ時の衝突という課題は残っている．そこで，「部品開発計画策定」というプロセスを設けることとした．

部品開発計画策定とは，当該製品で変更するソフトウェア部品に対し，他製品での開発計画の有無や開発内容を確認し，当該の製品開発ブランチにおける各ソフトウェア部品のマージ種別を定めるものである．

ここで，マージ種別は，更新（同じソフトウェア部品としてマージ），派生（異なるソフトウェア部品としてマージ），廃棄（当該製品専用の部品としてマージしない）の 3 つから選択する．各種別の選択基準は以下の通りである；

・更新：他製品での展開予定があり，衝突がない
・派生：他製品での展開予定があるが，衝突している
・廃棄：他製品での展開予定がない（当該製品固有）

更新を選んだソフトウェア部品はトランク上の開発元部品に単純マージ，派生を選んだソフトウェア部品はトランクに類似部品として新規登録，廃棄を選んだソフトウェア部品は製品開発ブランチに残し，コア資産とはしない．マージ方式で派生を選択した場合に，類似したソフトウェア部品が増えるという問題はある．しかし，本製品のプロダクトラインアーキテクチャが 2 章および文献[6]で示した通りソフトウェア部品を選択したものをビルド・リンクする方式を採用しているため，類似部品が数個程度であれば部品選択の間違いは生じにくい．

部品開発計画策定では，製品開発ブランチごとに，表 10 に示すブランチ帳票を作成する．ブランチ帳票は，縦軸にトランクに登録されている全ソフトウェア部品，横軸にソフトウェア部品ごとの版，前版からの変更点，ブランチ時点での他製品での利用状況（開発計画）を示した表である．本情報をもとに各ソフトウェア部品の当該製品での利用有無とマージ方式を定める．例えば，部品 A は他製品での利用状況はない（表中で”-”）ため，当該製品での開発内容をそのままマージする（更新とする），部品 C は製品 1 で開発中であり変更内容

も異なるため，派生部品としてマージ（新規登録）する，ということをブランチ時に計画する．

表 10 ブランチ・マージ帳票の例

部品名称	版	前版からの変更点	ブランチ時			マージ時	
			他製品での利用状況	利用有無	開発形態	開発実績	マージ可否
部品A	1.00	-	-	有	更新		
部品B-1	1.01	***	製品1	無	-		
部品B-2	2.00	部品B-1から派生	製品2	有	更新		
部品C	1.01	***	製品1	有	派生		

ソフトウェア部品の開発状況を他の開発者に周知することは，マージの衝突回避に有効である．本改善プロセスでは，上記のブランチ帳票をこの目的に使用している．具体的には，開発中の各製品のブランチ帳票をもとにソフトウェア部品ごとの開発状況を把握し，部品開発計画策定時にブランチ帳票の「他製品での利用状況」欄を自動更新する機能を実現している．同様の作業を手動で行う場合，各製品開発ブランチを調査し，開発実績および開発予定のあるソフトウェア部品を確認することとなるが，並行開発が多いためブランチ数も多くなり調査のための時間を要する．また，変更要求の確定が遅いため部品のマージ計画を見直すことも多いため，この作業を手作業で実行するのは現実的ではない．本作業を自動化することで，手動時と比べ調査および周知にかかる工数を 97.9%削減と試算している．

このように製品の開発開始時に製品開発ブランチを切り，ブランチ帳票を用いて部品開発計画を策定することで，①他製品の影響をあらかじめ考慮してソフトウェア部品を開発する，②他製品開発でのソフトウェア部品の開発状況を把握し開発完了後にできる限り衝突させずにマージする，の 2 点を可能とする．

5.2.3.4. マージ試験

多品種小変更型開発においては，問題②の通り，開発したソフトウェア部品を他製品に組み込んだ場合の動作保証をする必要があった．

そこで，開発したソフトウェア部品をマージする際に将来的に展開予定のある他製品への適合性を評価する「マージ試験」を設けている．図 29 はそのための具体的な環境である．

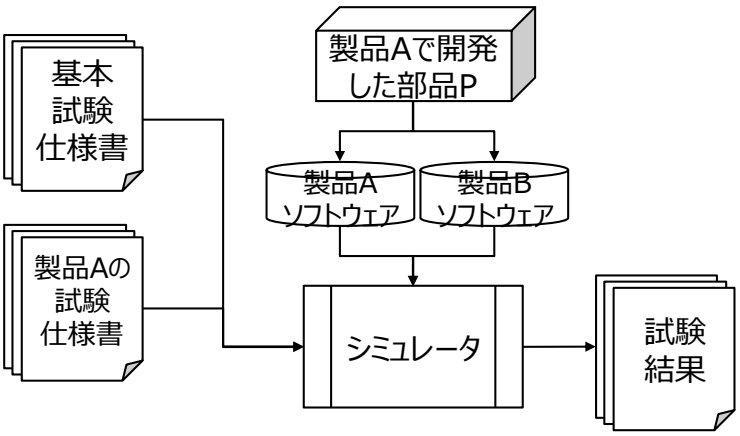

図 29 マージ試験環境

マージ試験では，トランクに保管されているあらかじめ準備された製品の基本試験仕様書と，製品開発ブランチにて開発した当該ソフトウェア部品用の試験仕様書に示された 2 種の試験を実施する．試験対象となるソフトウェアは，当該製品で開発した仕様を展開する可能性のある製品系列の最新版ソフトウェアとする．試験対象となる製品系列の選定は，ロードマップ（仕様の展開計画）や元となったソフトウェア部品の適用製品をもとに判断する．全製品系列への展開を想定するソフトウェア部品の場合，最大で 10 程度の製品で試験を実施する．例えば，製品 A で開発したソフトウェア部品 P を製品 B に将来展開する見込みがある場合は，製品 B の部品構成に，開発した部品を組み込んで先行的に試験を実施する．

マージ試験を製品の基本動作と当該ソフトウェア部品用の試験の 2 種を実施することで，他の製品部門に対し，開発したソフトウェア部品 P を組み込んでも製品が正常に動作し当該ソフトウェア部品が期待通りの機能を果たすことを保証している．これらの試験は事前に作成された試験仕様書に基づく自動実行を原則としているため，動作の保証にはほとんど人手を要しない．

マージ試験完了後に，対象となるソフトウェア部品のマージを実施する．マージに際しては，①マージしようとするソフトウェア部品が当初の部品開発計画に従い開発されているか，②他の製品開発ブランチにおいて，部品開発計画時に確認した通りにソフトウェア部品が開発されているか（マージの衝突が発生しうる割り込み開発がないか）を確認し，マージの衝突がない場合に速やかにリポジトリにマージする．マージの衝突が生じた場合は，再設計の上マージするか，別のソフトウェア部品として派生させて登録するかを検討し個別に対応する．

なお，本プロセスの導入後もソフトウェア部品の適合性に起因する不具合は 0 件を維持できており，マージを積極的に採り入れることによる品質への悪影響が生じていない．

5.3. 改善手法の評価

5.2.2 項で述べた改善前のブランチ・マージプロセスを 2010 年から適用し，2015 年から

5.2.3 項で改善したプロセスを導入している．改善手法適用開始直前の 2014 年度の各指数を基準として，各指標の推移を示し，適用の効果を示す．各年度のプロットは，各年度の開発で得られたデータの平均値とした．

5.3.1. マージコスト比率 Rc

問題①に示すように，多品種小変更型開発においては，製品メインブランチ上で開発が継続しやすく，本来コア資産とすべき変更がマージされず累積するという問題がある．本改善手法の構成管理プロセスとしての有効性を評価するため，改善前後の開発費に占めるコア資産のマージコストの割合に基づいて，マージ衝突を回避できているかという観点から評価する．ただし，改善前のプロセスでは，コア資産は先述の通り直接的にはマージされず，製品メインブランチとして変更が累積していたため，これらがマージされたとした場合のコストを仮想的にマージコストとする．仮想的なマージコストの算出は，製品メインブランチ上にあるマージ対象（開発形態を「更新」としたソフトウェア部品のうち衝突しているもの）の変更量と，変更量当たりのマージコストの平均値（実績値）から算出する．本マージコストにはマージ試験にかかるコストは含まない[i]．マージコスト比率は，次式とする．

Rc= A / B (1)

A: コア資産マージコスト

B: 当年度のソフトウェア開発費

コア資産マージコスト A は改善前後のプロセスにおいて以下のとおりである．
改善前：ブランチ・マージプロセスに実際にかけたコスト＋仮想的なマージコスト
改善後：ブランチ・マージプロセスに実際にかけたコスト（マージ試験含む）

なお Rc を年度ごとに比較するため，2014 年を 1.00 とした相対値 Rc-yy で示す．Rc-yy は次式と定義する．

Rc-yy= yy 年の Rc / 2014 年の Rc (2)

Rc-yy の推移を図 30 に示す．

i ここでのコストは人の工数のみを考えており，マージ試験は自動実行のため人の工数は 0 としている．

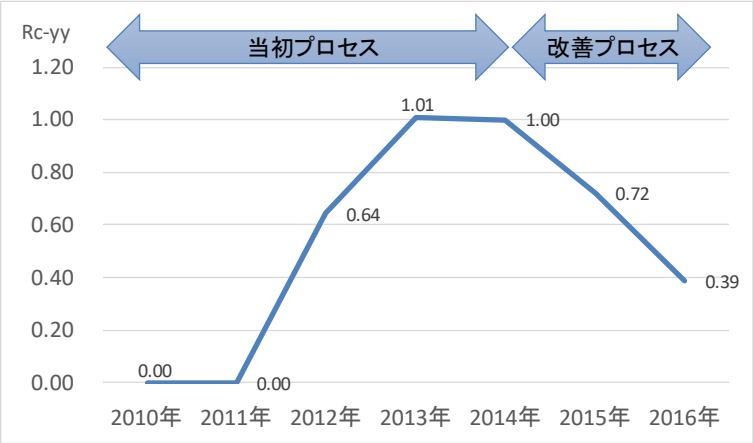

図 30 マージコスト比率

当初プロセスにおいては，毎年一定の割合で，並行開発間での衝突が生じ，トランクへのマージが行われなかったためコア資産にマージすべき変更が累積していたが，改善プロセス導入後は，15 年に 28%減，16 年に 61%減となっている．これは，当初プロセスにおいては，ソフトウェア開発者の緊密なコミュニケーションや日程調整だけでは開発の衝突を回避できずマージすべき内容が累積してしまったのに対し，改善プロセス導入後は，開発の衝突を防止することができているといえる．また，マージすべきソフトウェア部品は当年度中にマージできており，本改善手法が有効であると評価できる．

5.3.2. 製品メインブランチ数 Nb

プロセスの改善を行った結果，コア資産が派生しなかったことを確認する．改善前のプロセスにおける派生したコア資産は製品メインブランチであり，改善後のプロセスにおける派生したコア資産はマージされなかった場合の製品開発ブランチが製品メインブランチとなる．このことから改善前後の製品メインブランチ数 Nb が増加しなかったことを確認することで，コア資産が一つ（トランクのみ）に維持できたことを検証する．

製品メインブランチ数の推移を図 6 に示す．

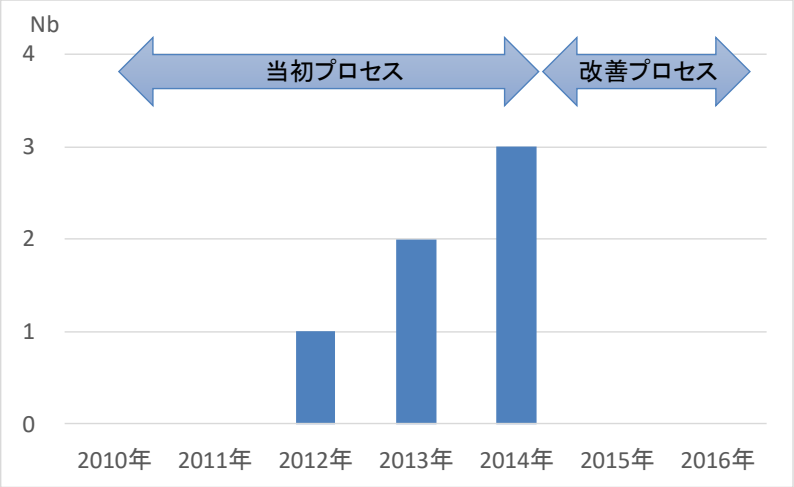

図 31 製品メインブランチ数

当初プロセスにおいては，年を追うごとにブランチ数が増加し，2014 年に 3 本の製品メインブランチが存在していた．これに対し，プロセス改善以降は，製品メインブランチ（マージされなかった製品開発ブランチ）は存在しておらず 2 年後も 0 本のままであり，目論見通りコア資産の派生を防ぐことができている．これは，各製品部門が製品メインブランチを囲い込み，固有のコア資産を形成する必要がなくなったことを意味している．

このようにコア資産の派生を 0 で維持できた要因は，5.3.1 項でも議論した通り，マージ計画を立てることでマージの衝突が回避しやすくなったことに加え，他製品で開発したソフトウェア部品の品質が保証されているためコア資産に統合しやすくなったことが主要因と考える．

5.4. 妥当性への脅威

本章で実践した他開発の状況を可視化してブランチ・マージする手法は，コア資産開発の計画性の高さや（計画通り開発できること，計画外の開発が発生しないこと）が重要となる．本章の対象である空調機は，他社動向を見ながら仕様変更する特性を持ちあわせているため，計画外の開発が生じやすく，マージ時に想定外の衝突が発生しているケースも散見された．このような状況を解決するためには，Fine-grained revision control[58]のような細やかにブランチ・マージを繰り返す手法も有効と考える．本適用対象では，製品部門が製品ごとにソフトウェアを管理したいという要望があるため製品ごとのブランチ・マージという手法を採用したが，マージの衝突が一定数残っているため，先述の Fine-grained revision control と組み合わせた手法も検討の余地がある．

また，マージ試験では，開発したソフトウェア部品を将来どの製品系列に組み込むかを想定して，事前に適合性を評価している．製品への組み込みが予想できない，もしくは製品系列が多すぎて評価が困難な場合に本手法の適用が難しいという事態も想定されるが，本適

用対象では，製品系列は 10 程度と大きく増加しておらず，製品開発のたびにビルド対象の製品系列が増えるといった問題は発生していない．

5.5. 関連研究

5.5.1. ソフトウェアマージの衝突回避

ソフトウェアマージの衝突については，衝突の検知・回避[55][59]，マージ技術およびツール[60][61]の各分野で様々な研究がなされており，Mens らが各手法を整理している[57]．

衝突の検知に，グラフ分割という手法がある[57]．グラフ分割とは，ソフトウェア構成要素（クラスやソフトウェア部品）間の依存関係をグラフによって可視化する手法である．衝突する可能性のある構成要素を見つけ出すことに優れるが，頻繁に依存関係が変更される場合のグラフ更新に負荷が生じる．また，衝突の回避の技術にカプセル化がある[57]．カプセル化とは変数やメソッドを隠蔽することで，変更の伝播を局所化しマージ時の衝突を減らすものである．メソッド等の境界を越えた変更の場合，効果は限定的である．

この他にも衝突を回避する手法に Fine-grained revision control がある[58]．Fine-grained revision control は，一回あたりの変更量をごく小さくし，小さな開発を積み上げていくことで，マージの衝突を回避する．開発規模の大きな変更が並行した場合に，マージが衝突しやすく多大なコストがかかる傾向にある．Adams らはワークステーションの開発において，Fine-grained revision control をベースとした手法を用い，短期間に小さな開発を実施しマージすることの有効性を述べている[58]．機能ごとにブランチを設けて開発を進めマージをする手法は，Git-flow や Github-flow でも用いられている[62][63]．1 回あたりの変更量をごく小さくし，小さな開発を積み上げていく手法は有効な手法の一つと考えるが，多品種小変更型開発においては，市場ごとに異なる製品部門（課）が各製品の変更点を管理するという特徴があるため，製品開発組織および製品開発プロセスとの親和性において，製品開発ブランチ内で変更点が管理できる点に優位性がある．

マージ時の衝突を回避する手法の一つにお互いの変更を周知するという方法がある．Brun らは，マージの衝突を特定・管理・回避することを支援する Crystal というツールを提案し，複数のエンジニアが協調的に開発する場合のコストや難易度を下げる実践的な取り組みであると評価している[64]．ソフトウェア部門の関心事であるマージの衝突を回避できる点において Brun らの取組みは優れているが，同時に，多品種小変更型開発においては製品部門の関心事である自製品向けの最適化開発を実現できる必要がある．本章での取り組みは，マージの衝突を回避するだけでなく製品ごとの最適化開発も実現することが可能である．

5.5.2. SPLE の品質保証

SPLE のアプリケーション開発における品質保証はテストを中心とした事例が数多く報告されている[65].

SPLE におけるテスト手法の一つに，ドメイン開発やアプリケーション開発にて開発したテストケースを再利用する手法がある．この手法は，ドメイン開発や他アプリケーション開発で開発したテストケースを固有のアプリケーション開発に適合させて再利用するもので，多くの事例がある[66].

多品種小変更型開発における製品部門にとっては，開発されたコア資産が自製品に適合することが重要であり，製品部門がコア資産の利用を判断する前に，適合性を評価している必要がある．いずれのテスト手法も，アプリケーション開発の開始前にコア資産の適合性を保証することはできないが，我々の取組みでは将来の部品適用を想定して先行的に試験しているため，アプリケーション開発の開始前にコア資産の適合性が保証可能である．

5.6. ブランチ・マージプロセスにおける改善と考察のまとめ

本章では，並行開発が多い業務用空調機への 5 年の SPLE 適用経験におけるコア資産の派生（統合されない）の原因を分析し，製品部門によるコア資産の固有化および未評価部品の利用回避の 2 つの問題によることを見出し，これらを解決するためにブランチ・マージプロセスの改善を行なった.

改善プロセスは，ブランチ時に衝突回避のための計画を立てる作業と，マージ時に部品の適合性を確認する作業からなる．この改善プロセスを適用した結果，マージの衝突を回避することに成功し，マージコスト比率を 61%削減，コア資産ブランチ（製品メインブランチ）も 0 本を維持できている．この結果，3 章および文献[45]にも示した通りソフトウェア部品再利用率の向上も実現できた.

これらの結果に基づき，構成管理プロセスは，製品開発組織および製品開発プロセスとの親和性があるブランチ・マージプロセスを構築することが重要であるとの知見を得た.

第6章

まとめ

本論文では，SPLE 成功のための要件とその要件の実装が不十分だった場合に引き起こされる問題および解決策について議論した．

2 章で述べた通り，SPLE には 29 個の技術領域があり，これらが正しく実践されない場合に，コア資産であるソフトウェア部品の再利用率が低下することを明らかにした．特に，①上流工程からの体系的な再利用プロセスが不十分，②可変性管理が不十分（仕様差異を生み出すフィーチャの増加に対して構成選択が難しくなる），③構成管理（並行開発時のブランチ・マージ）が不十分，という 3 つのプロセス実装不足があった場合にコア資産の再利用率が低下することをデータから導いた．

3 章では，一つ目の課題である「上流工程からの体系的な再利用プロセスが不十分」に対する改善プロセスを提案し，改善前後の結果を定量的に比較検証した．コア資産保守・製品導出における改善プロセスは，①開発ロードマップからアプリケーション開発を 1 つの幹開発と複数の枝開発に分離，②幹開発の中で，個別製品の開発を行いつつ，後続する枝開発のバリエーションを吸収可能なソフトウェア部品とそのソフトウェア部品と関連付けた要求仕様書マスタを作成，③枝開発の中で，その要求仕様書マスタからのカスタマイズとソフトウェア部品の組立て，を実現する．

コア資産保守・製品導出プロセスの改善によって，仕様の確度を高めたのちにコア資産を開発し後続する開発でコア資産が再利用できるようになった．この結果，類似部品発生率が 66.7%抑制でき，コア資産の再利用率や製品群全体の生産性も向上した．

4 章では，「可変性管理が不十分（仕様差異を生み出すフィーチャの増加に対して構成選択が難しくなる）」という課題に対して，要求仕様書の可変性記述とソフトウェアの可変性の構成手段の 2 つの見直しを図り，改善手法の評価を行っている．本改善は，要求仕様書に製品-機能マトリクスとデータ仕様書を導入することで，可変性記述の一覧性と可読性を向上させるとともに，これら可変性記述から製品ソフトウェアへの実装・選択方法を仕様設計時点で明確にし，かつビルドへ直結する構成手段を提供するものである．

可変性管理を見直すことによって，ソフトウェアへの理解の浅い製品部門とコア資産を共有することに成功し，仕様共通率を約 2.5 倍に高めている．さらに構成決定パラメータ数を 56%低減し，構成決定を容易化している．

5 章では，3 つ目の課題である「並行開発時の構成管理（ブランチ・マージ）が不十分であり，コア資産が維持できない」に対して，①アプリケーション開発開始時に開発予定

のソフトウェア部品のブランチ状況を確認しマージ計画を検討すること（部品開発計画策定），②開発終了後に将来の製品適用を見据え変更したソフトウェア部品を使用する他製品系列の動作保証を行うこと（マージ試験），を提案し，改善前後での比較検証結果から議論を行った．

ブランチ・マージプロセスを改善した結果，マージの衝突を回避することに成功し，マージコスト比率を 61%削減，コア資産ブランチ（製品メインブランチ）も 0 本を維持できている．

これらの議論をまとめると表 11 のようになる．

表 11 SPLE の実装が不十分な場合に生じる問題と改善策のまとめ

章	実施が不十分な場合に生じる問題	改善提案	効果	技術領域
3 章	コア資産の再利用/開発の判断が誤りやすい（不必要な類似部品の発生を増やしコア資産が劣化する）	①各製品開発を幹開発と枝開発に分離 ②幹開発でのコア資産開発 ③枝開発での再利用	類似部品発生率 66.7%抑制	要求工学 ソフトウェア統合技術計画
4 章	仕様差異を生み出す特徴（フィーチャ）の増加に対して構成選択が難しくなる	①要求仕様書の可変性記述見直し ②ソフトウェアの構成手段見直し	構成決定容易化（仕様共通化率 2.5 倍，構成決定パラメータ数 56%削減）	要求工学 部品開発 構成管理（可変性管理）
5 章	マージの衝突回避のためコア資産の固有化が進む． 他製品の変更がマージされたコア資産の利用を控える	①ブランチ時の部品開発計画策定 ②マージ試験	マージコスト比率 61%減 製品メインブランチ（コア資産の派生）0 件	テスト 構成管理（並行開発管理）

空調機（室外機）への適用結果の分析から，長期的に SPLE を成功させるためには，①コア資産の保守・運用管理（コア資産の開発方法および体系的な再利用を促す仕組み），②可変性管理（可変性の共有と，可変性に合わせた構成手段の選択），③並行開発の管理（ブランチ・マージプロセス），の 3 つの技術管理が重要であると考える．

これらをまとめると，多品種小変更型組込みソフトウェア開発への SPLE フレームワークは図 32 の通りとなる．

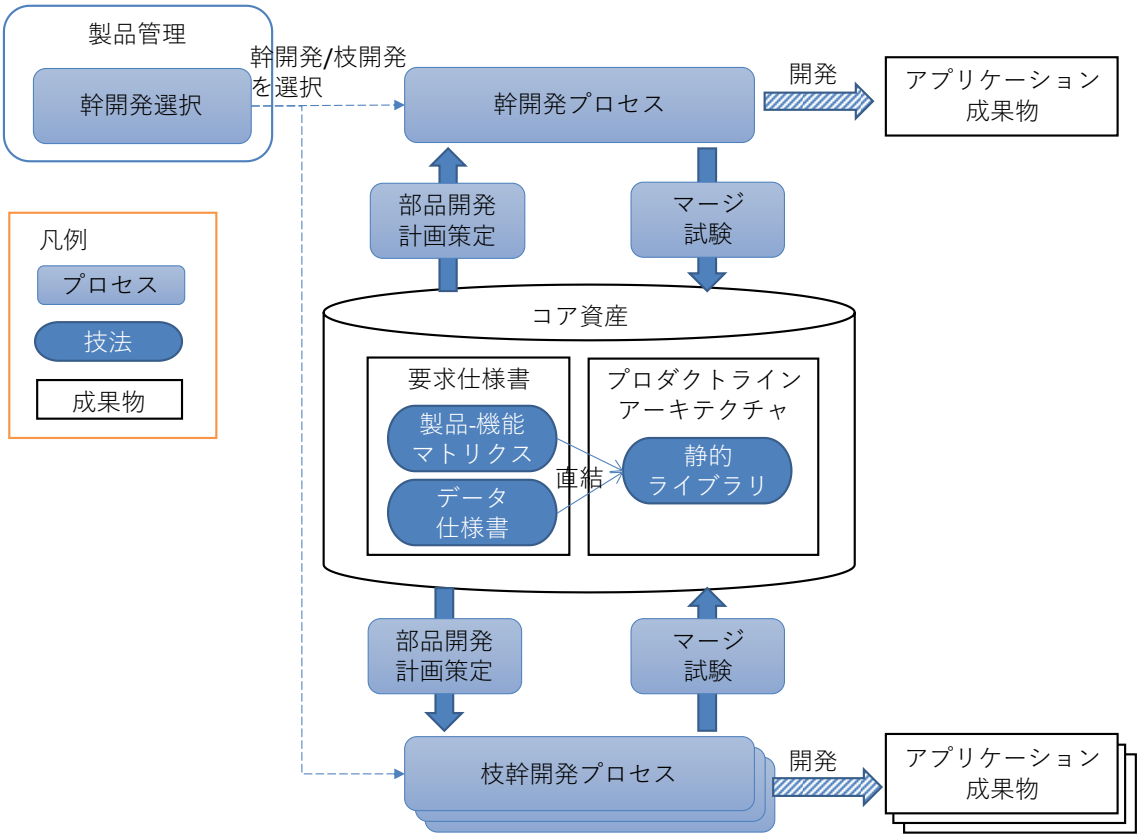

図 32 多品種小変更型開発における SPLE フレームワーク

多品種小変更型開発においては，3 章に記載の通り個々のアプリケーション開発（製品開発）における仕様確定が非常に遅く，コア資産をプロアクティブに開発することが難しい．そのため，今後のベースとなるコア資産を開発する幹開発と，開発したコア資産を再利用することを前提とした枝開発に分離する幹開発選択を設ける．幹開発においては，製品開発を行いつつも，後続する枝開発で利用可能なコア資産の開発を行う．

各開発においては，製品部門によるコア資産の固有化やコア資産の利用回避を防ぐため，コア資産をブランチする際に部品開発計画策定によってマージの衝突の回避を図るとともに，製品開発完了時にマージ試験によってマージするコア資産が他製品においても利用可能であることをあらかじめ評価する（詳細 5 章）．

これらの開発で用いられるコア資産は，製品 - 機能マトリクスおよびデータ仕様書で構成される要求仕様書と静的ライブラリ方式で実現されたプロダクトラインアーキテクチャから成る．これらの成果物は表記法と実現手段が一対で直結しており，製品機能を選択すると該当するソフトウェア部品が選択される仕組みとなっている．これによってソフトウ

ェアを解さない製品部門とも可変性を共有し，コア資産の再利用が図られることを狙っている（詳細 4 章）.

本フレームワークは，多品種小変更型開発の特徴を持つ空調機（室外機）を題材にプロセス改善の前後で比較検証をしているため，一定の汎用性があるものと考えるが，組織構成やソフトウェア開発の言語および規模によっても SPLE 適用時の成否が変わるためさらなる検証が必要である.

また，本適用対象においては，プロトタイピングやアジャイル開発との組み合わせ，ブランチ・マージプロセスのさらなる改善等によって，再利用性や生産性の改善が可能な余地があるとみている．今回比較検証で得られた知見に加え，近年の技術動向も加味しながら，組込み産業の発展のため，再利用技術の昇華と一般化に向けて研究開発を進めていきたい.

参考文献

[1] 吉村, “製品間を横断したソフトウェア共通化技術-ソフトウェアプロダクトラインの最新動向”, 情報処理, Vol.48, No.2, pp.171-176, 2007.

[2] 独立行政法人情報処理推進機構, “組込みソフトウェア開発データ白書 2019”, p39, 2019.

[3] 岸知二, “ソフトウェア再利用の新しい波-広がりを見せるプロダクトライン型ソフトウェア開発-: 0. 巻頭言: プロダクトライン開発と再利用技術”, 情報処理, Vol.50, No.4, pp.265-267, 2009.

[4] Garlan, D., Allen, R., and Ockerbloom, J., “Architectural Mismatch: Why Reuse is so Hard”, IEEE Software, Vol.12 No.6, pp.17-26, 1995.

[5] Clements, P., and Northrop L., “Software Product Lines: Practice and Patterns”, p.608, Addison Wesley, Reading, MA, 2001. 前田卓雄訳: ソフトウェアプロダクトライン, p.652, 日刊工業新聞社, 2003.

[6] Nagamine, M., Nakajima, T., and Kuno, N., “A Case Study of Applying Software Product Line Engineering to the Air Conditioner Domain”, the 20th International Systems and Software Product Line Conference, pp.220-226, Beijing, China, 2016.

[7] Clements, P., Northrop, N., et al., “A Framework for Software Product Line Practice, Version 5.0.”, White Paper of CMU Software Engineering Institute, http://www.sei.cmu.edu/productlines/frame_report/index.html, 2012.

[8] “Product Line State of the Practice Report”, CMU/SEI-2002-TN-017, 2002.

[9] “Product Line Engineering: The State of the Practice”, IEEE Software, pp.52-60, 2003.

[10] Dager, J. C., “Cummins’s Experience in Developing a Software Product Line Architecture for Real-Time Embedded Diesel Engine Controls”, 1st Conference on Software Product Lines, pp.23-45, Denver, CO, 2000.

[11] Wijnstra, J. G., “Component Frameworks for a Medical Imaging Product Family”, In International Workshop on Software Architectures for Product Families, pp.4-18, Berlin, Heidelberg, 2000.

[12] 小田川, “発電監視制御システムにおけるプロダクトライン構築事例”, JEITA 組込み系ソフトウェア・ワークショップ 2010, 2010.

[13] 平鍋, 野中, 及部, “アジャイル開発とスクラム 顧客・技術・経営をつなぐ協調的ソフトウェア開発マネジメント”, 翔泳社, 2021.

[14] Haidar, H., Kolp, M., and Wautelet, Y., “Agile Product Line Engineering: The AgiFPL Method. In ICSOFT”, pp.275-285, 2017.

[15] J. Díaz, et al., “Agile Product Line Engineering- A Systematic Literature Review”, Software: Practice and Experience, Vol.41, No.8, pp.921-941, 2011.

[16] 林, “ソフトウェアプロダクトラインのアジャイル開発方法論に関する研究”, 2018.

[17] Marques, M., et al., “Software Product Line Evolution: A systematic Literature Review”, Information and Software Technology, Vol.105, pp.190-208, 2019.
[18] 吉村, 宮崎, 横山, “オブジェクト指向組み込み制御システムのモデルベース開発法”, 情報処理学会論文誌, Vol.46, No.6, pp.1436-1446, 2005.
[19] Thomsen, T., “Integration of International Standards for Production Code Generation”, SAE transactions, pp.338-349, 2003.
[20] Botterweck, G., et al., “EvoFM: Feature-Driven Planning of Product-line Evolution”, In Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering, pp. 24-31, Cape Town, South Africa, 2010.
[21] Pleuss, A., et al., “Model-driven Support for Product Line Evolution on Feature Level”, In Journal of Systems and Software, Vol.85, No.10, pp.2261-2274, 2012.
[22] Pohl, K., Günter, B., Van der Linden, F. J., “Software Product Line Engineering: Foundations, Principles and Techniques”, 2005
[23] 吉村, ダルマリンガム ガネサン, ディルク ムーティック, ”プロダクトライン導入に向けたレガシーソフトウェアの共通性・可変性分析法”, 情報処理学会論文誌, Vol.48, No.8, pp.2482-2491, 2007.
[24] 城谷, 岸, “SPL 開発におけるペアワイズ法を用いたテスト手法について”, 第 78 回全国大会, pp.319-320, 2016.
[25] 田原, 野田, “ユースケースからのフィーチャモデル導出の提案”, 情報処理学会研究報告, Vol.2014-SE-183, No.3, pp.1-8, 2014.
[26] Van der Linden, F. J., Klaus, S., and Eelco, R., “Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering”, Springer Science & Business Media, 2007.
[27] Pohl, K., Günter, B., and Van der Linden, F. J., “ソフトウェアプロダクトラインエンジニアリング”, 株式会社エスアイビー・アクセス, 2009.
[28] Capilla, R., Bosch, J., and Kang, K. C., “Systems and Software Variability Management, Concepts Tools and Experiences”, Concepts Tools and Experiences, 2013.
[29] ISO/IEC 26550:2015, “Software and Systems Engineering - Reference Model for Product Line Engineering and Management”, International Organization for Standardization, 2015.
[30] Buhrdorf, R., Churchett, D., and Krueger, C. W., “Salion’s Experience with a Reactive Software Product Line Approach”, International Workshop on Software Product-Family Engineering, pp.317-322, Siena, Italy, 2003.
[31] 西, “製品品質の決定要因としての組込みソフトウェアと組込みソフトウェア・クライシス”, 品質, 日本品質管理学会誌 Vol.34, No.4, pp.343-349, 2004.
[32] Klaus, S., Verlage, M., “The Economic Impact of Product Line Adoption and Evolution”, IEEE Software, Vol.19, No.4, pp.50-57, 2002.

[33] Rubin, J., Czarnecki, K., and Chechik, M., “Managing Cloned Variants: A Framework and Experience”, the 17th International Software Product Line Conference, pp.101-110, Tokyo, Japan, 2013.
[34] Fogdal, T., Scherrebeck, H., Kuusela, J., Becker, M., and Zhang, B., “Ten Years of Product Line Engineering at Danfoss: Lessons Learned and Way Ahead”, the 20th International Systems and Software Product Line Conference, pp.252-261, Beijing, China, 2016.
[35] Adam, S., Klaus, S., “Effective Requirements Elicitation in Product Line Application Engineering: An Experiment”, International Working Conference on Requirements Engineering: Foundation for Software Quality, pp.362-378, Essen, Germany, 2013.
[36] Czarnecki, K., Helsen, S., and Eisenecker, U., “Staged Configuration Using Feature Models”, International Conference on Software Product Lines, pp.266-283, Boston, MA, 2004.
[37] 野田, 岸, “プロダクトライン開発における可変性のモデル化手法”, コンピュータソフトウェア, Vol.31, No.4, pp.66-76, 2014.
[38] Faulk, S. R., “Product-Line Requirements Specification (PRS): An Approach and Case Study”, Fifth IEEE International Symposium on. IEEE, pp.48-55, 2001.
[39] Stoiber, R., and Glinz, M., “Modeling and Managing Tacit Product Line Requirements Knowledge”, Managing Requirements Knowledge, Second International Workshop, pp.60-64, IEEE, 2009.
[40] Loesch, F., Ploedereder, E., “Optimization of Variability in Software Product Lines”, 11th International Software Product Line Conference, pp.151-162, Kyoto, Japan, 2007.
[41] John, I., “Using Documentation for Product Line Scoping”, IEEE Software, Vol.27, No.3, pp.42-47, 2010.
[42] 野田, “ソフトウェア再利用の新しい波-広がりを見せるプロダクトライン型ソフトウェア開発-: 2. プロダクトラインの可変性管理-可変性のモデル化とアーキテクチャ設計”, 情報処理, Vol.50, No.4, pp.274-279, 2009.
[43] Gacek, C., and Anastasopoules, M., “Implementing Product Line Variabilities”, In Proceedings of the 2001 Symposium on Software Reusability, pp.109-117, Toronto Ontario, Canada, 2001
[44] Bosch, J., et al., “Variability Issues in Software Product Lines”, In International Workshop on Software Product-Family Engineering, pp.13-21, Bilbao, Spain, 2001.
[45] 長峯, 中島, “多品種小変更型開発におけるコア資産保守・製品導出手法の改善と実践”, デジタルプラクティス, Vol.10, No.3, pp.639-655, 2019.
[46] Czarnecki, K., et al., “Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches”, In Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, pp.173-182, Leipzig, Germany, 2012.

[47] Kang, K. C., et al., “Feature-Oriented Domain Analysis (FODA) Feasibility Study”, Carnegie-Mellon University Software Engineering Inst, 1990.
[48] Svahnberg, M., Van Gurp, J., and Bosch, J., “A Taxonomy of Variability Realization Techniques”, Software: Practice and Experience, Vol.35, No.8, pp.705-754, 2005.
[49] Kästner, C., “Virtual Separation of Concerns”, toward preprocessors 2.0. it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik, Vol.54, No.1, pp.42-46, 2012.
[50] Neighbors, J. M., “Draco: A Method for Engineering Reusable Software Systems”, Software Reusability, Vol.1, pp.295-319, 1989.
[51] Czarnecki, K., Helsen, S., and Eisenecker, U., “Staged Configuration Through Specialization and Multilevel Configuration of Feature Models”, Software Process: Improvement and Practice, Vol.10, No.2, pp.143-169, 2005.
[52] Classen, A., Hubaux, A., and Heymans, P., “A Formal Semantics for Multi-level Staged Configuration”, VaMoS, Vol.9, pp.51-60, 2009
[53] Greenfield, J., and Short, K., “Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools”, In Companion of the 18th annual ACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and Applications, pp.16-27, Anaheim, CA, 2003.
[54] Deelstra, S., Sinnema, M., and Bosch, J., “Product Derivation in Software Product Families: A Case Study”, Journal of Systems and Software, Vol.74, No.2, pp.173-194, 2005.
[55] Tichy, W. F., “Tools for Software Configuration Management”, In Proceeding of the International Workshop on Software Version and Configuration Control, pp.1-20, Grassau, Germany, 1988.
[56] Munson, J. P., and Dewan, P., “A Flexible Object Merging Framework”, In Proceedings of the 1994 ACM conference on Computer Supported Cooperative Work, pp.231-242, Chapel Hill, NC, 1994.
[57] Mens, T., “A State-of-the-Art Survey on Software Merging”, IEEE Transactions on Software Engineering, Vol.28, No.5, pp.449-462, 2002.
[58] Adams, E., et al., “SunPro Engineering a Practical Program Development Environment”, In Advanced Programming Environments, pp.86-96, 1987.
[59] Feather, M. S., “Detecting Interference When Merging Specification Evolutions”, In Proceedings of the 5th International Workshop on Software Specification and Design, pp.169-176, Pittsburgh, PA, 1989.
[60] Leßenich, O., Apel, S., and Lengauer, C., “Balancing Precision and Performance in Structured Merge”, Automated Software Engineering, Vol.22, No.3, pp.367-397, 2015.
[61] Ghiotto, G., et al., “On the Nature of Merge Conflicts: A Study of 2,731 Open Source Java Projects Hosted by Github”, IEEE Transactions on Software Engineering, Vol.46, No.8, pp.892-915, 2018.

[62] Kalliamvakou, E., et al., “The Promises and Perils of Mining GitHub”, In Proceedings of the 11th Working Conference on Mining Software Repositories, pp.92-101, Hyderabad, India, 2014.
[63] Dwaraki, A., et al., “GitFlow: Flow Revision Management for Software-Defined Networks”, In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, pp.1-6, Santa Clara, CA, 2015.
[64] Brun, Y., et al., “Proactive Detection of Collaboration Conflicts”, In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp.168-178, Szeged, Hungary, 2011.
[65] Metzger, A., and Pohl, K., “Software Product Line Engineering and Variability Management: Achievements and Challenges”, In Future of Software Engineering Proceedings, pp.70-84, Hyderabad India, 2014.
[66] Engström, E., and Runeson, P., “Software Product Line Testing–A Systematic Mapping Study”, Information and Software Technology, Vol.53, No.1, pp.2-13, 2011.