博 士 論 文

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

D2017SE001　林 健吾

指導教員　青山 幹雄

2018 年 2 月

南山大学大学院　理工学研究科　ソフトウェア工学専攻

An Agile Development Methodology for Software Product Line

D2017SE001　Kengo Hayashi

Supervisor Mikio Aoyama

February 2018

Graduate Program in Software Engineering
Graduate School of Science and Engineering
Nanzan University

要約

近年のソフトウェアシステムの開発には，製品のグローバル化や市場要求の高度化，複雑化により，多様性と俊敏性が競争の鍵となっている．多様性への対応には SPLE（Software Product Line Engineering）が提案されている．俊敏性への対応には ASD（Agile Software Development）が提案されている．多様性と俊敏性に同時に対応するために両方のアプローチを統合した APLE（Agile Product Line Engineering）が研究されているが，開発方法論として確立されているとはいえない．

本研究では製品の俊敏な進化と多様性の実現を包括的に解決するために，SPL（Software Product Line）の開発全体を包括した管理可能な開発を実現する開発方法論を提案する．開発全体を包括管理するためには組織が保有する開発資源のコントロールが課題となる．提案した開発方法論は開発資源の各ポートフォリオを管理された状態に導くことで複数 SPL の包括管理を実現する. SPL に管理性を備えるために ASD を統合してアジャイルマネジメントを適用する．マネジメントを支えるテクノロジー領域として，SPLE における開発プロセス，プロダクト可変性，アーキテクチャの 3 領域の課題に分解して解決する．

SPLE の開発プロセスでは，可変点を中心とした反復性の高いプロセスが実行されることに着目し，2 層のイテレーションプロセス構造を備えた SPLE のための ASD を提案する．これまで，ASD では短期開発プロジェクトへの適用が想定されていない. そこで，複数の開発プロジェクトを包括的に開発対象とすることで ASD を SPLE のアプリケーション開発に適用可能とし，生産性の測定や開発量の推定能力を高めることで開発の包括的な管理性を向上する．

プロダクト可変性に関しては，可変性をモデル化して構造化することで可変点の開発の順序制約を明らかにし，独立した開発単位の集合にプロジェクトを分割する方法を提案する．開発プロセスを一括したウォータフォール型~~開発~~から分割した反復型の ASD に移行することで，テスト環境などの資源消費を平準化して資源不足による開発量の変動を抑え，開発全体の管理性を向上する．

アーキテクチャに関しては，アプリケーション開発をドメイン開発と並行開発できるアーキテクチャへのリファクタリング方法を提案する．これによって，可変性を開発ビューレベルでモジュール化し，ドメイン開発とアプリケーション開発間や，アプリケーション開発アクティビティ間の衝突を抑制し，開発アクティビティの並行化を実現する．並行開発が実現できることで，開発資源の割当て制約が軽減され，開発の管理性を向上する．

提案した APLE の開発方法論を自動車システムのソフトウェア開発の実プロジェクトに適用し，その結果から，多様な製品を俊敏に開発する提案開発方法論の有効性と妥当性を示す．

本研究ではポートフォリオマネジメントを指向した APLE の開発方法論を提供することで，各分野の SPL のコンテキストへの適用を可能とし，SPLE を運用する組織に対して包括的に管理可能な開発活動の推進に貢献する．

本研究の成果はソフトウェア工学の発展に関して，次の 3 つの点で貢献する．

(1) 従来，個別に研究されてきた SPLE と ASD の 2 領域を統合した新たな開発方法論を創出したこと．
(2) APLE を実現する開発方法論を，プロセス，アーキテクチャ，マネジメントにわたる包括的技術体系として提案していること．
(3) 提案開発方法論を実際の自動車ソフトウェアプロダクトライン開発に適用し，多面的な評価尺度を設定してデータを収集し，その効果を定量的に評価していること．

Abstract

In recent years, the variability and agility are increasingly important for the development of software systems due to globalization of products and sophistication and complication of market demands. SPLE (Software Product Line Engineering) approach has been proposed to deal with variability, while ASD (Agile Software Development) has been proposed to accelerate agility. To meet both variability and agility, APLE (Agile Product Line Engineering), an integration of both approaches, has been proposed. However, it is not established yet as a development methodology.

In this research, to accommodate both agility and variability in product releases, the authors propose a development methodology that makes the entire development of SPL (Software Product Line) manageable. The authors introduce a portfolio management as a driver to improve manageability of development for SPL. The proposed development methodology reconstructs the development process, product variability and architecture in SPLE and refines the granularity to the predictable development amount of software products and stable productivity.

In the development process of SPLE, the authors focus on the fact that the highly iterative process centered around the variation point is executed. And, the authors propose an agile development method for SPLE with twofold iterative process structure. Conventionally, ASD is not expected to be applied to short-term development projects. Therefore, embracing multiple projects into a development makes ASD applicable to SPLE application engineering, and improves the total management of the development by improving the productivity monitoring and estimation of development efforts.

In product variability, the authors propose a method to divide a project into a set of independent development units by clarifying the ordering constraints of variation point development by modeling and structuring the variability. By reengineering the development process from a single water-fall to multiple incremental, the authors will enable to reduce resource consumptions, including test environment, equalize the fluctuation of development volume arising from the shortage of resources, and improve manageability of the entire development.

In architecture, the authors propose a refactoring method to architecture, which enables to parallelize application engineering with domain engineering. By modularizing variability at the development view level, possible collisions between domain engineering and application engineering, and between application engineering activities are mitigated, and parallelization of development activities are enhanced. Increasing concurrency in the development reduces the constraints in resource allocations, and improves the manageability of the entire development.

The authors applied the proposed APLE development methodology to the actual project in software development of automotive software systems, and proved the effectiveness and validity of the proposed development methodology in the development of evolvable and variable products from the result.

目次

1 はじめに

1.1 研究の背景

SPLE（Software Product Line Engineering）は，低コスト，高品質で SPL（Software Product Line）における多様な製品を開発するアプローチである[10][39]．SPLE では，ドメイン開発とアプリケーション開発の 2 つの開発領域で製品の多様性を低コストで実現するアプローチをとる．ドメイン開発では，SPL における共通性と可変性を分析してコア資産を構築する．アプリケーション開発では，コア資産から個別製品を開発する．要求の多様性に対応するために，SPLE は自動車ソフトウェア開発を含む多くのソフトウェアシステム開発に導入されている[23][49].

複数の SPL を並行して開発する MPLE（Multiple Product Line Engineering）が実践されている[46]．例えば，自動車ソフトウェア開発において，Bosch のガソリンシステム開発では市場セグメントごとに分けて SPL を構築している[35]．その他の分野でも，コンポーネントベースのアーキテクチャにおいて，相互作用や相互依存するコンポーネントごとにプロダクトラインを構築し，それらを組合わせることで，マルチプロダクトラインが構成されている[48].

自動車システムを含むソフトウェアシステムの開発では，俊敏な製品進化が求められている．俊敏な製品進化に対しては，短いサイクルでインクリメンタルに製品を進化させる ASD（Agile Software Development）が実践されている[9][18][29]. 製品の多様性とその俊敏な進化に対応するため, ASD と SPLE を統合した APLE（Agile Product Line Engineering）も実践されている[11][18].

製品の多様性とその俊敏な進化に対応するための開発方法の研究開発が様々な分野で推進されている．しかし，製品のグローバル化や市場要求の高度化，複雑化により[12]，SPLE におけるドメイン開発とアプリケーション開発が並行化するなど，さらなる俊敏性が必要となり，従来の開発方法では対応できない状況となっている．

1.2 研究の目的

SPLE において複雑化する製品の多様性とより俊敏な進化への要求に対処するために，ドメイン開発とアプリケーション開発を並行開発するコンテキストにおいて，SPL 全体を包括的に管理可能にする新たな開発フレームワークが求められている．本研究では，SPLE の開発活動を管理するために，ポートフォリオマネジメントを指向した APLE の開発方法論を提案する．方法論の提案過程において SPL 全体の開発ポートフォリオマネジメントの必要性を示す．提案する開発方法論を自動車システムのソフトウェア開発の実開発に適用し，本開発方法論の有効性を示す.

提案する開発方法論では，製品の多様性と開発の俊敏性の全体最適を図るための開発フレームワークのモデルを，ポートフォリオマネジメントの概念を拡張して定義する．本モデルにおけるポートフォリオマネジメントの管理性の向上を狙い，開発資源の割当ての平準化を実現するための開発プロセスとアーキテクチャモデルを提案する．アーキテクチャモデルでは可変性の構造分析に基づく可変性のモジュール化を提案する.

本研究では，ポートフォリオマネジメントを指向した APLE の開発方法論を提供することで，各分野の SPL のコンテキストへの適用を可能とし，SPLE を運用する組織に対して包括的に管理可能な開発活動の推進への貢献を期待する.

1.3 本論文の構成

本論文の構成を以下に示す.

第 2 章は研究課題について説明する．第 3 章は関連研究について述べる．第 4 章はアプローチを示す．第 5 章は SPLE のためのアジャイル開発方法のフレームワークを示す．第 6 章は可変性のモデル化と構造分析，第 7 章は SPL のアプリケーション開発の並行開発アーキテクチャについて説明する. 第 8 章は提案する開発方法を適用した自動車システムのソフトウェア開発について示す．第 9 章は提案する開発方法論を適用した開発の評価結果である．第 10 章は提案する開発方法論についての考察を述べる．第 11 章は今後の課題，第 12 章は本研究のまとめを述べる.

2 研究課題

2.1 SPL の開発全体を包括した管理可能な開発方法論

本研究で対象とするソフトウェア開発のコンテキストでは，複雑な多様性を備えた SPL を取り扱い，製品進化と同時並行的に多様な製品提供を求められる俊敏性が期待される．このようなコンテキスト上の組織には，個別の製品進化や多様な製品開発だけでなく，SPL の開発全体を包括した管理可能な開発方法論が必要である．

SPLE におけるアプリケーション開発では，単一システムのソフトウェア開発の規模は比較的小さい[10][39]．しかし，SPL から生成する製品が増加すると，個々の製品の納期が近接し，開発に割り当て可能な期間と必要な資源が短期間に集中する．加えて，俊敏な製品進化への要求によって，単一の SPL では要求の多様性を吸収できなくなり，複数の SPL の並行開発が必要となる[3]．

SPL の管理可能な開発を実現するためには，SPLE のドメイン開発やアプリケーション開発を，個々の開発プロジェクトとして管理するだけでは十分でない．また，単一の SPL を開発管理するだけでも不十分である．なぜならば，組織が開発に割り当てられる資源には限りがあり，それぞれの開発期間の重複は資源の制約下に置かれることで相互依存を生じるためである．SPL の管理可能な開発を実現するためには，複数の SPL を包括した管理方法の確立が必要である．

本研究では，製品進化と多様な製品提供を同時並行的に実現しながら，SPL の開発全体を包括した管理可能な開発を実現する開発方法論を提案する．図 2.1 に示すように SPL の開発全体を包括した管理可能な開発方法論の中核を成すのは，包括した管理を支える開発プロセスとプロダクト可変性，アーキテクチャである．それぞれの研究課題について以下に説明する．

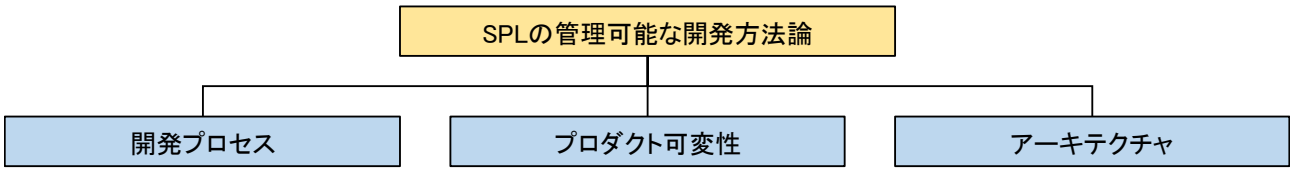

図 2.1 開発方法論の研究課題

2.2 開発プロセス

SPLE の開発プロセスは，単一の SPL をスコープとして，SPL における共通性と可変性を分析してコア資産と個別製品を開発するアーキテクチャを実現することが主体となっている[10][39]．そのため，開発を管理するためのマネジメントは，単一のプロジェクトマネジメントの方法に従うことが一般的である．

SPL の包括管理を実現するためには，従来のアーキテクチャ指向の開発プロセスではなく，マネジメント指向の観点を追加した開発プロセスの再設計が必要となる．マネジメントを指向した場合，SPL を包括管理するためには，管理指標を計画，監視，コントロールすることが可能となる開発プロセスの構造を備えることが必要となる．また，管理指標が存在するだけでは管理可能な開発は実現できない．管理指標の管理性が高い，すなわち管理し易い開発プロセスの構造を備えていることも必要である．

本研究では，SPL の包括管理を実現するために，計画，監視，コントロールするための管理指標を提供でき，かつ，同指標の管理性の高い開発プロセスの設計を課題とする．

2.3 プロダクト可変性

SPLE において，SPL における共通性と可変性の分析は，製品開発を成功させるための重要な要因となっている．特に，アプリケーション開発においては，開発コストを決定づける主要因となっている[39]．

分析された可変性は SPL において，アプリケーション開発ごとに変更する可能性のある要素である可変点（Variation Point）と，可変点の中の選択肢である変異体（Variant）として設計される[10][39]．これらの可変性は開発プロセスの実行制約にもなり得る．例えば，自動車ソフトウェア開発では，可変点と変異体の組合せは膨大であり，かつ依存性が高いことが知られている[51]．このような状況において，可変性の依存関係が不明であると，可変点を独立して開発することが難しく，可変点に対する実装とテストを一括して実施する必要が生じる．その結果，開発プロセスに対する制約が大きくなり，管理性の向上が損なわれるリスクが生じる．

本研究では，SPL の包括管理を実現するために，開発プロセスに対する制約を軽減し，開発プロセスの自由度を向上する可変性の構造分析方法の確立を課題とする．

2.4 アーキテクチャ

SPLE では，SPL における共通性と可変性を分析することで，可変性を可変点と変異体の組合せに分解して参照アーキテクチャとして設計し，アプリケーション開発時に可変点を構造化することでアプリケーションアーキテクチャを決定する[39]．

参照アーキテクチャはドメイン開発においてコア資産上に定義され，アプリケーションアーキテクチャはアプリケーション開発において個別のアプリケーションとして参照アーキテクチャ上に展開される．従来の SPLE では SPL の開発をドメイン開発の後にアプリケーション開発を順序づけている．しかし，俊敏な製品進化と俊敏な製品展開が求められるコンテキストでは，開発順序の制約が大きいと，開発の効率性が低下する．いくつかのアプリケーション開発とドメイン開発を同時に設計して実装する方法も考案されている[39]が，設計制約の束縛が大きい．

本研究では，SPL の包括管理を実現するために，ドメイン開発とアプリケーション開発の依存性を低減し，並行開発が可能となる SPL のアーキテクチャ設計方法の確立を課題とする．

3 関連研究

本研究の関連研究として，複数プロジェクトの包括管理についての研究がある．また，SPL を対象とした開発プロセス，可変性分析モデル，アーキテクチャ設計の研究について述べる．最後に関連研究に基づく本研究の位置づけを述べる．

3.1 複数プロジェクトの包括管理

3.1.1 BAPO モデル

SPLE の評価フレームワークとして BAPO モデルが提案されている[20][31]．BAPO は，SPL の開発に対する次の４つの関心事の頭文字を並べた略語である．

(1) Business：SPL から利益を得る戦略
(2) Architecture：ソフトウェアを構築する技術的な手段
(3) Process：ソフトウェア開発におけるロールや責務，関係
(4) Organization：ロールや責務の組織構造への割り当て

図 3.1 に BAPO のモデルを示す．各要素は相互に依存しているが，矢印で接続された関心事は，影響を強く与える依存関係とその向きを示している．これらの関心事の状態を評価することで，SPL の開発の適正を測ることができる．

BAPO モデルでは，Business の関心事がその他の関心事に対して最も影響力が強い．Business としての評価が最も高いとされるのは，SPL がビジネスゴールに向けて戦略的に資産を活用できている状態である．すなわち，投資対効果を最大化するように，ビジネス上の市場投入時期や利益の期待値とコストモデルが予測可能であり，製品ロードマップ上にフィードバックできる状態が実現できていることを指す．

ビジネスゴールを定義し，組織の保有する資産がどのように運用されているかを監視し，計画にフィードバックする仕組みが，SPL の包括管理に必要である．

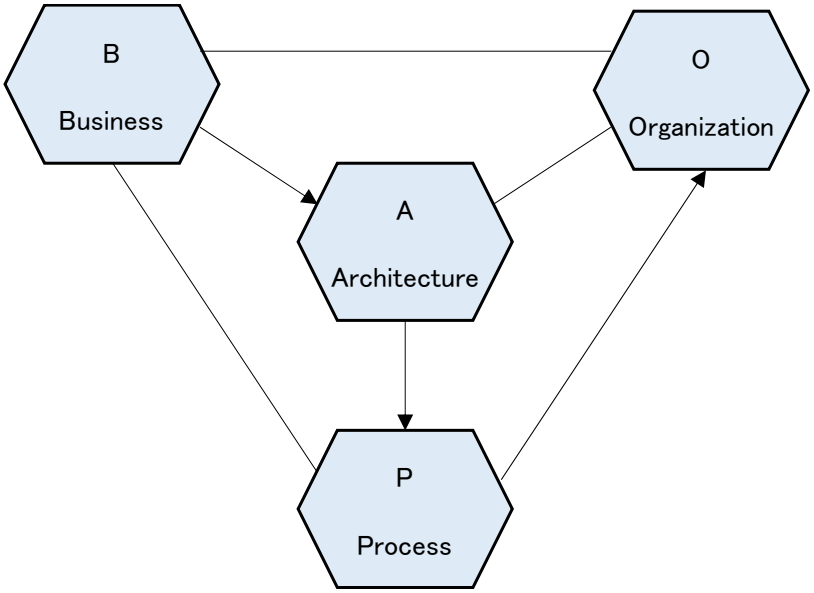

図 3.1 BAPO モデル

3.1.2 製品ポートフォリオマネジメント（PPM）

製品やプロジェクトを包括したポートフォリオは，個別のプロジェクトではなくビジネスの短期的かつ長期的な継続を確保するためのコンセプトである[15]．Ebert は，「PPM（製品ポートフォリオマネジメント）はビジネスで最大の価値を生むために顧客や市場に製品（あるいはソリューションやサービス）を投入するかの決定を統治する規律と役割である」と述べている[13][15]．また，Bekkers は，PPM は 4 つのビジネス上の機能のひとつとして識別されてきたものであり，それはソフトウェア製品管理である，としている[6][15]．残りの 3 つのビジネス上の機能としては製品計画，リリース計画，要求管理を挙げている．

Jagroep は，ポートフォリオ実装フレームワークを提案する中で，コアプロセスとして製品ライフサイクルマネジメントと製品ポートフォリオマネジメント，プロジェクトポートフォリオマネジメントの 3 つのマネジメントプロセスを提示している[15]．これらのコアプロセスの上位には，戦略プロセスを配置している．PPM は，戦略プロセスと製品ライフサイクルマネジメントで決定された製品戦略をドライバとして実行するプロセスである．

製品ポートフォリオマネジメントに先立って立案される製品戦略を，製品ポートフォリオ分析によって決定する方法が提案されている[20][39]．製品ポートフォリオ分析では，図 3.2 に示すように製品の市場シェアと市場の成長度合いの 2 軸で市場領域を 4 象限に分け，領域毎に製品を投入するための投資を分析する．市場シェアが低く，市場が成熟して成長が鈍化している Poor Dogs の領域を狙うのではなく，市場シェアが高く市場が未成熟で成長が期待できる Stars の領域を守り，市場が成熟したところで Cash Cows の領域を確保するといった製品戦略の立案をサポートする．

製品ポートフォリオのための製品戦略を立案する具体的な方法は提案されているが，製品ポートフォリオマネジメントを適切に実行するための具体的な方法は確立されていない．

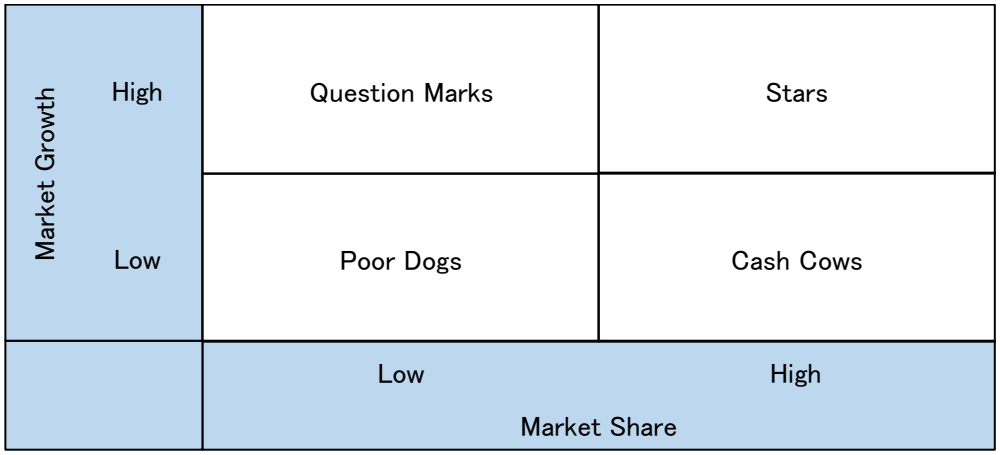

図 3.2 製品ポートフォリオ分析

3.1.3 アジャイルポートフォリオマネジメント

ポートフォリオマネジメントでは，資産と投資の運用を任意の視点で調整して適切に管理することが必要である．Krebs は，プロジェクトと資源，資産の３つのポートフォリオを ASD のフレームワークを利用して管理する方法を提案している[25]．3 つのポートフォリオの概要を以下に示す．

(1) プロジェクト：プロジェクトの投資対効果を測定し，監視するポートフォリオ
(2) 資源：人的資源の量と質（スキル）を測定し，監視するポートフォリオ
(3) 資産：開発済みの製品などの資産の保守や廃棄を測定し，監視するポートフォリオ

これらのポートフォリオを管理するために, Scrum などの ASD の開発フレームワーク[32][45]を利用する．これらのフレームワークは，プロジェクトの開発量と組織の生産性を普遍量で見積もるのではなく，2～3 か月の期間の実績から推定した組織の生産性を求め，計画を段階的に詳細化していくアプローチをとる．

Leffingwell はポートフォリオマネジメントチームを設置し, バリューストリームに対して資源を配置する方法を提案している[29]. 投資対効果の高い開発テーマに開発資源を割り当て, 監視とコントロールを実行する．ソフトウェアの開発チームが複数存在し, 開発が大規模化している場合には, 各チームの生産性を正規化する．大規模な開発において開発プロジェクトを俯瞰しながら投資対効果を高める管理方法として有効である[40][47].

これら ASD の開発フレームワークを利用した方法論は，柔軟なファイナンスモデルや測定の透明性などの ASD の規律に基づいている．

3.1.4 生産管理における工程能力の測定とコントロール

工場における生産管理では，工程能力を管理指標として定義している[19][24]．工程能力とは，工程が一定期間，統計的管理状態であるときの能力をいう．工程が偶発的に能力を発揮している状態は工程能力として認められない．測定対象がある一定のばらつきの範囲内において測定，コントロールされている状態が，生産現場が管理可能である状態である．

工程能力は任意の対象を定めて能力として測定する．例えば，測定対象を品質に定めた場合は QC（Quality Control）が可能となる[52]．一方，工程能力において量に着目した特性を工程許容量と呼ぶ．ソフトウェアの開発を，量に着目して管理可能な状態とするためには，測定期間としての一定期間の単位を規定し，ばらつきを測定できる量としての指標を規定する必要がある．測定したばらつきが，ある一定の範囲内に収まるようコントロールされていることが，開発が管理可能な状態といえる前提条件となる．

3.2 開発プロセス

3.2.1 SPLE

SPLE は，低コスト，高品質で SPL における多様な製品を開発するアプローチである[10][39]．図 3.3 に示すように，SPLE ではドメイン開発とアプリケーション開発の 2 つの開発領域に分けて要求の多様性に対応する．

ドメイン開発では SPL における共通性と可変性を分析してコア資産を構築する．共通性とは，SPL 内の製品間で同一の特徴点の集合である．可変性とは，SPL 内の製品間で差異が現れる特徴点の集合である．例えば，ある SPL で警報機能を常に備えて製品ごとに機能差がないのであれば、その SPL において警報機能には共通性がある．自動車のエンジンがガソリンかディーゼルかモータかで何らかの振る舞いを変えるのであれば，その SPL においてエンジンには可変性がある．コア資産は，製品を開発するためのソフトウェア，設計，テスト仕様，プロセスなど，開発に利用可能なあらゆる資産を指す．

アプリケーション開発では，コア資産から個別製品を開発する．アプリケーション開発の開発プロセスでは，要求開発，設計，実装，テストを順次実行する[39]．各プロセスは，コア資産で定義した可変点に着目して，プロセス，テスト仕様も含めて資産を再利用し，可変性を構成することで品質，期間，コストの最適化を図る[16]．しかし，アプリケーション開発における多様性に着目した，効率的なプロセス資産の運用は確立されていない．

SPLE におけるポートフォリオ管理の研究は少ない[43][44]．SPL として，どのような投資対効果のある製品ラインナップを揃えることが望ましいかを示唆するアプローチは提案されているが[39]，個々のアプリケーション開発を包括的に管理する方法は確立されていない．

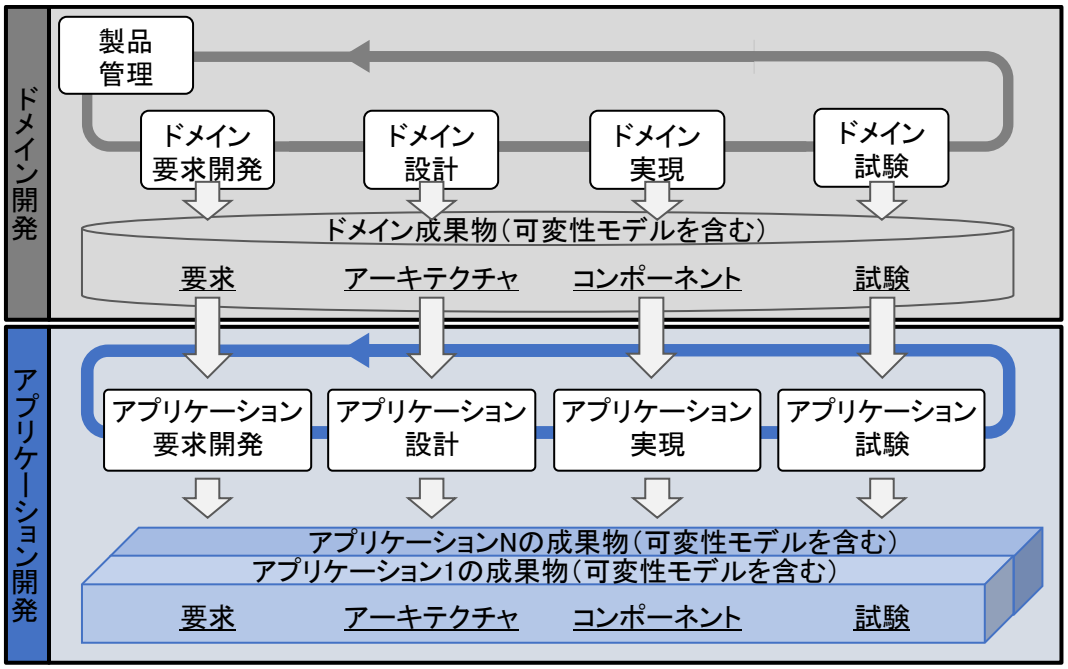

図 3.3 SPLE の開発モデル[39]

3.2.2 アジャイル開発（ASD）

ASD は，製品開発を短期間かつ細粒度で反復してインクリメンタルに開発することで，市場要求や環境の変化に俊敏に対応するアプローチである[5][8][29][32][45]．何を開発すればよいか，どのように開発すればよいかが明らかでないプロジェクトに対して，短期間かつ細粒度な開発でフィードバックによる学習を繰り返すことでゴールに近づく．

XP，Scrum に代表される ASD では，ユーザストーリと呼ぶ小規模な開発単位を対象に要求定義，実装，テストを反復して実行する．ユーザストーリは，ストーリ同士の開発規模を相対的に見積もったストーリポイントによって開発量の全体を把握する[5][32][45]．開発チームの生産性はタイムボックスと呼ばれる一定の反復期間を利用して測定される．タイムボックスの中で開発完了したストーリのリストを測定し，複数のタイムボックスでの平均消化ストーリポイントを算出することで生産性を算出する．イテレーション（タイムボックス内の一連の開発）を反復することで平準化される開発の生産性に基づき，計画にフィードバックして開発を管理する[8][29].

Leffingwell は大規模アジャイル開発のための SAFe（Scaled Agile Framework）フレームワークを提案している[29]. SAFe では大規模開発を複数のスクラムチーム[45]で協調して開発するためのフレームワークを提供している．Leffingwell や Turek は複数チームのプロジェクトを包括して管理するための方法として，ポートフォリオで決定した粗粒度の複数のバックログから，ストーリを抽出してチーム単位のバックログを再定義する方法を提案している[29][50].

しかし, 多くの ASD の方法論では, MPLE のように短期間で複数の開発が並行する場合は想定していない．学習とフィードバックに基づいたアプローチであるため，短期開発では開発チームが開発方法を学習する前に開発を終えてしまうためである．

3.2.3 APLE（Agile Product Line Engineering）

APLE は，SPLE と ASD を統合したアプローチである[11][18]．これらのアプローチでは，SPLE における SPL の多様性に対応する労力の低減と, ASD における俊敏な製品開発[8][18][29]の両方の利点を活用することを目指している．

Hanssen は，製品進化の段階に応じて SPLE と ASD の 2 つのアプローチを選択的に採用する方法を提案している[17]. また, Noor は, SPL の特徴点であるフィーチャの決定に顧客を参画させる方法を提案している[37]. Rumpe は，SPL において製品進化と並行しながらコア資産を運用する方法を提案している[42]．2 つのアプローチの統合方法は様々である．

SPLE のアプリケーション開発において，MPLE の開発を管理するために ASD と組み合わせた方法は確立されていない．

3.3 可変性分析モデル

SPLE では SPL における製品同士の共通性と可変性の分析を開発の基礎活動としている[39]．可変性には製品外部から観察できる外部可変性と，設計の詳細化で生じて外部からは観察できない内部可変性に分類できる[39]．

3.3.1 フィーチャモデル

フィーチャモデルは SPL における外部可変性をフィーチャの集合としてモデル化する[7]．フィーチャは製品の外部から観察でき，製品を特徴づける因子である．フィーチャモデルは，SPL を根とした木構造でフィーチャの関連性を表現する．

FAMA（FeAture Model Analyzer）はグラフとして図示可能なフィーチャモデルである[7][20]．図 3.4 に示すように，FAMA では製品を構成するフィーチャを，Mandatory（必須），Optional（選択的），Many-of（いくつか），One-of（内のひとつ）のいずれかで，上位のフィーチャから下位のフィーチャに対する関係として表現する．

木構造で表現したフィーチャモデルを利用して，フィーチャ分析する解析方法として FODA（Feature-Oriented Domain Analysis）や FORM（Feature Oriented Reuse Method）が提案されている[20][21][22][27][38]．外部可変性をモデル化することで，SPL において再利用性を検討しながら，フィーチャ同士の関係に矛盾や相関がないかを明らかにして設計することが可能となる．

外部可変性を対象とした分析方法は，製品計画や抽象度の高い設計プロセスでの利用が中心となる．また，SPL を根とした木構造で表現することから，解析対象が大きくなると取り扱う木構造も大きくなるため，詳細な解析には目的に応じた工夫が必要となる．

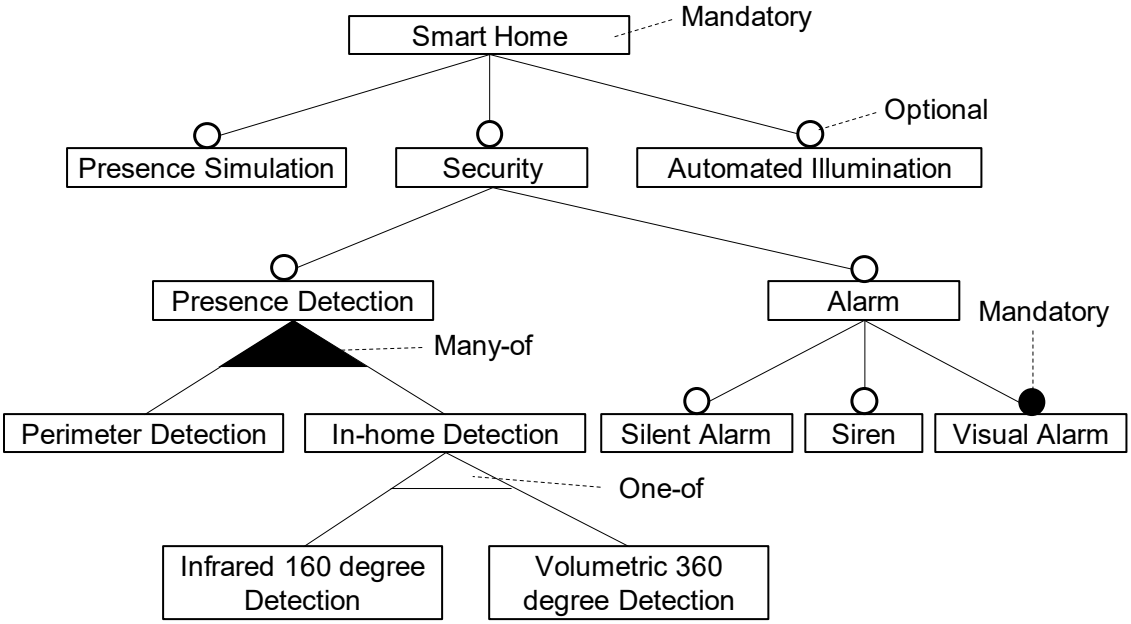

図 3.4 FAMA モデルの例[20]

3.3.2 OVM（Orthogonal Variability Model）

内部可変性を含む可変性モデルとして，OVM（Orthogonal Variability Model）が提案されている[8]．OVM はフィーチャモデルと異なり，共通性を省いて可変性に特化してモデル化する[34][39]．図 3.5 に，OVM の表記法とメタモデルを示す．OVM では可変性として，SPL に存在する可変点と変異体を識別し，可変点同士，変異体同士，可変点と変異体，の依存関係をモデル化する．可変点（Variation Point）は可変する特徴点であり，変異体（Variant）は可変点における選択肢である．例えば，3.2.1 で挙げた可変性の例では，エンジンが可変点であり，ガソリン，ディーゼル，モータが変異体となる．

製品開発プロセスで生成される成果物資産と可変点や変異体との依存関係も表現することが OVM では可能である．フィーチャモデルや UML モデルなど，その他のモデルと直交させることができるため，分析対象に応じて OVM モデルは様々な形に拡張して利用されている[2][14][34]．

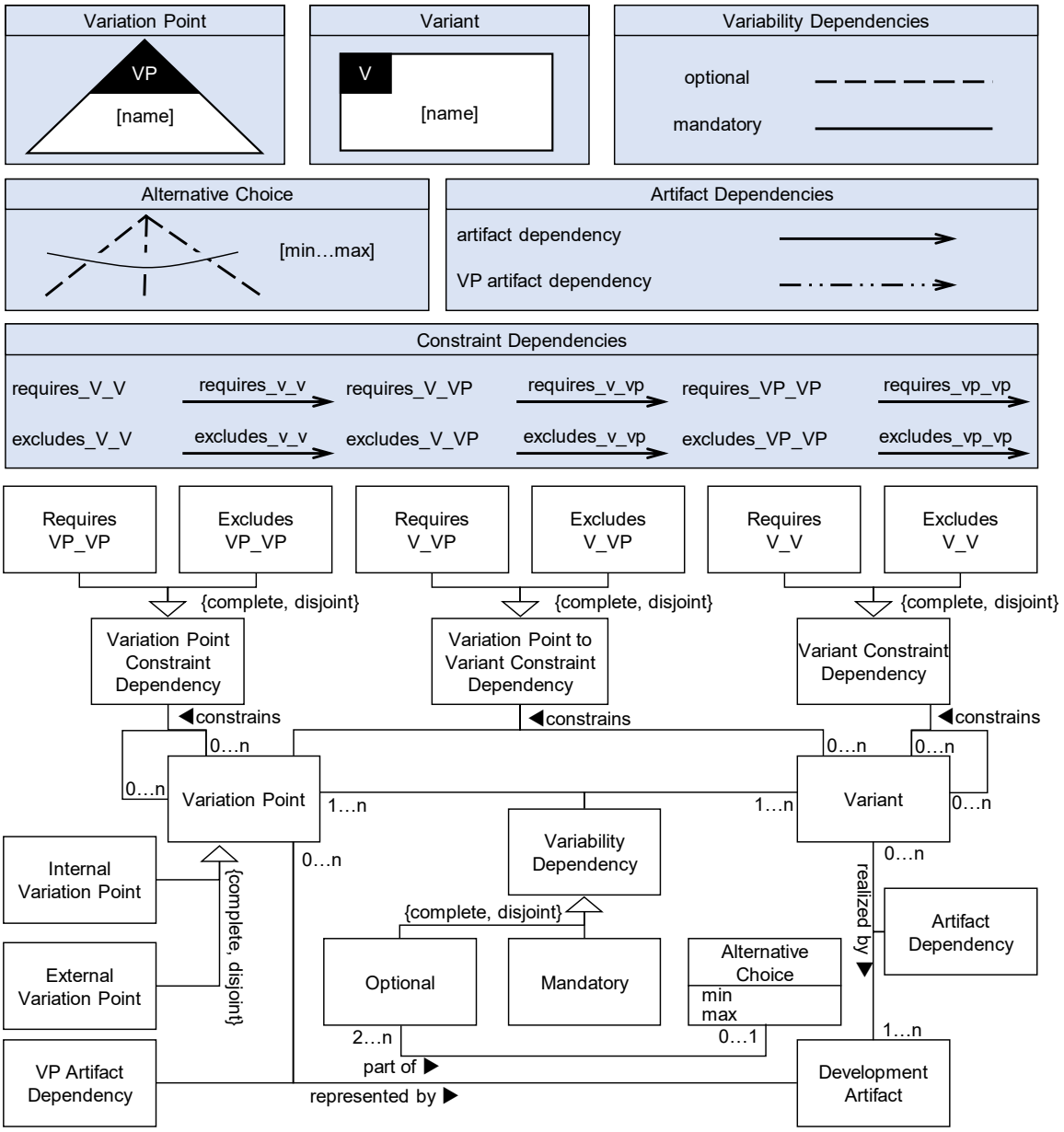

図 3.5 OVM 表記法とメタモデル[34][39]

3.4 プロダクトライン開発アーキテクチャ

3.4.1　アーキテクチャビュー

アーキテクチャビューはソフトウェアのアーキテクチャ設計で利用する技術である．アーキテクチャビューでは，ステークホルダの関心事に対応する複数のビューによってアーキテクチャを表現する[41].

Kruchten はアーキテクチャビューとして図 3.6 に示す 4+1 ビューを提案した[26]．4 つの標準ビューは，論理，プロセス，物理，開発ビューである．論理ビューはシステムの機能的側面を表現する．プロセスビューはシステムの同期性などの振る舞い的側面を表現する．物理ビューはシステムを構成する要素のハードウェアへの配置など物理的側面を表現する．開発ビューは開発環境におけるソフトウェアの構造的側面を表現する．1 つの拡張ビューは，システムのユースケースシナリオを分析するビューである．4+1 ビューを発展させて，様々なビューモデルが定義されている[33][41].

Rozanski はアーキテクチャビューとして 7 つのビューを提案している[41]．コンテキスト，機能，情報，並行性，開発，配置，運用ビューである．Rozanski はさらに，ビュー全体を横断する品質特性やシステム特有の性質を 10 のパースペクティブとして提案している．SPLE における可変性の設計は，進化パースペクティブとして，ビューを横断して設計するべき関心事であるといえる[4][34][39][41].

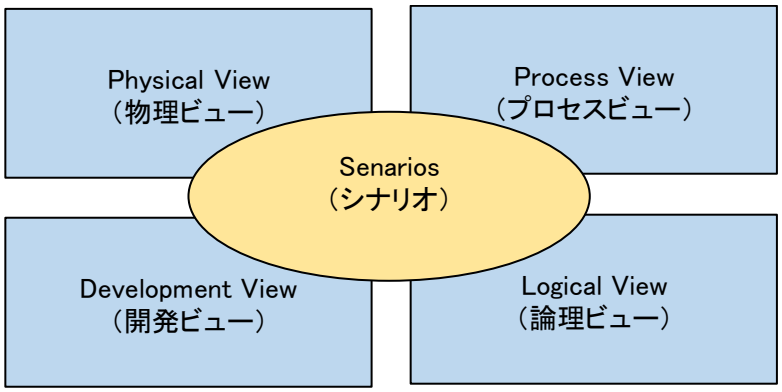

図 3.6 4+1 ビューモデル

3.4.2 Architectural Runway

アーキテクチャ設計はプロジェクト，あるいは製品ライフサイクルを通して，システムの開発に密接に関係する[41]．小規模で低リスクのプロジェクト，大規模なプロジェクト，計画駆動のプロジェクト，アジャイルプロジェクトなど，プロジェクトの性質によってアーキテクチャ設計プロセスを適切に設計する必要がある．

Leffingwell は, 大規模アジャイル開発のための SAFe フレームワークの中で, AR (Architectural Runway) を提案している[28][29][30]．AR は，短期間かつ細粒度で反復してインクリメンタルに開発する ASD において，リファクタリングの頻度を抑え，現在および今後予期される要求を組み込めるための既存の，または計画されたインフラであると定義している．図 3.7 に，AR の開発と利用を時系列に表現した概念図を示す．顧客の要求を満たすことに集中したフィーチャチームとは別に，システムのアーキテクチャを改善，あるいは拡張するための AR 開発チームを組織する．そして，フィーチャチームの開発を阻害しないように，インクリメンタル開発の中で継続的にアーキテクチャの改造を推進する．

アーキテクチャの進化を ASD のフレームワークに則って実行する場合, AR に基づいた組織とプロセスの連携が応用可能である．

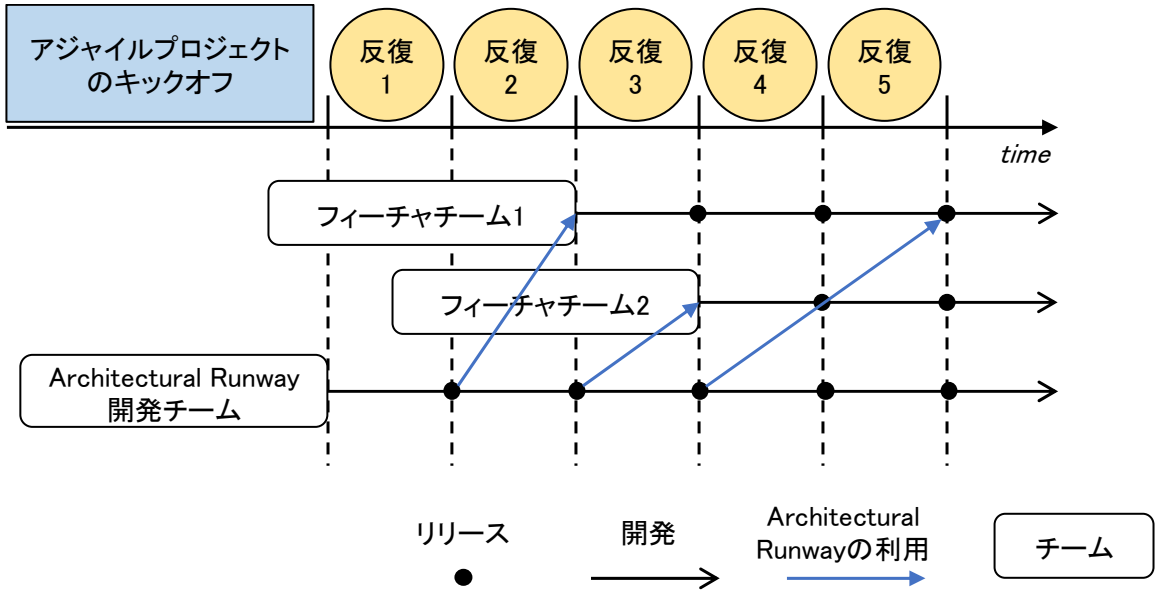

図 3.7 Architectual Runway の開発と利用

3.5 関連研究における本研究の位置づけ

3.5.1 複数プロジェクトの包括管理

本研究の位置づけはSPLEにおけるPPM(3.1.2)とアジャイルポートフォリオマネジメントの方法論(3.1.3)の融合である．PPM では管理フレームワークは定義されているが，管理性を向上させるための具体的な方法論が提示されておらず，製品ポートフォリオに基づいた戦略実行結果の達成是非の確認に留まる．アジャイルポートフォリオマネジメントの方法論では管理性を向上させる具体的な方法論を提示しているが，インクリメンタル開発を前提とした開発方法論であり，アーキテクチャ中心の SPLE への適用方法は確立されていない．

PPM のフレームワークにアジャイルポートフォリオマネジメントの方法論を SPLE 上で融合する方法を提示することで，複数 SPL の包括管理を管理性の向上と共に実現することが期待できる．

3.5.2 開発プロセス

本研究の位置づけは，SPLE（3.2.1）と ASD（3.2.2）を統合した APLE（3.2.3）としての，SPLE におけるアプリケーション開発への ASD の導入である．SPLE はアーキテクチャ中心の開発方法であるが，開発管理が容易となるプロセス構造を形成していない. ASD は変化への俊敏な対応を目指して反復開発することで管理性を向上するが，短期開発では効果が限定的である．APLE は 2 つの方法論を様々な形で統合しているが，ポートフォリオマネジメントの管理性向上を目指したプロセス構造に着目した統合方法は確立されていない．

SPLE のアプリケーション開発で数多くの製品を束ねて包括的に開発管理の対象とすることで，個々の開発は短期間であっても複数開発を包括した長期な開発と見做すことが可能となる．開発プロジェクトが切り替わっても学習によるフィードバックを継続させるプロセス構造を構築することで，管理性を向上する開発フレームワークを確立することが期待できる．

3.5.3 可変性分析モデル

本研究の位置づけは OVM（3.3.2）を応用した可変点の開発順序制約の分析方法の提案である．フィーチャモデル（3.3.1）では可変性と共通性の 2 つの性質を分析対象としているため，可変性のみを対象とした分析には冗長なコストが掛かる．OVM では可変性に特化することで効率的な分析が可能となるが，規定の依存関係では開発順序制約を適切に分析するための表現方法に不足がみられる．

OVM を応用して OVM 拡張モデルで開発順序制約の分析に必要表現方法を補完することで，可変点同士の開発依存構造を明らかにして独立した可変点開発の集合に分類することが可能となる．独立した開発の集合の分類は，反復的なタイムボックスによる ASD の開発を支援し，開発環境やテスト工数の負荷分散を果たして管理性の向上に寄与することが期待できる．

3.5.4 プロダクトライン開発アーキテクチャ

本研究の位置づけは，アーキテクチャビュー（3.4.1）の論理ビューと開発ビューを利用した可変性のモジュール化方法の提案と，AR（3.4.2）を利用した可変性のリファクタリング方法の実現である．アーキテクチャビューではアーキテクチャ設計の観点を示しているが，具体的な設計方法はコンテキストに依存するため明確には示されない．ドメイン開発とアプリケーション開発を同時並行で実行するコンテキストを対象とした開発方法も確立されていない.

開発ビューに着目して可変性をモジュール化してドメイン開発とアプリケーション開発の並行性を確立することで，ASD の開発フレームワークにおける AR を実現することが可能となる．開発並行性の確保は開発リソースの配置の自由度を高め，管理性の向上に寄与することが期待できる．

4 アプローチ

4.1 アプローチの全体像

4.1.1 アプローチの概要

多数の顧客の多様な要求を SPL として包括管理の下に開発するための本研究のアプローチを図 4.1 に示す．本アプローチは，ポートフォリオドリブンアジャイルマネジメント，2 層イテレーションプロセス構造，開発順序制約の解析，可変性のモジュール化の 4 つのアプローチから構成される．

(1) ポートフォリオドリブンアジャイルマネジメント：ASD の透明性により管理可能な開発を実現する．
(2) 2 層イテレーションプロセス構造：ASD を適用可能なプロセス構造にし，管理性を向上する．
(3) 開発順序制約の解析：可変性による制約を軽減し，開発プロセスの負荷を平準化し自由度を向上する．
(4) 可変性のモジュール化:開発の依存性を軽減して並行開発を支援し,開発資源の割当て制約を軽減する．

各アプローチは SPL におけるビジネス領域とテクノロジー領域の各問題領域に対応する．ビジネス領域は SPL の製品戦略とポートフォリオマネジメントの領域である．テクノロジー領域は SPLE のプロジェクトを実行する領域である．ビジネス領域の区分けは PPM（製品ポートフォリオマネジメント）の概念に基づく[13][15]．製品個別のプロジェクト管理ではなく，製品を包括してビジネスの長期的な継続を確保するために概念が PPM である．PPM は製品ポートフォリオの分析と製品ライフサイクルの分析で立案された戦略に従い実行され，下位に個別のプロジェクトポートフォリオマネジメントを位置づけている[13][15][20][39]．

本研究でのアプローチでは SPL の包括管理可能な開発を実現するため，SPL のポートフォリオマネジメントの管理容易性を向上させるように，各問題領域にアプローチする．各アプローチについて次節から説明を述べる．なお，ポートフォリオマネジメントの起点となる製品戦略の立案方法については諸研究[15][20][39]を参照し，本研究では対象としない．

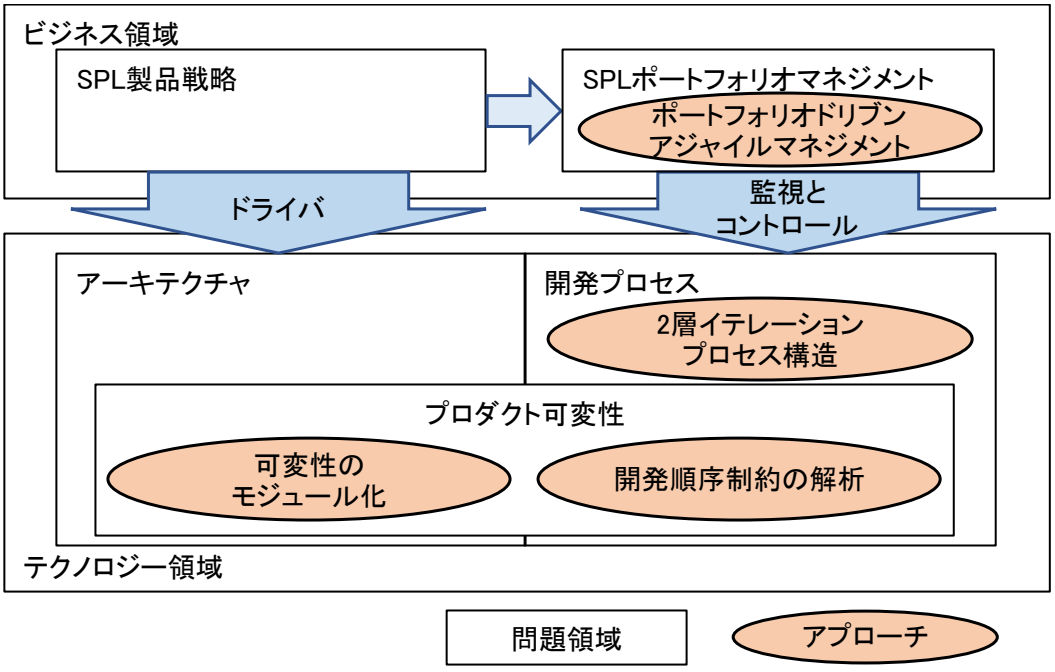

図 4.1 研究のアプローチの全体像

4.1.2　アプローチの着想

アプローチの着想として，ソフトウェア開発とハードウェア開発におけるプロダクトラインの開発管理の時間的なコスト比率が高い対象プロセスの相違点を示す．時間的なコスト比率の高い管理対象プロセスが異なることでソフトウェア開発では開発の包括管理へのアプローチが必要となることを提示する．包括管理へのアプローチとしてポートフォリオマネジメントにアプローチすることで，テクノロジー領域内の各領域へのアプローチが必要となることを論じる．

SPLE はハードウェア開発のプロダクトライン開発から着想したソフトウェア工場の考え方を発展させたアプローチである．ソフトウェア工場の考え方では計画的に個別に開発したソフトウェア部品を組み合わせて製品系列上の製品を数多く生み出すことで品質とコストを改善する．ハードウェア開発では部品の組合せは製造プロセスで同一製品の製造個体ごとに実行される．ソフトウェア開発では製造個体ごとには実行されない点が相違点となる．

SPLE では製造個体ではなく製品ごとに部品を組み合わせることで，開発コストと管理の時間的なコスト比率が製造プロセスではなく要求分析と設計プロセスに移る．ハードウェア開発では製造個体の組み立てに時間的なコストを必要とし，開発の全体プロセスに対して製造プロセスが占める時間的なコストの割合が大きい．製造プロセスが開発管理の時間的なコストの比率が高くなるため生産管理 (3.1.4) で管理性を確保する．一方，ソフトウェア開発では製造個体の組み立ては必要なく，製品としての要求分析と設計プロセスの時間的なコストが占める割合が大きい．ソフトウェア開発の管理性を高めるためには要求分析と設計プロセスの管理性を高める必要がある．

ハードウェア開発に対してソフトウェア開発の管理の時間的なコスト比率が高いプロセスが製造プロセスではないという相違点が，開発の管理領域に対するアプローチを必要とする．ハードウェアの製造プロセスでは製品系列の製品個体の組み立てラインを共有するため，製造ラインを管理対象とすることで製品系列の包括管理が実現できる．一方，SPLE の要求分析と設計プロセスは製品系列を対象にドメイン開発において包括的に扱われるものの，部品の組合せや追加変更も多くアプリケーション開発での個別実行が必要であり，製品ライフサイクル上に占める時間的なコストの割合が大きい．本研究では管理領域に対するアプローチとしてポートフォリオドリブンアジャイルポートフォリオマネジメントを提案する．

管理領域に対するアプローチはエンジニアリングにおけるアーキテクチャ，開発プロセス，プロダクト可変性へのアプローチを必要とする．管理領域としてのポートフォリオマネジメントへのアプローチは，ビジネス領域へのアプローチである．BAPO モデル（3.1.1）を参照するとビジネス領域への作用はアーキテクチャと開発プロセスへの作用を生じる．SPLE のアーキテクチャはプロダクト可変性への対応が中心である．これらの領域をビジネス領域に対するテクノロジー領域としてまとめる．図 4.2 に本研究におけるアプローチの観点を示す．

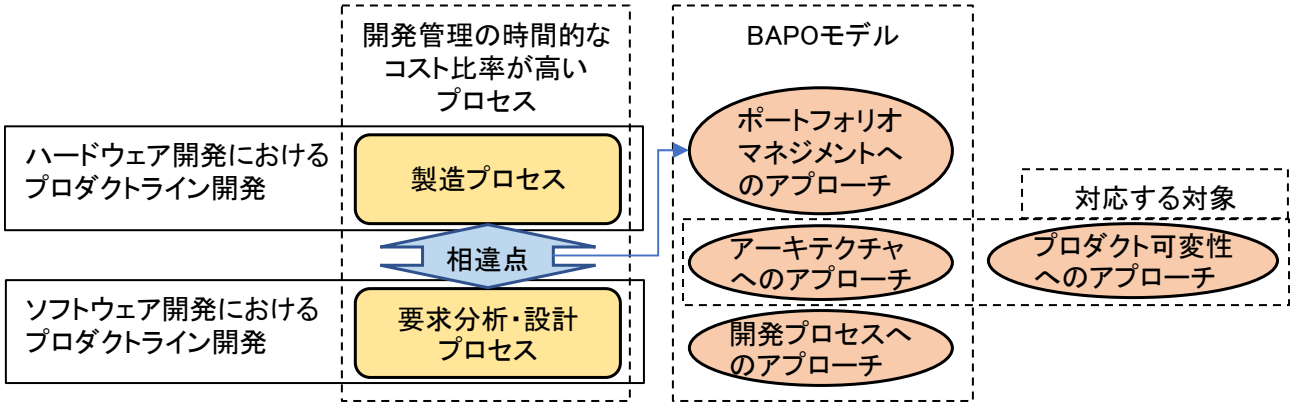

図 4.2 研究のアプローチの着想

4.2 ポートフォリオドリブンアジャイルマネジメント

多数の顧客の多様な要求を SPL として包括管理の下に開発するため，SPL に対するポートフォリオドリブンアジャイルマネジメントのアプローチを提案する．図 4.3 に，本研究で対象とする SPLE におけるポートフォリオドリブンマネジメント構造の概念図を示す．

ポートフォリオドリブンマネジメント構造は，ビジネス領域とテクノロジー領域に分割される．

(1) ビジネス領域

ビジネス領域は SPL 製品戦略と SPL ポートフォリオマネジメントの領域に分割される．SPL 製品戦略では，SPL としてどのようなポジションの市場に向けて開発するかや，開発した資産をどのように運用するかの戦略を立案する[20][39]．SPL ポートフォリオマネジメントでは，製品戦略に基づいてプロジェクト，資源，資産が正しく運用されているかを監視し，コントロールする[25]．

(2) テクノロジー領域

テクノロジー領域はアーキテクチャと開発プロセスの領域に分割される．アーキテクチャはビジネス領域のポートフォリオ戦略をドライバとして開発され，開発プロセスはアーキテクチャに適応する[20][31]．開発プロセスは SPL の各プロジェクトに展開実行され，ポートフォリオマネジメントの監視とコントロール対象となる．アーキテクチャに基づいてプロダクト可変性が定まり，開発プロセスにおける分析と管理の対象となる．

SPL のポートフォリオマネジメントを実現するためには，テクノロジー領域が監視とコントロールの面で管理性が高い状態であることが望ましい．本研究では，ポートフォリオドリブンとして，テクノロジー領域の各サブ領域において，管理性を向上させるためにアジャイルマネジメント方法に基づいたアプローチを提案する．これらのアプローチを統合して，SPL の包括管理可能な開発を実現する．

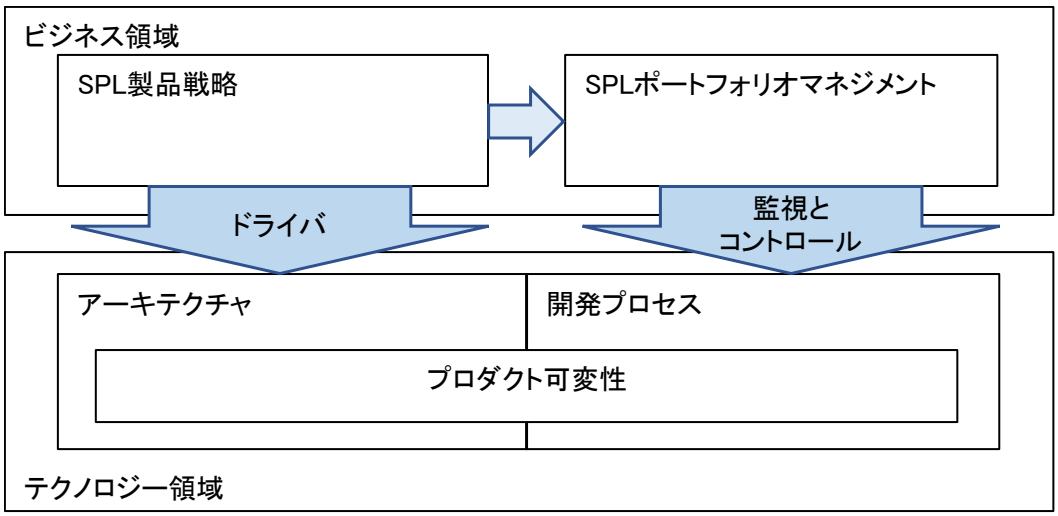

図 4.3 ポートフォリオドリブンマネジメント構造の概念図

4.3 2層イテレーションプロセス構造

4.3.1 短期アプリケーション開発へのアジャイルマネジメント方法適用の課題

アジャイルマネジメント方法は，反復的な開発プロセスによる学習とフィードバックに基づいて，開発の管理性を向上する[8]. Scrum や Kanban[1]などのよく知られた ASD ではタイムボックスを開発の反復単位とする．開発全体を通してタイムボックスごとに生産性を測定してコントロールし，計画にフィードバックする[8][29].

これらのアジャイルマネジメント方法は短期プロジェクトでは効果的に働かない．短期プロジェクトでは，開発チームがプロジェクトにおける開発プロセスを学習し，生産性を改善する前にプロジェクトが完了してしまうためである．SPLE における短期なアプリケーション開発にアジャイルマネジメント方法を導入するためには，学習とフィードバックが働く構造の構築が課題となる．

4.3.2 アジャイルアプリケーション開発のための 2 層イテレーションプロセス構造

SPLE におけるアプリケーション開発のプロセスは製品開発ごとに反復される．SPL の可変点はドメイン開発の可変性分析に基づいて前もって設計される．アプリケーション開発では，未実装の変異体の補完や可変点ごとの変異体の選択によって，コア資産から個別の製品を開発する．製品を開発する度に，同一の可変点に対する変異体を選択実装するプロセスは反復される. そのため, 製品ごとの可変点実装プロセスは反復性を示す.

図 4.4 に SPLE のアプリケーション開発においてアジャイル開発方法を適用する概念図を示す.

図 4.4 の上図は，従来のアジャイル開発プロジェクトにおける反復的なプロセス構造を示す.

本研究のアプローチである 2 層イテレーションプロセス構造の概念を図 4.4 の下図に示す．SPLE のアプリケーション開発は，Project A，B，C，D といった単一の反復的な開発プロセスの並びとみなすことができる．さらに，製品開発の各開発プロセスは，複数プロジェクトを跨いで反復的に実行され得る．そのため，本研究の ASD の反復的なプロセス構造は 2 層で実現する．単一プロジェクト内のマイナー反復ループと，複数プロジェクトを跨いだメジャー反復ループである．本研究では，SPLE のアプリケーション開発に 2 層のプロセス構造による ASD とマネジメント方法を導入することで，複数の製品開発を包括して管理性を向上する効果を期待する.

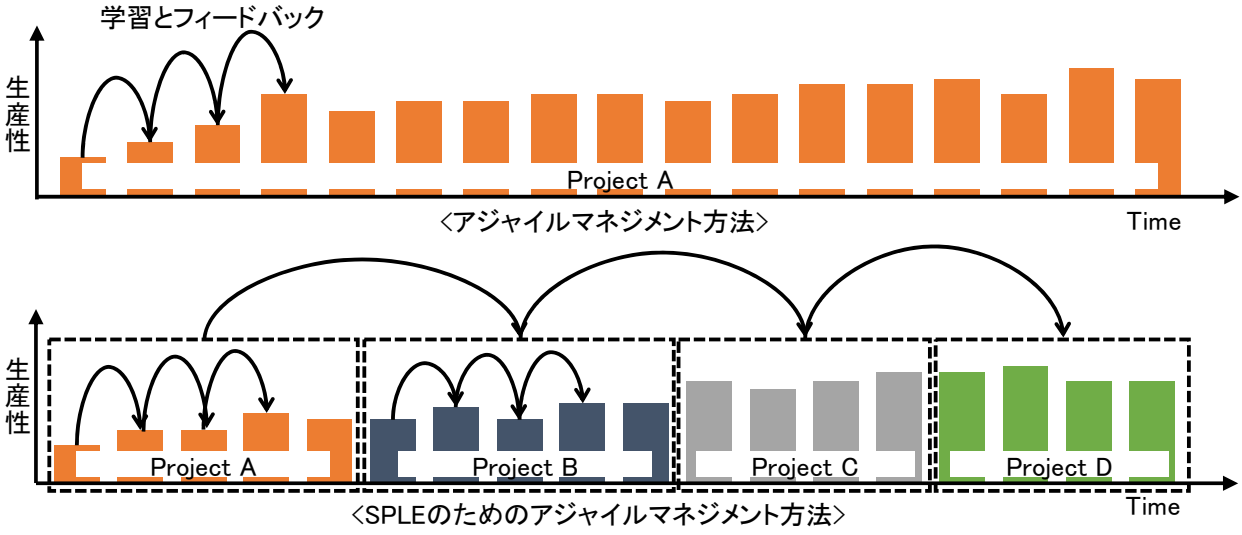

図 4.4 SPLE のアプリケーション開発に ASD を適用する概念図

4.4 可変性の構造分析による開発順序制約の解析

アプリケーションを ASD の開発フレームワークでインクリメンタルに開発するためには，開発要素（ストーリ）が INVEST の性質を満たすことが望ましい[8]. INVEST とは, Independent（独立している）, Negotiable（対話を引き出す），Valuable（ユーザ価値を提供する），Estimable（見積り可能である），Small（小さい），Testable（テスト可能である）の各単語の頭文字を並べた指針としての略語である．これらの内，Negotiable，Valuable，Estimable は新たな機能や可変点の開発では重要であるが，SPLE のアプリケーション開発の設計済みの可変点の変異体実装では，元々条件が満たされている．

INVEST の Independent，Small，Testable を満たすべく可変点の開発を分割することを考える．Independent については, 可変点の部分集合が, その他の可変点の集合に対して依存関係を持たなければよい．Small は定性的な尺度であるが，開発チームが定めた反復単位であるタイムボックス[8] [29]の期間に収まるか否かである．依存関係を外に持たない部分集合が Small を満たさない場合はさらに分割する必要がある．Testable は依存関係のある可変点のテスト可能性を表す．依存先の可変点が実装されていないと，依存する可変点は Testable でない．Testable であるためには，依存される可変点は，その他の可変点に先立って開発される必要がある．

インクリメンタルに開発するための Small な可変点の開発に分割できるように，可変性の構造を分析することで，Independent で Testable を満たす可変点の開発順序の制約を分析するアプローチを提案する．図 4.5 に，本研究のアプローチにおける，可変性の構造分析とアプリケーション開発モデルの関係を示す．

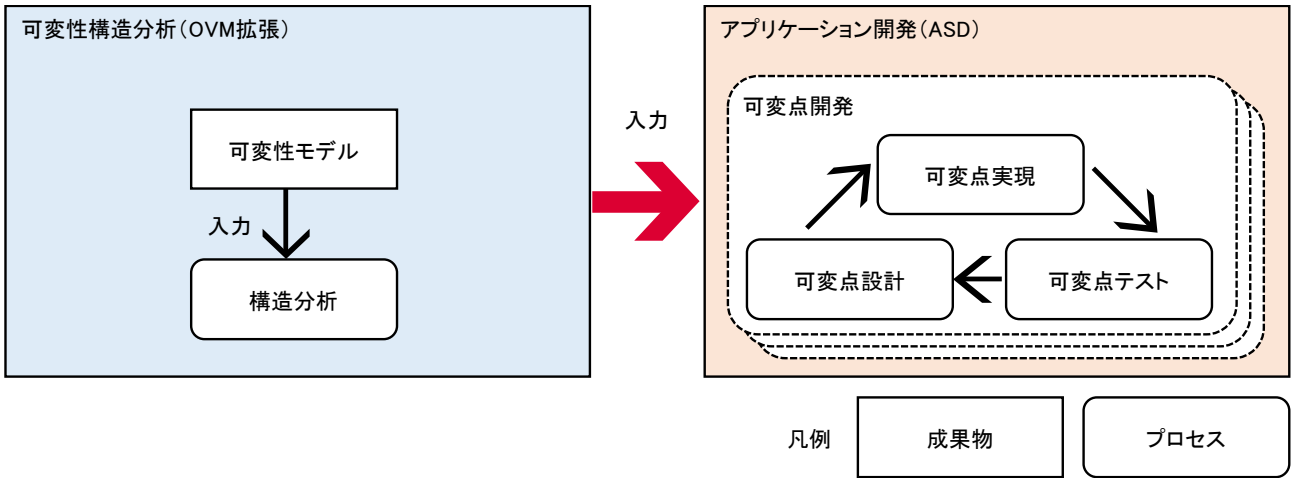

図 4.5 可変性構造分析とアプリケーション開発の関係

4.5 アーキテクチャにおける開発ビューポイント上の可変性のモジュール化

SPLE のドメイン開発とアプリケーション開発の同時並行開発を実現するために，アーキテクチャにおける開発ビュー上の可変性のモジュール化設計の導入を提案する．図 4.6 に，可変性のモジュール化の概念図を示す．

可変性のモジュール化は，アーキテクチャにおける論理ビューと開発ビューの２つのビューを利用して既存の SPLE のアーキテクチャをリファクタリングすることで実行する．各ビューは，Kruchten の 4+1 ビュー[26]に含まれる．

(1) 論理ビュー

システムの機能的構造を論理的に表現したビュー．コンポーネント図やクラス図などのモデルを利用して設計できる．

(2) 開発ビュー

モジュールやサブシステムなどのソフトウェア構造を表現したビュー．構成管理上のソースコードファイルの配置など，ソフトウェア開発に関連する関心事を取り扱う．

本研究における開発ビュー上での可変性のモジュール化とは，可変性を表す可変点や変異体を実装上のファイル構造としてモジュール化し，共通性を表すロジックから分離して開発上独立させることと定義する．図 4.6 では，初期のアーキテクチャではバンパ判別モジュールが開発ビュー上では単一モジュールとして設計されている．論理ビューでみたとき，バンパ判別モジュールは，可変点であり，変異体に A から C の判別方法を備えた可変性を有することがわかる．そのため，開発ビューをリファクタリングし，判別方法 A から C のロジック部，あるいはクラスをレイヤ構成，ならびにファイル構成としてモジュール化して分離している．

可変性がモジュール化され，開発ビュー上で独立していることで，ドメイン開発とアプリケーション開発が独立並行して開発可能となることが期待できる．図 4.6 では，更新後の開発ビューにおいて，ドメイン開発でバンパ判別部と判別方法 A を開発しているとする．アプリケーション開発で判別方法 B，C を開発実装する場合，任意のタイミングでドメイン開発と衝突せずに開発することが可能となる．

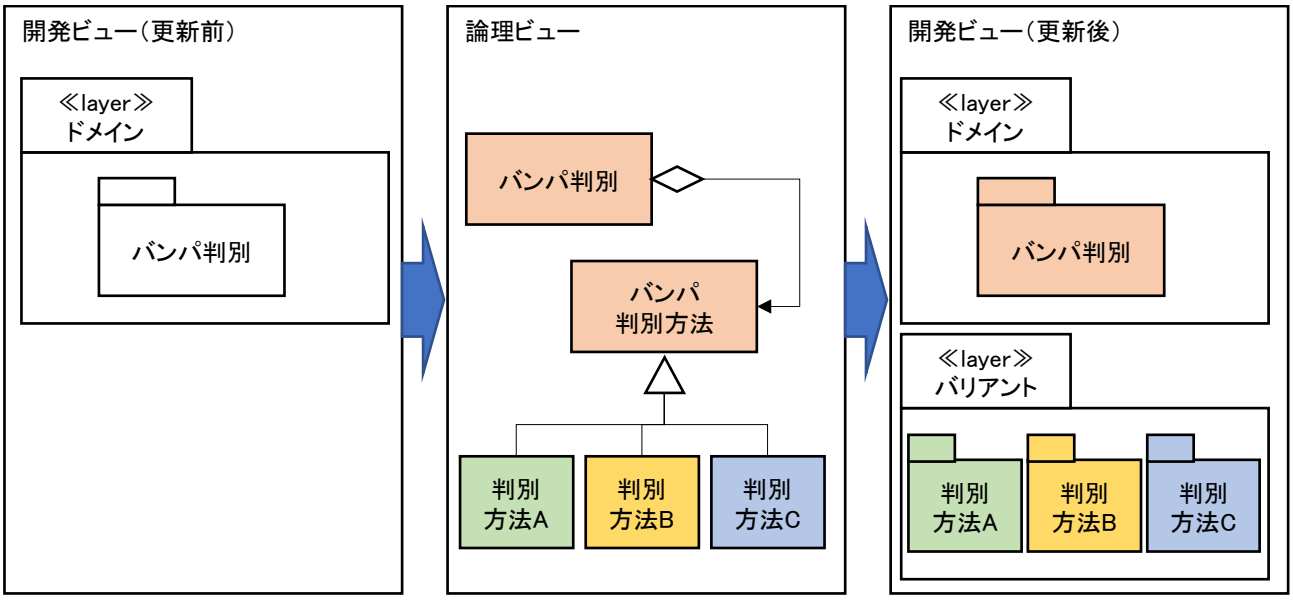

図 4.6 開発ビューポイント上の可変性のモジュール化の概念図

5 SPLE のためのアジャイル開発方法

2 層イテレーションプロセス構造で，ポートフォリオドリブンアジャイルマネジメントを実現する開発方法を提案する．2 層のイテレーションプロセスを機能させるための基礎であるプロセス資産の概念を示し，開発方法のモデルを提示する．

5.1 プロセス資産モデル

5.1.1 プロセス資産の概念

本研究では図 5.1 に示すようにアプリケーション開発のためのプロセス資産の概念を適用する．4.3.2 で述べたように，ある可変点（VPx）を実装するためのプロセスは，複数のアプリケーション（Application 1, … , Application N+1）の開発を跨いで反復的に再利用される．再利用可能なプロセスの集合がプロセス資産である．

あるアプリケーション開発から別のアプリケーション開発に一定以上の期間が空いた場合，学習とフィードバックによる安定化の効果が低下してしまう．アプリケーション開発が反復されたときに，生産性を安定させたい．学習効果の継続を狙い，本研究ではプロセスを資産として定義し再利用することを提案する．

本研究では，プロダクトラインの可変性を分類し，類似の可変性に対してはプロセスが反復可能であることに着目した．この結果，プロセス資産として蓄積するために，以下 3 つのプロセスパターンを特定した．

(1) プロセスパターン 1：アプリケーションを導出する際に，反復して同一の可変点（VP1）の変異体を決定するプロセス（P1）．

(2) プロセスパターン 2：対象とする可変点（VP2，VP 3）は異なるが，分析対象や試験方法が異なるだけで，類似の作業内容や作業規模であるプロセス群（P2，P2-1，P2-2）．

(3) プロセスパターン 3：コア資産が開発ライフサイクルの中で成長することで新たな可変点（VPx）が生じる．これに対応して，アプリケーション開発の反復の中で，新たに実行されるプロセス（Px）．

これらのプロセスは反復性を備えるため，プロセス資産として蓄積することで再利用可能となる．

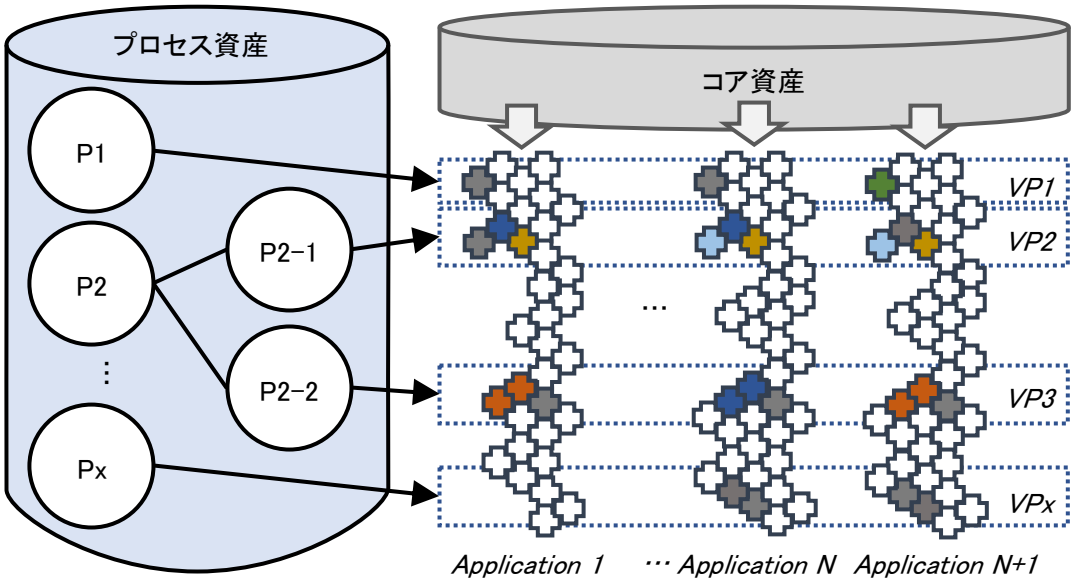

図 5.1 反復性のあるプロセス資産

5.1.2　プロセス資産のモデル

プロセス資産のモデルを図 5.2 に示す．プロセス資産は，1 つ以上のプロセスユニット，或いはプロセスユニットグループで構成される．プロセスユニットとプロセスユニットグループを定義する．

5.1.2.1　プロセスユニット

プロセスユニットは製品開発で反復実行されるプロセスである．プロセスユニットは，ユニット名，プロセス定義，プロセス成果物，見積り規模から構成される．以下に，各構成要素を示す．

(1)　ユニット名
プロセスユニットを特定するための名称である．他のプロセスユニット，プロセスユニットグループと重複しない．

(2)　プロセス定義
プロセスユニットを実行するための作業手順である．プロセス成果物で作業手順が示される場合は概要の記述でもよい．作業者がプロセスを反復して実行できる記述内容とする．

(3)　プロセス成果物
プロセスを実行したときの成果物の実体，あるいは成果物へのリンクを記録する．反復実行する度に追加して記録する．プロセス定義と組み合わせ，反復実行するときに参照する．

(4)　見積り規模
プロセスユニットの開発規模である．ASD のストーリポイント[5][32][45]と同様にポイントで表現する．計画時に過去の見積り結果として参照する．プロセスユニットを登録するときには実績がないため，計画時に他のプロセスユニットと比較して相対的に規模を見積る．

5.1.2.2　プロセスユニットグループ

プロセスユニットグループは，作業内容や作業規模が類似したプロセスユニットへの関連付けである．プロセスユニットグループは，分類名と 1 つ以上のプロセスユニットから構成される．分類名は，関連付けられたプロセスユニットの集合に抽象化した名前を割り当て，再利用を容易にするための名称である．

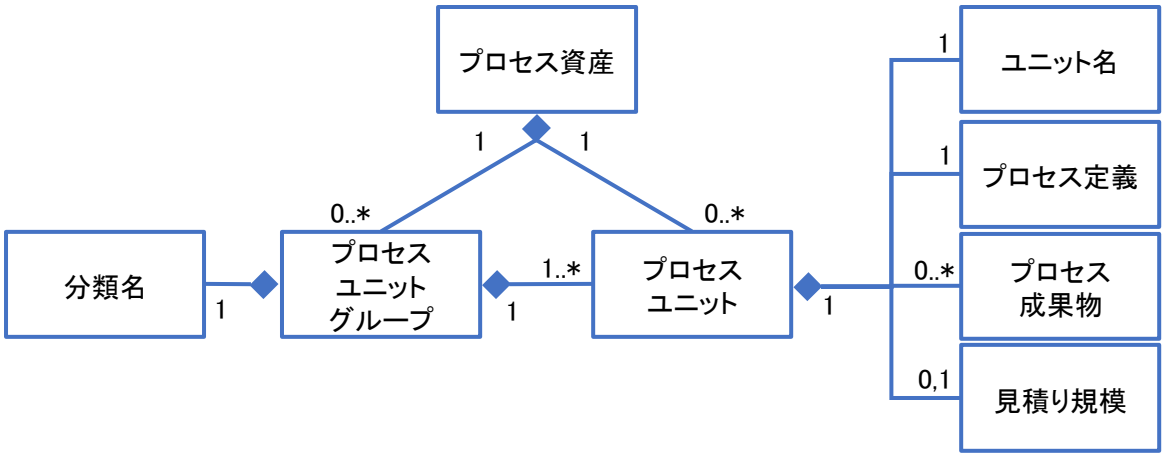

図 5.2 プロセス資産モデル

5.1.3　プロセス資産の例

表 5.1 にプロセスユニットの例を示す．5.1.2 で提案したプロセス資産の各要素が表中に示されている．“反復性”列は，各プロセスユニットがアプリケーション開発の中で反復されるか否かを示す．これらのプロセスユニットはプロセス資産として登録される．

プロセス資産において，5.1.1 のすべてのパターンのプロセスユニットが登録される．#3 と#7 のプロセスユニットはプロセスパターン 1 である．#11 のプロセスユニットはプロセスパターン 2 である．このプロセスユニットのプロセス定義の詳細はプロセスを実行する直前に記述される．#4 から#6 と#8 から#10 のプロセスユニットは，プロセスパターン 3 に分類される．これらのプロセスはグループを形成し，分類名が与えられる．

プロセスユニットの中の開発プロセスを分割する方針は開発モデルに依存する．我々の実践では，要求開発（#1），設計（#3，#4，#5，#6，#11），実現（#7），テスト（#2，#8，#9，#10）などのプロセス単位に分割している．従来の ASD では，ストーリは要求開発からテストまでのプロセスセットを通しで開発する．ASD は開発フレームワークとして適用し，開発プロセスは SPLE に従っている．このストーリの分割方針は，プロセスユニットの再利用を簡単にするための工夫である．

表 5.1 プロセス資産の例

#	分類名	ユニット名	プロセス定義	見積り規模	反復性
1	*(None)*	Node-to-node communication analysis	Analyze variation points in the communication specifications: 1. Compare with communication specification *S1* of the similar product. 2. Confirm specification *S2* of communication node with difference. 3. Confirm the specification *S3* of communication data. …	8	Yes
2	*(None)*	Load test on a special environment	Load testing on a special environment: 1. Load an execution object into the simulator *Sim1*. 2. Set high load condition *C1* on *Sim1*. 3. Try 10 times to calculate average and maximum CPU usage. …	5	No
3	*(None)*	Identification of variants	Identify the variant from a variation point list: 1. Compare product specification *S4* with variation point list *VL1*. 2. Determine variants at variation points. 2-1. At variation point *VP1*, variants are selected from *V1*, *V2*, *V3* with reference to specification *S5*. …	8	Yes
4	Add type to variants in communication data (Design)	Add engine type (Design)	Add the engine type to a set of variants in the communication data X: 1. The function *F1* in the module *M1* of the component *C2* is to be changed.	1	Yes
5		Add T/M type (Design)	Add the transmission type to a set of variants in the communication data Y: 1. The function *F1* in the module *M1* of the component *C2* is to be changed.	1	Yes
6		Add drive system type (Design)	Add the drive system type to a set of variants in the communication data Z: 1. The function *F1* in the module *M1* of the component *C2* is to be changed.	1	Yes
7	(None)	Implementation	Implementation: 1. Implement according to design specifications. 2. Eliminate warning of static analysis tool *TL1*. 3. Self-check using self-check sheet *CS1*. …	5	Yes
8	Add type to variants in communication data (Test)	Add engine type (Test)	Testing of #4: 1. Inspect that the engine type is determined at timing *T1*. …	1	Yes
9		Add T/M type (Test)	Testing of #5: 1. Inspect that the T/M type is determined at timing *T2*. …	1	Yes
10		Add drive system type (Test)	Testing of #6: 1. Add that the drive system type is determined at timing *T3*. …	1	Yes
11	(None)	Add steering type (Design)	Add the steering type as a variant point referring to the communication data W	3	No

5.2 SPLE のためのアジャイル開発モデル

5.2.1 開発モデル

図 5.3 に提案する反復型プロダクトライン開発モデルを示す．本開発モデルは，SPLE におけるアプリケーション開発を対象とする．アプリケーション開発は，プロダクト開発とプロダクトラインポートフォリオ管理から構成される．

プロダクト開発では個々のアプリケーションを開発する．プロセス資産定義，プロダクト計画，プロダクト構築の 3 つのアクティビティから構成される．ここで開発されるプロダクト 1…N は，SPLE におけるアプリケーション 1…N と同義である．開発領域としてのアプリケーション開発と，個々の製品開発であるアプリケーション開発を区別するため，後者をプロダクト開発と呼ぶ．

プロダクトラインポートフォリオ管理は，プロダクト開発を統括する．ポートフォリオ計画，ポートフォリオコントロールの 2 つのアクティビティから構成される．

アジャイル開発の方法はプロダクト構築のタイムボックス開発を中心にモデルに統合される．ドメイン開発ではアーキテクチャ中心の開発方法に則る．タイムボックスにより測定可能となる生産性を起点として，その他のアクティビティが連携することでアジャイルマネジメントに基づくポートフォリオ管理を実現するモデルである．

本開発モデルの各アクティビティを以下に示す．

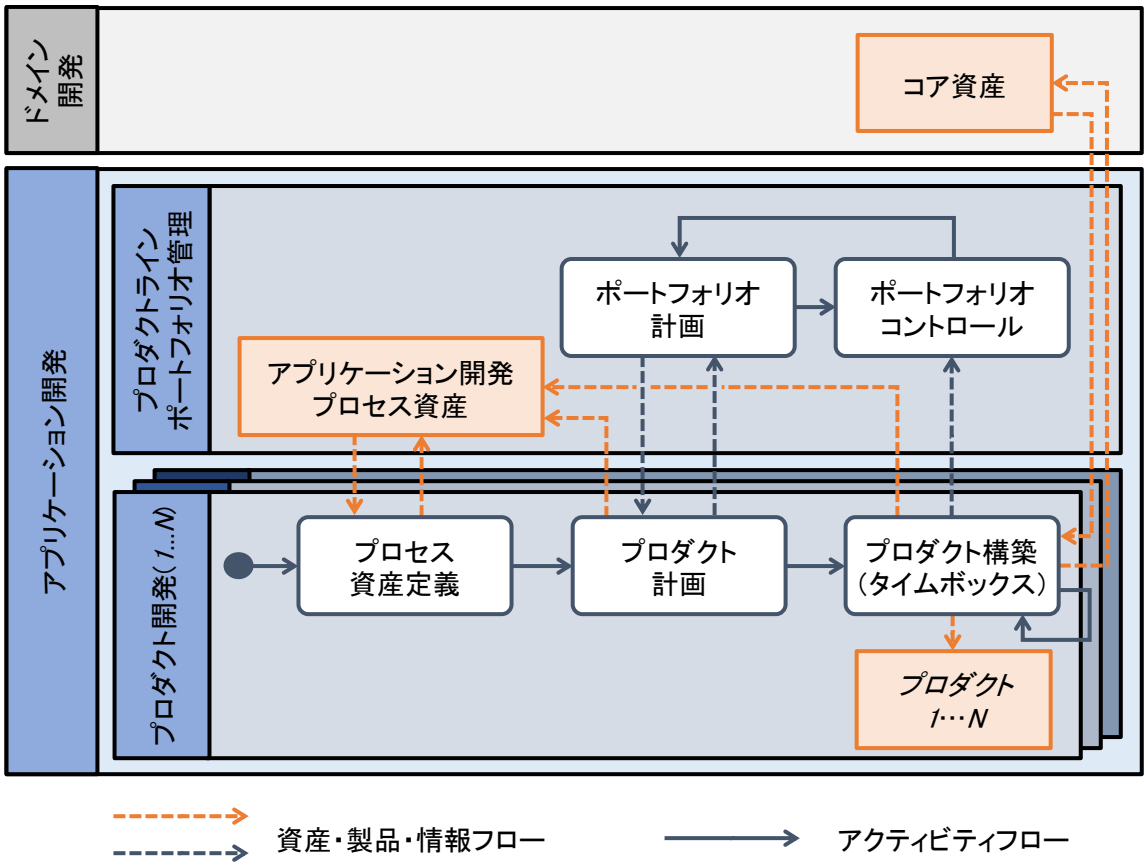

図 5.3 反復型プロダクトライン開発モデル

5.2.2 プロダクト開発

5.2.2.1 プロセス資産定義

プロセス資産定義アクティビティは，プロダクト開発の計画と同時に実行する．プロダクト開発の計画において，開発プロセスを設計するために，蓄積したプロセス資産を参照しながら，プロセス資産を登録更新する．本アクティビティは，以下（1）～（3）のタスクを実行する．

(1) プロセスユニットの新規登録

プロセスユニット，あるいはプロセスユニットグループを新規登録する．プロセスユニットは可変点を分析することで作成する．プロダクト開発で管理しやすい粒度であることが望ましく，表 5.1 に示したように，要求開発（#1），設計（#3，#4，#5，#6），実装（#7），テストなどのプロセス単位に分割する．

(2) プロセスユニットグループの関連付け

可変点に変異体を追加するプロセスユニットを定義するとき，同様の作業手順や作業規模となるプロセスユニットが存在する場合は，プロセスユニットグループを定義して関連付ける（#4，#5，#6，#8，#9，#10）．

(3) 反復未定プロセスユニットの登録

特定のプロダクト開発のための調査など，プロダクト開発ごとに反復される予定のないプロセスも設計される．これらのプロセスは，今後反復される可能性は低くても，その他のプロセスユニットの規模見積りや，将来反復された場合の参考作業実績として参照できるよう，プロセスユニットとして登録しておく（#2）．

5.2.2.2　プロダクト計画

プロダクト計画アクティビティでは，プロダクトバックログの作成とプロセスユニットの規模見積り，開発スケジュールの策定を実行する．図 5.4 にプロダクト計画のモデルを示す．

(1)　プロダクトバックログの作成

プロダクトバックログの作成では，プロダクト構築で実行するプロダクトバックログを作成する．プロダクトバックログは ASD のプロダクトバックログ[29][32][45]と同様な開発アイテムのリストである．本開発モデルでは，ストーリではなくプロセスユニットを用いてプロダクトバックログを作成する．6 章の可変性の分析により，可変点ごとに開発を独立して開発できる場合は，従来の ASD と同様にストーリを利用してプロダクトバックログは作成するよう変更できる．

(2)　プロセスユニットの規模見積り

プロセスユニットの規模見積りでは，プロセスユニットを反復実行するときに記録された見積り規模を採用する．初回登録時には，見積り規模が記録されていないため，他の記録済みのプロセスユニットと相対的に規模を見積もる．プロダクト開発自体が初めての場合は，ASD と同様にプランニングポーカー[9][45]などで規模を見積もる．

(3)　開発スケジュールの策定

プロダクトラインポートフォリオ計画では，当該プロダクトのプロダクトバックログの総ポイント数を基に，どのタイムボックスで何ポイントを当該プロダクト開発に割り当てるかを定める．プロダクト計画ではポートフォリオ上のタイムボックスごとの割当てに従って，スケジュールを策定する．

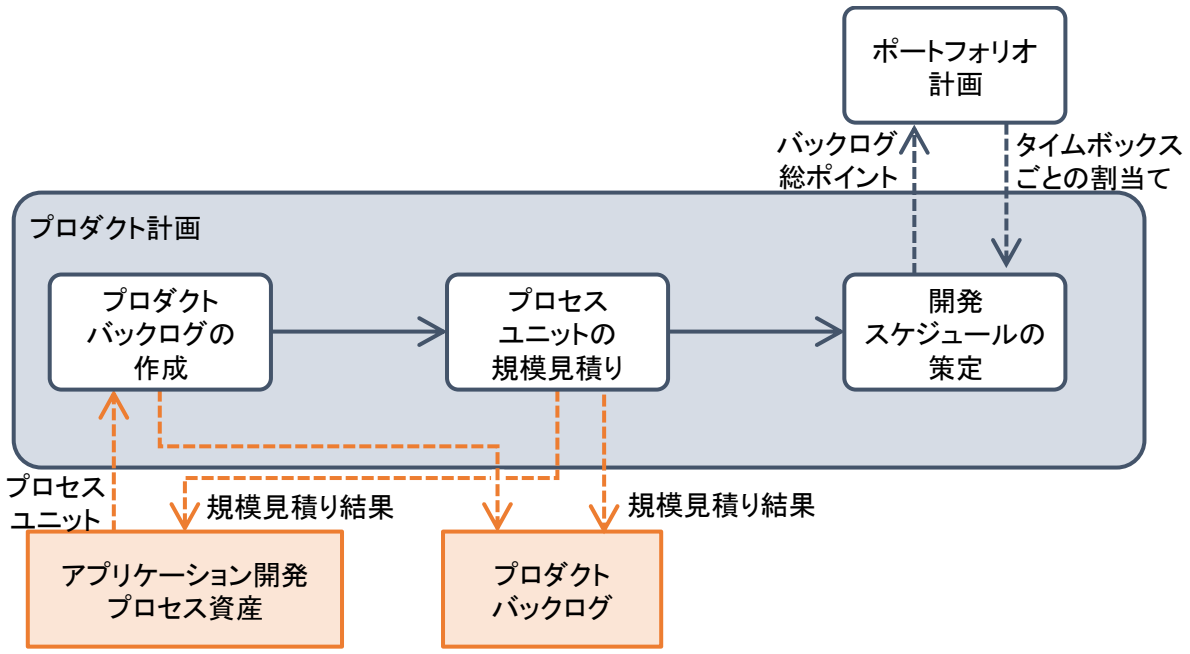

図 5.4 プロダクト計画のモデル

5.2.2.3　プロダクト構築

プロダクト構築アクティビティでは，スプリントバックログの作成とタイムボックスコントロールを実行する．図 5.5 にプロダクト構築のモデルを示す．

(1)　スプリントバックログの作成

個々のタイムボックスは ASD と同様にスプリントと呼ぶ[29][32][45]．プロダクト構築ではスプリントで取り組む開発アイテムをスプリントバックログとして作成する．

スプリントバックログは，異なる複数のプロダクトバックログを集約して作成する．ポートフォリオ計画において，今回スプリントでどのプロダクトバックログからどれだけのポイントを割り当てるかが定められている．定められたプロダクトバックログから，定められたポイントを満たすだけ，プロセスユニットを引き出して集約する．プロダクトの優先度もポートフォリオ計画に従う．

(2)　タイムボックスコントロール

タイムボックスコントロールでは，スプリントバックログに登録されたプロセスユニットを順に実行する．タイムボックスのコントロールは，Scrum の方法を適用する[9][45]．タイムボックスコントロールの目的は，プロセスユニットの見積り規模に対する作業実績のばらつきの吸収である．スプリントバックログを生成するときに，開発チームの稼働時間上限まで使わずバッファを設ける．バッファを設けることで，各スプリントで一定した量のプロセスユニットを完了するようコントロールする（図 5.6）．

タイムボックスで定められた期間（1～3 週間）の開発を完了すると，タイムボックスで完了したプロセスユニットの見積り規模の総和が定まる．見積り規模の総和はポートフォリオコントロールの入力とする．

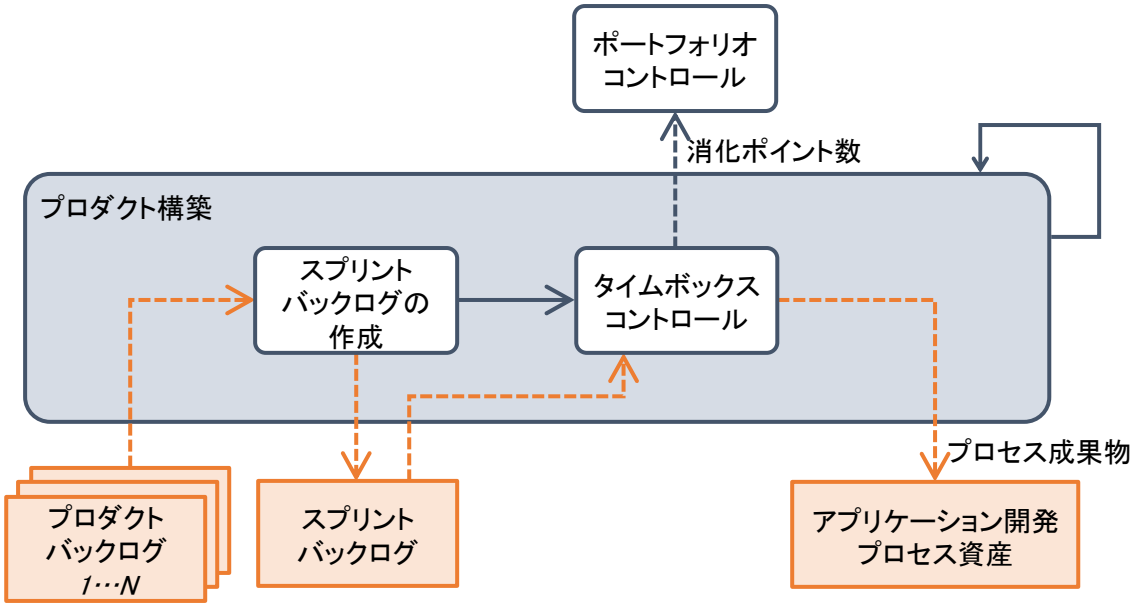

図 5.5 プロダクト構築のモデル

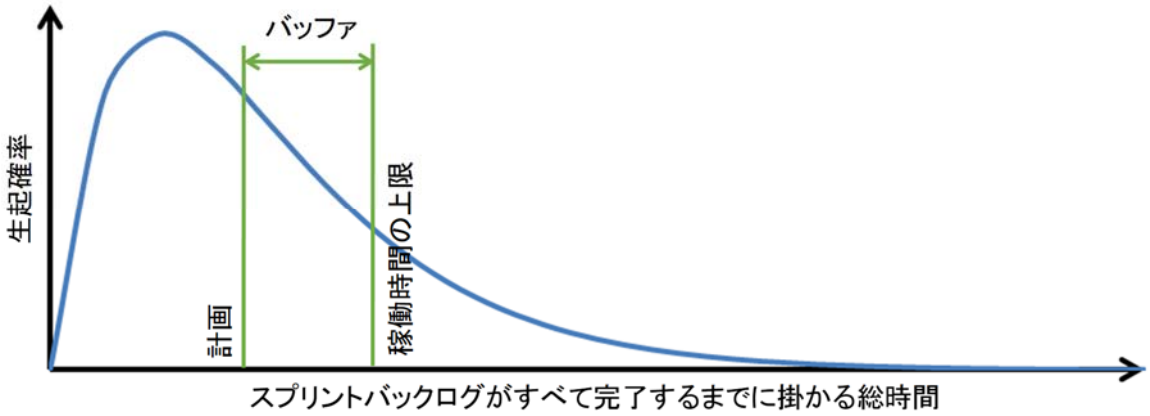

図 5.6 バッファによるばらつきの吸収

5.2.3　プロダクトラインポートフォリオ管理

プロダクトポートフォリオ管理は，すべてのプロダクト開発を統括して予算や資源を管理する．ソフトウェア開発の費用のほとんどは，人的資源に費やされることから，人的資源の管理を中心に述べる．以下に，ポートフォリオ計画，ポートフォリオコントロールの各アクティビティについて示す．

5.2.3.1　ポートフォリオ計画

ポートフォリオ計画アクティビティでは，いつ，何を，どれだけ開発するかを計画する．各プロダクト開発の規模見積りとマイルストーン，ポートフォリオコントロールから入力される生産性を入力情報とする．これらの入力を基に，この先のどのスプリントで，どのプロダクトのプロセスユニットを，何ポイント分実行するかを定める．

ポートフォリオ計画において，生産性を超えるスプリントを検出した場合，人的資源の追加投入，プロダクト計画の開発量低減やマイルストーンの交渉を検討する．

ポートフォリオ計画では，プロダクトの開発が要請された時点で，プロダクト開発を仮見積りする．プロダクト計画で詳細に見積もる前に，過去のプロダクト開発のポイント総和を利用して，中長期の計画を立てておく．

5.2.3.2　ポートフォリオコントロール

ポートフォリオコントロールアクティビティでは，プロダクト開発におけるタイムボックスコントロールで得られた消化ポイントから生産性を算出し，プロダクト開発を統括した進捗を監視し，改善活動を立案・実行する．

消化ポイントと生産性を併せて，ポートフォリオ計画に沿っているかの進捗を監視し，今後のポートフォリオ計画を見直す．生産性の推移としての変化が大きい場合，タイムボックスで実行できるプロセスユニットの量が変化しているため，計画の見直しが必要となる．

ポートフォリオコントロールでは，生産性が高く安定することを目指して，開発チームのプロセスを改善するための施策を検討する．目的は，生産性を高めることと，生産性を安定させて開発計画の見通しの正確性を高めることである．

6 可変性のモデル化と構造分析

SPLE において ASD を適用するためには，可変点の集合を独立した部分集合に分割する必要がある．そのため，本研究では，OVM モデルを拡張して，可変性の構造を分析し，可変性における可変点と変異体の組合せで生じる開発順序制約を分析する方法を提案する．

6.1 可変性構造分析のための OVM 拡張モデル

可変点の依存関係を整理し，依存関係と開発における分割方法を対応付ける．その結果として，新たな依存関係を導入し，新たな依存関係を表現可能とする OVM の拡張モデルと表記法を示す．

6.1.1 可変点の依存関係と分割方法

可変点の依存関係と分割方法を整理すると，3 つのパターンに分類できる（図 6.1）．2 つの可変点 vp1 と vp2 の依存関係を例として以下に示す．{*} は開発ストーリとしてまとめられる可変点の集合を示す．VPo は vp1, vp2 と依存関係のない，vp1，vp2 以外の可変点の集合を示す．→は依存関係による開発の順序制約を示す．

(1) vp1 と vp2 に依存関係が存在しない場合：
{vp1，vp2，VPo}は（{vp1, VPo}, {vp2}）または（{vp1}, {vp2, VPo}），（{vp1}, {vp2}, {VPo}）に分割できる．分割された集合同士は任意の順序で開発することができる．

(2) vp2 が vp1 に依存しており，vp1 は vp2 に依存していない場合：
{vp1，vp2，VPo}は（{vp1, VPo}→{vp2}）または（{vp1}→{vp2, VPo}），（{vp1}→{vp2}, {VPo}）に分割できる．vp2 を含む集合が，vp1 を含む集合よりも後で開発する順序制約が生じる．

(3) vp1 と vp2 が相互に依存している場合：
{vp1, vp2, VPo}は（{vp1, vp2}, {VPo}）だけに分割できる．vp1 と vp2 の分割開発することはできない．vp1 と vp2 はいずれかを統合テストする前には，共に実装が完了しているという順序制約が生じる．

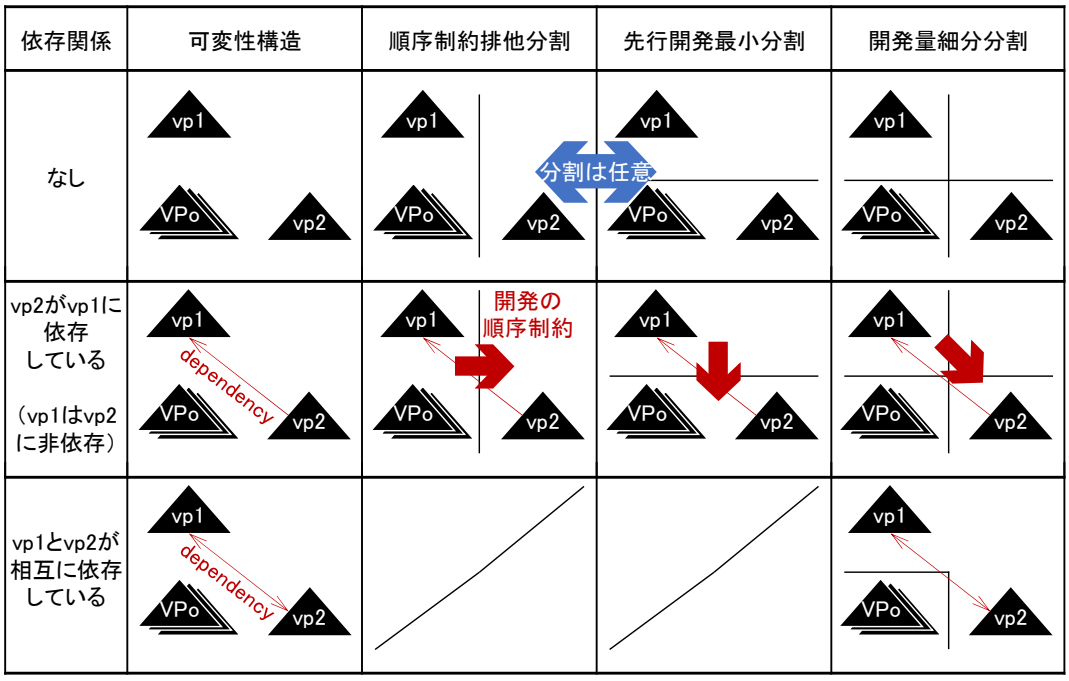

図 6.1 可変性の構造と開発分割方法

6.1.2　OVM 拡張モデルと表記法

既存の OVM（図 3.5）では，依存関係として Requires（必要とする）と Excludes（排他する）が定義されている．これらの依存関係では前項で示した(3)のパターンを明示できないため，新たな依存関係として Co-Requires（共に必要とする）を導入する．

Requires，Excludes，いずれの依存関係も付与されていない可変点同士は，前項(1)のパターンとして識別できる．いずれかの依存関係が付与されている場合は，前項(2)のパターンとして識別できる．具体的な分析方法は 6.2 で示す．

前項(3)のパターンは，Requires の依存関係が双方向に成立している場合に規定される．双方向の依存関係を 2 つの単方向の依存関係で表現することは煩雑となる．そのため， Co-Requires（共に必要とする）という依存関係を定義し，双方向の依存関係を明示できるよう OVM を拡張する．図 6.2 に，拡張した OVM の表記法とメタモデルを示す．

OVM 拡張モデルを利用して可変性の構造を表現した例を図 6.3 に示す．本研究では，関連した表記法も拡張した．従来の OVM では，可変点から開発資産への依存矢印は，可変性構造図から他の開発資産の表現図へ直接依存関係を矢印線として記述している．これに対し本研究では，分析時に関心事に集中できることを目的として，複数開発資産と分析図を分割した．そのため，可変点，変異体に識別番号を付与し，各開発資産の表現図内で識別番号領域（▲や■）による関連付けを可能とした．

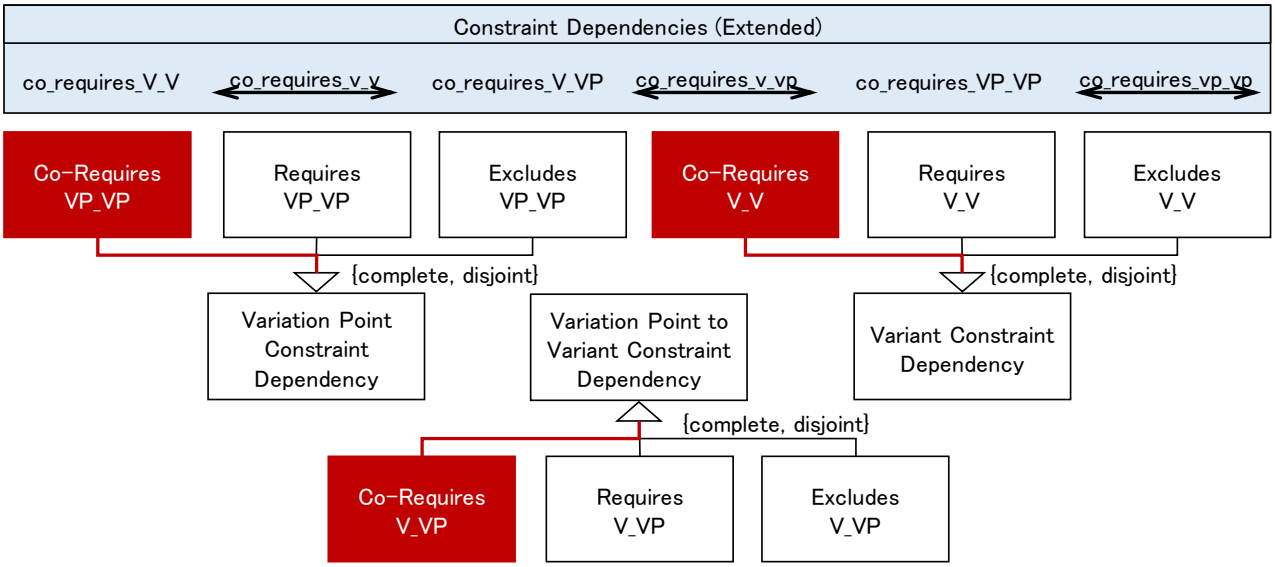

図 6.2 OVM の拡張表記法とメタモデル

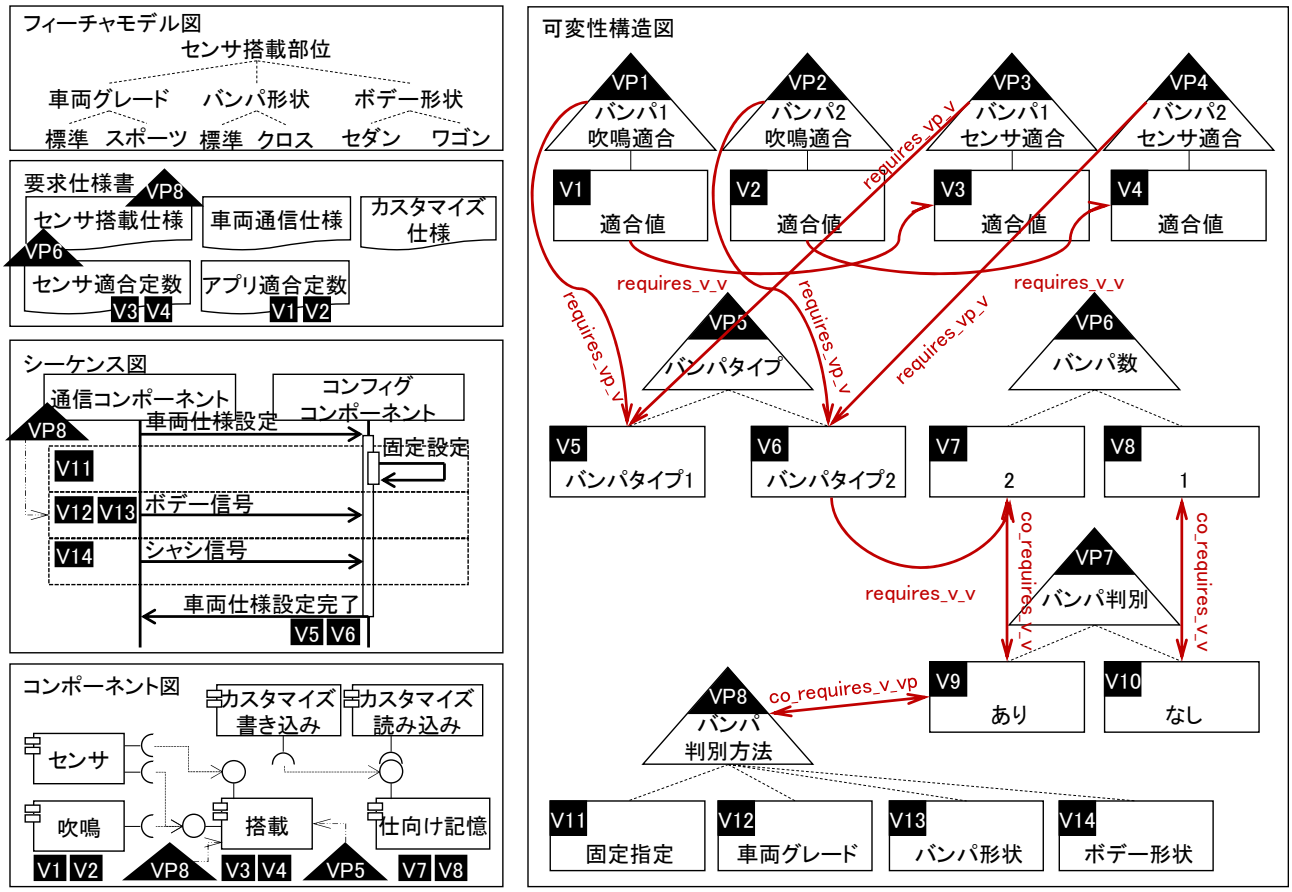

図 6.3 拡張 OVM による可変性の構造記述例

6.2 SPLE における可変性構造の分析方法

OVM 拡張モデルを用いて SPLE における可変性の構造を分析する方法を示す．SPLE における OVM 拡張モデルの生成方法を示し，開発順序制約の分析方法を示す．

6.2.1 SPLE における OVM 拡張モデルの生成

本項では SPLE における OVM 拡張モデルを生成するスコープを定義する．スコープ定義は，SPLE の形態に基づく定義と，モデルの生成タイミングに基づく定義の 2 つの基準がある．これらの 2 つの基準に基づき，適切にスコープを定義する必要がある．

(1) SPLE の形態に基づくモデルのスコープ定義

OVM 拡張モデルは依存関係にある可変点と変異体の集合をスコープとしている．SPLE の形態によってはスコープをさらに分割することで，関心事を限定して分析コストを低減できる．SPLE では次の 2 つの形態を取り得ることから，それぞれに応じたスコープの定義が必要となる．

(a) コア資産ベーススコープ定義

1) SPLE の形態：製品は特定のコア資産から開発するが，並行運用するコア資産が複数存在する[35].
2) モデルのスコープ：コア資産ごとに OVM 拡張モデルを生成する．コア資産同士は並行に運用されるが，依存関係は独立して扱えるためである．

(b) モノリシックスコープ定義

1) SPLE の形態：製品の構成コンポーネントが，コンポーネントごとにコア資産を個別に有する[48].
2) モデルのスコープ：すべてのコア資産をスコープに含める．可変性の依存はコンポーネントを横断して現れる．スコープを分割すると分析すべき依存関係を見落とすリスクが生じる．

(2) モデルの生成タイミングに基づくモデルのスコープ定義

OVM 拡張モデルを生成するタイミングによっても，モデルを生成するスコープを分割することで，関心事を限定し，分析コストを低減できる．

(a) ドメイン開発時

1) モデルの生成タイミング：ドメイン開発時に可変点を設計するときにモデルを生成する．
2) モデルのスコープ：設計対象となるすべての可変点，変異体をスコープとしてモデルを生成する．

(b) アプリケーション開発時

1) モデルの生成タイミング：アプリケーション開発時に可変点を設計するときにモデルを生成する．
2) モデルのスコープ：要求分析や設計工程で分析対象となる可変点，変異体，コア資産にスコープを限定する．アプリケーション開発では開発期間の制約がドメイン開発よりも強く，すべての可変点，変異体を対象にモデル生成することは，投資対効果として低いため，今回の開発で変更を加える可変点を優先してモデルを生成する．但し，依存関係の見落としが発生し得るような，コンポーネント単位などでのスコープ限定は避ける．

6.2.2　可変性の依存関係による開発制約の分析

可変性の依存関係によって，アプリケーション開発における開発アイテムの分割と開発順序に制約が生じる．これらの制約を明確にすることで，合理的な開発計画が設計可能となる．これらの制約は，6.1.1，6.1.2 で示した通り，OVM 拡張モデルを用いて分析が可能である．開発アイテムの分割への制約と分割後の開発順序制約を示し，分析において明確にすべき，依存元と依存先における可変点と変異体の組合せの作用と，Excludes 依存関係の扱いについて述べる．

(1)　開発アイテムの分割方法の分析

開発アイテムの分割方法は，6.1.1 で示した通り，OVM 拡張モデルにより，可変性の依存関係を明らかにすることで分析できる．依存関係を明らかにした後は，どの程度の開発量に開発アイテムを分割したいかによって任意の分割が可能である．

(2)　開発順序制約の分析

開発順序の制約も，6.1.1，6.1.2 で示した通り，OVM 拡張モデルにより，可変性の依存関係を明らかにすることで分析できる．但し，可変性としての依存関係と，開発順序制約としての依存関係は，依存の方向で一致しない場合がある．OVM 拡張モデルで定義した依存関係ごとの順序制約を図 6.4 に示す．

可変点同士に依存関係がない場合，Requires の依存関係の場合，Co-Requires の依存関係の場合は，可変性の依存関係と順序制約が一致する（図 6.4－(A)，(C)，(D)）．Excludes の依存関係の場合に順序制約が反転する．詳細な理由を(4)にて述べる．

(3)　依存元と依存先における可変点と変異体の組合せ

依存元と依存先の対象は，可変点と変異体で組合せが存在する．しかし，どのような組合せでも，可変点同士の依存関係として扱える. 例えば, 図 6.4－(A)のように, VP2 の変異体 v3 から VP1 の変異体 v1 に Requires の依存があるとする．VP2 で変異体を決定するためには，VP1 において v1 が選択されるか否かが決まってからでないと，VP2 で v3 と v4 が選択可能であるのか，v4 のみ選択可能であるかが定まらない．

(4)　Excludes 依存関係の扱い

Excludes の依存関係は，依存の方向を反転させた Requires と同じ扱いとする．例えば，図 6.4－(B)のように，VP1 の変異体 v1 から VP2 の変異体 v3 に Excludes の依存があるとする．VP2 で変異体を決定するためには, VP1 において v1 が選択されるか否かが決まってからでないと, VP2 で v3 と v4 が選択可能であるのか, v3 が排他されて v4 のみ選択可能であるかが定まらない．

図 6.3 で示した OVM 拡張モデルであれば，図 6.5 で示す可変点の順序集合に沿って，インクリメンタルに開発できる．

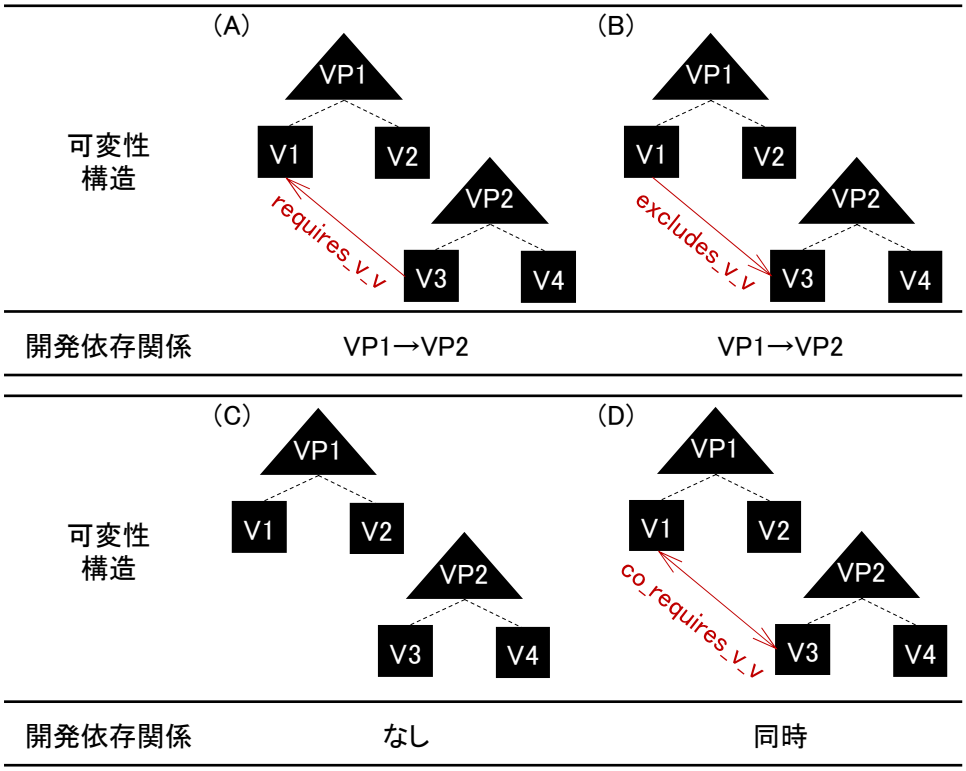

図 6.4 可変性の構造と開発順序制約

{ VP6, VP7, VP8 }→{ VP5 }→{ VP3, VP4 }→{ VP1, VP2 }

図 6.5 可変点の集合を分割した開発順序生成例

6.3 可変性の構造分析に基づくアジャイルアプリケーション開発方法

6.3.1 開発モデル

本研究では，図 6.6 に示すインクリメンタルなアジャイルアプリケーション開発モデルを提案する．本モデルは，OVM 拡張モデルによる可変性の構造分析に基づいて機能する．

本モデルは，アプリケーションバックログ設計，機能開発，可変点開発の 4 つのアクティビティで構成される．本モデルを取り入れることで，SPLE のためのアジャイル開発方法（5 章）において，プロダクト開発アクティビティ（5.2.2.2）の「プロダクトバックログの作成」プロセスは「アプリケーションバックログ設計」に置き換えられ，両モデルが統合される．以下，各アクティビティの詳細を示す．

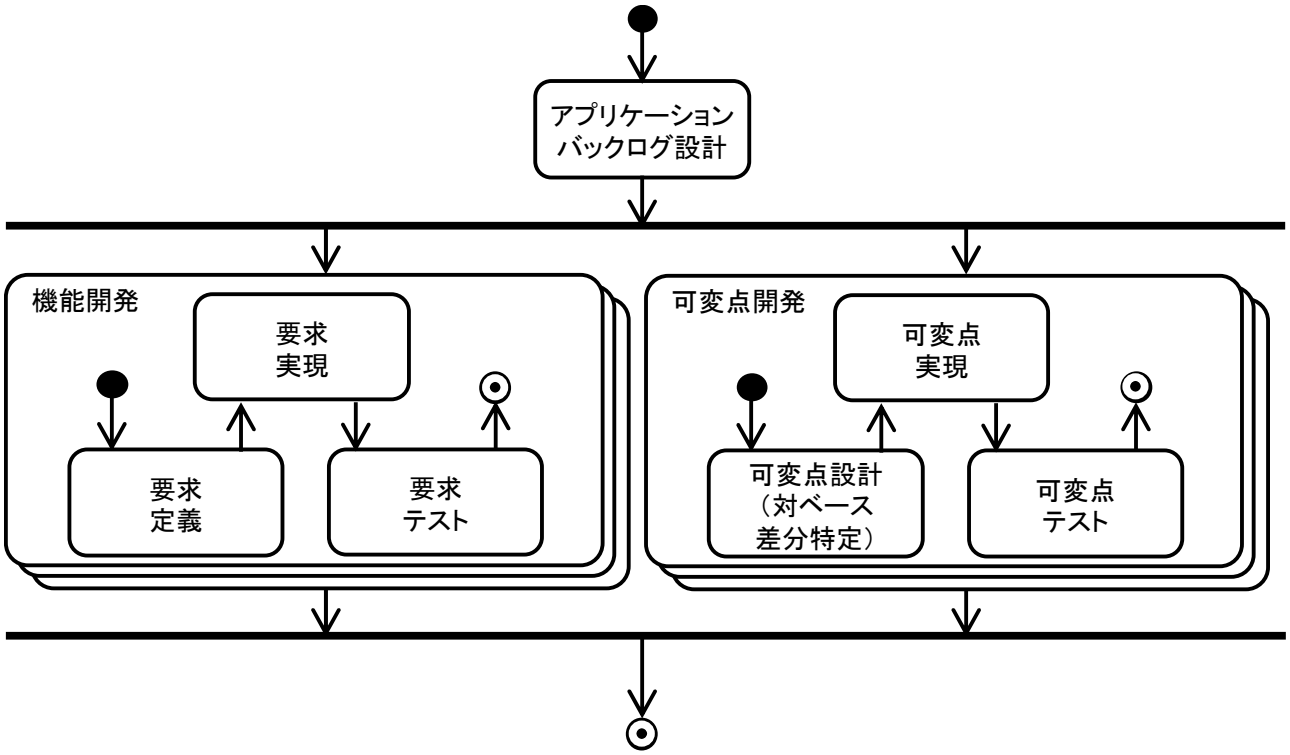

図 6.6 インクリメンタルなアプリケーション開発モデル

6.3.2 アプリケーションバックログ設計

6.2.2 による OVM 拡張モデルの分析で明らかになった分割方法を用いて，INVEST の方針に従ってアプリケーションを開発するバックログを設計する．バックログは，製品の開発に必要なストーリの集合であり，開発順序を規定した開発アイテムのリストである[9].

バックログは，OVM 拡張モデルの分析結果と開発規模の見積りを比較して設計する．依存関係が複数の可変点を連絡する場合，開発分割方法は複数案できる．そのため，分割方法を変えながら，開発規模の見積もりを繰り返すことで，規定のタイムボックスに収まる分割方法を採択して，ストーリを定義してバックログ設計する．本バックログに従って，機能開発と可変点開発のストーリをインクリメンタルに開発していく．

6.3.3 機能開発

製品固有の機能開発は従来の ASD と同様に開発する．機能開発のアクティビティは要求定義，要求実現，要求テストの 3 つのタスクを実行する．要求定義タスクではストーリの実現内容と実現方法を定義する．要求実現タスクでは定義に従ってソフトウェアを実装する．要求テストタスクでは定義に従ってソフトウェアが実現されているかをテストする．

ストーリごとに要求定義から要求テストまでのタスクを一通り実施することで，テスト工数とテスト環境の負荷を平準化する効果が期待できる．ストーリごとにテストを実施することで，一度のテスト実施工数が小さく，開発期間を通してテストが分散されるためである．

機能開発の中で新たな可変点が発生した場合は，OVM 拡張モデルを用いて既存の可変点と新たな依存関係が生じないかを設計検証する．新たな依存関係が生じた場合，依存関係に応じた順序制約を分析して開発順序を見直す．その結果，未開発の可変点開発に含まれる開発アイテムを先行して開発する必要がある場合は，開発対象のストーリを一度バックログに戻すか，依存箇所を追加で実現するバックログを生成してシステムに副作用がない状態で実現する．

6.3.4 可変点開発

可変点開発では変異体の実装を対象にストーリ開発する．可変点開発のアクティビティは可変点設計，可変点実現，可変点テストの 3 つのタスクを実行する．可変点設計タスクでは可変点においてどの変異体に決定するかを設計する．設計では開発対象の製品と類似製品をベースソフトウェアとして定める．ベースソフトウェアに対して変異体を変更するか否かを分析する．可変点実現タスクでは，設計で決定した変異体を新たに実現する必要がある場合は追加実装し，既存の変異体である場合はコンフィグレーションする．可変点テストは可変点設計に基づいて実現結果をテストする．

機能開発と同様に，ストーリごとに可変点設計から可変点テストまでのタスクを一通り実施することで，テスト工数とテスト環境の負荷を平準化する効果が期待できる．実現する可変点の集合はそれらの依存関係が明らかになっている．先行する開発は後に続く開発と独立して開発できることがわかっているため，その後のストーリ開発で回帰テストを計画しなくてもよい．開発後期でのテスト工数の増加と集中を軽減することができる．

7 SPL のアプリケーション開発の並行開発アーキテクチャ

本研究では，SPLE でドメイン開発アプリケーション開発の並行開発を可能とするために，開発ビューポイント上の可変性のモジュール化を実現するリファクタリング方法を提案する．提案方法のモデルは開発言語系に依存しないが，実装は開発言語系に依存する．本研究では，開発言語として C 言語を対象に実装例を提示する．

7.1 可変性のモジュール化リファクタリング方法

開発ビューと論理ビューを利用しながら，開発ビュー上の可変性をモジュール化するためにアーキテクチャをリファクタリングする方法の手順を図 7.1 に示す．本手順は，開発ビューと論理ビュー，個別のビューでのアクティビティと，両方のビューを横断したアクティビティから構成される．

開発ビューでは可変点の特定と可変点の再配置のアクティビティを実行する．論理ビューでは可変性のモデル化を実行する．横断したビューではマッピングルールの策定を実行する．各アクティビティについて，以下の節で示す．

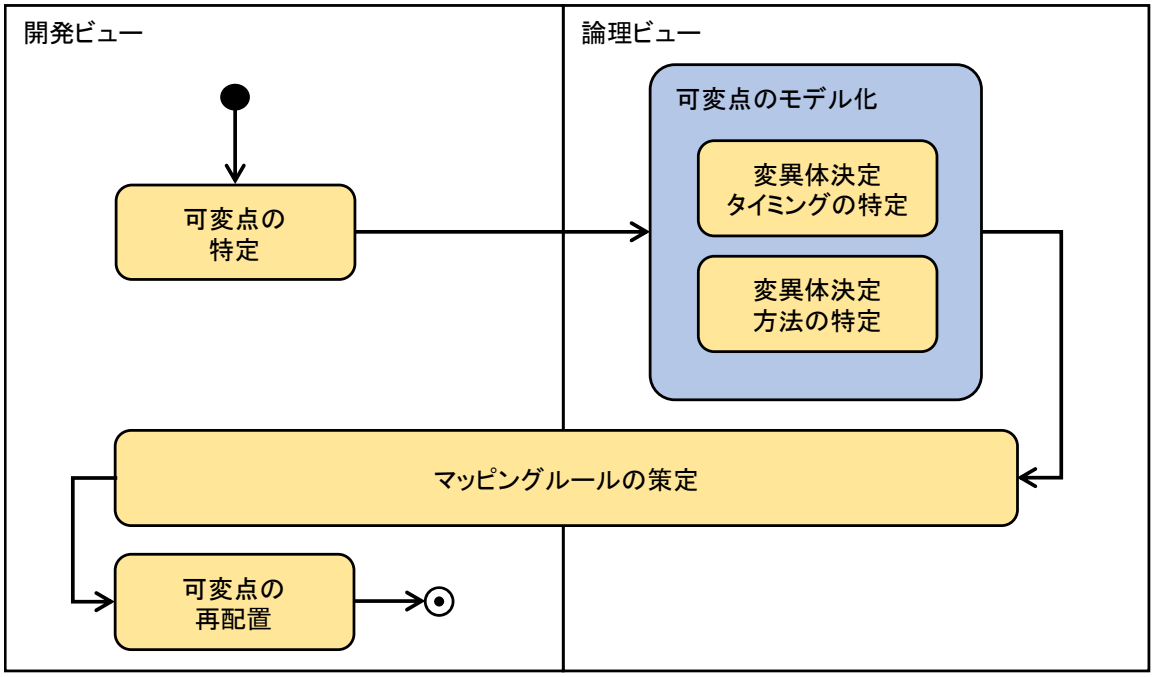

図 7.1 可変性をモジュール化するリファクタリング手順

7.2 可変点の特定

可変点の特定はソフトウェアに含まれる可変点をもれなく抽出し，リストアップするアクティビティである．可変点のリストが予め特定され，管理されている場合は本アクティビティの省略が可能である．可変点のリストが不明，あるいは管理の欠如がみられる場合は，本アクティビティによってリストを作成，または補完する．

可変点のリストアップ方法は，組織の可変点の実装方法ならびに言語系に依存する．オブジェクト指向言語であれば，クラスの継承関係から抽出する方法が有効である．また，アスペクト指向言語であれば，アスペクトから抽出する方法で実装できる．C 言語の場合は，#ifdef マクロや#define マクロを対象としたキーワード検索，ならびに関数ポインタの抽出によりリストアップすることが可能である．

7.3 可変点のモデル化

可変点のモデル化は，変異体決定タイミングの特定と変異体決定方法の特定のアクティビティを実行し，論理ビュー上で可変点をモデル化する．可変点をモデル化して分類することで，所有しているソフトウェアの可変点をパターン化し，開発ビュー上で再配置するためのルールを策定するための標準化を行う．

7.3.1 変異体決定タイミングの特定

各可変点の変異体の決定タイミングを特定する．決定タイミングは以下 3 種類に分類される[4]．決定タイミングは，同時並行開発を阻害するか否かを判断する因子であり，リファクタリング対象の識別に利用する．

(1) コンパイル時

プリプロセッサなどでコンパイル時に変異体を決定する．このパターンの決定タイミングは，変異体追加時に，単一ファイルの編集で衝突が発生する可能性があり（図 7.2），リファクタリング対象の候補に挙げる．

(2) リンク時，ロード時

リンク対象やロード対象のビルド済みオブジェクトを選択することによって変異体を決定する．このパターンの決定タイミングは，並行開発を阻害しないため，リファクタリング対象の可変点から除外する．

(3) 実行時

システムへのバックアップ値の入力や，その他パラメータの実行時判定によって変異体を決定する．このパターンの決定タイミングも，変異体追加時には，コンパイル時と同じ状況に陥る可能性がある．そのため，リファクタリング対象の候補に挙げる．

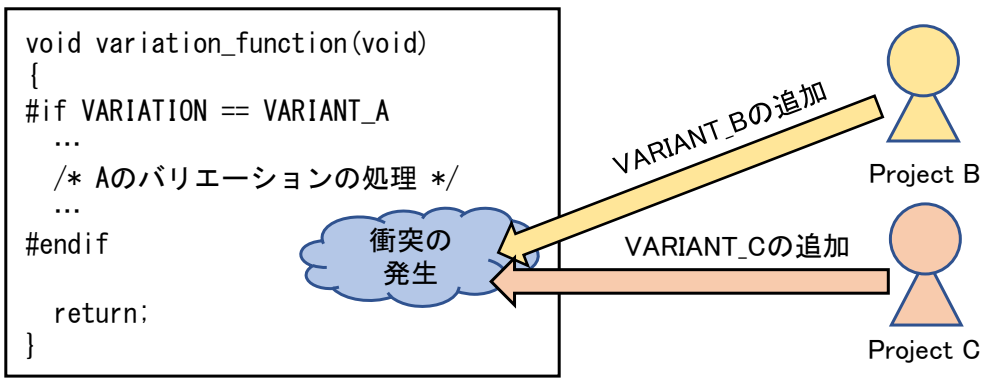

図 7.2 コンパイル時の決定における単一ファイルの編集衝突発生例

7.3.2 変異体決定方法の特定

各可変点の変異体決定方法を分類する．固定方法は可変点をパターンとして分類する際の因子となる．可変点の変異体の決定方法，ならびに実現方法の例を表 7.1 に示す．「プラグイン」のように決定タイミングが自明である決定方法もあるが，分類を目的とした観点と，リファクタリング対象の決定を目的とした観点を切り分けて取り扱う．

表 7.1 を目安に各可変点の実現方法を分類したら，同一の分類の中でさらに実現方法の共通性を分析してパターン分類を進める．マッピングルールの策定アクティビティは，この分類ごとにマッピングルールを策定する．

表 7.1 可変点の実現方法と変異体の決定方法[36]

可変点の実現方法・ 変異体の決定方法	概要
継承	オブジェクト指向におけるクラスやインタフェースの継承.
拡張	異なり得る機能を固定的な機能から取り除き拡張可能に．典型例は Strategy パターン.
構成	本体と分離されたリソースを利用．定義ファイルなど.
パラメータ	複数のモジュールを内包させ，パラメータで選択．テキストプリプロセッサも含む.
テンプレート	特定の型に応じてコンポーネントの定義をインスタンス化.
生成	仕様や設計記述からコードを生成.
コンポーネント代替	開発時に複数の代替モジュールから選択して挿入.
プラグイン	実行時にコンポーネントを選択して挿入.
アスペクト	アスペクト指向プログラミング利用．可変性をアスペクト化.
実行時条件	設定や定義ファイルによる実行時の切り替え.

7.4 マッピングルールの策定

7.3 で分類された可変点のモデルごとに，開発ビュー上に再配置するためのマッピングルールを策定する．マッピングルールは，可変点の実装方法の変換ルールと，構成管理上の配置方法の変換ルールから構成される（図 7.3）．

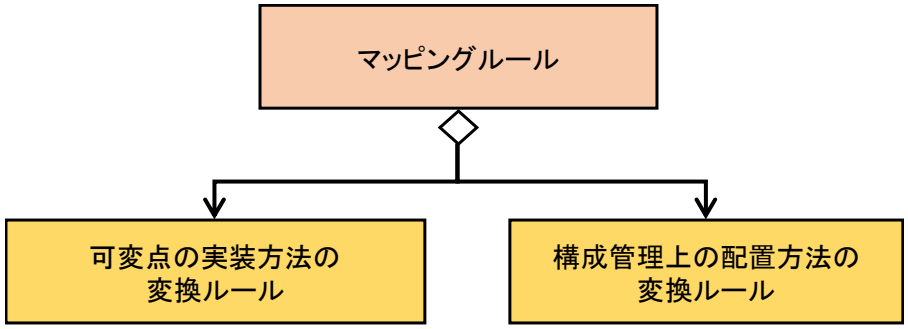

図 7.3 マッピングルールの構成

7.4.1 可変点の実装方法の変換ルール

可変点の実装方法の変換ルールは，変異体決定方法の変換ルールである．決定方法を変更することで，アプリケーション開発の同時並行開発が可能となるよう，構成管理上の配置方法を変換することが可能となる．

図 7.4 に可変点の実装方法の変換ルール例を示す．この場合，変換前は表 7.1 における「パラメータ」に分類される決定方法を採用している．この場合，(a)各製品を表現するパラメータが変異体となっている．変換後では，同じようにパラメータで変異体を指定するが，製品を示すパラメータではなく，(b)(c)機能としての変異体を表すパラメータに変換されている．

本ルールの適用だけでは，複数アプリケーションの同時開発時に単一ファイルの編集の衝突を抑制することはできない．構成管理上の配置方法の変換ルールと組み合わせる必要がある．

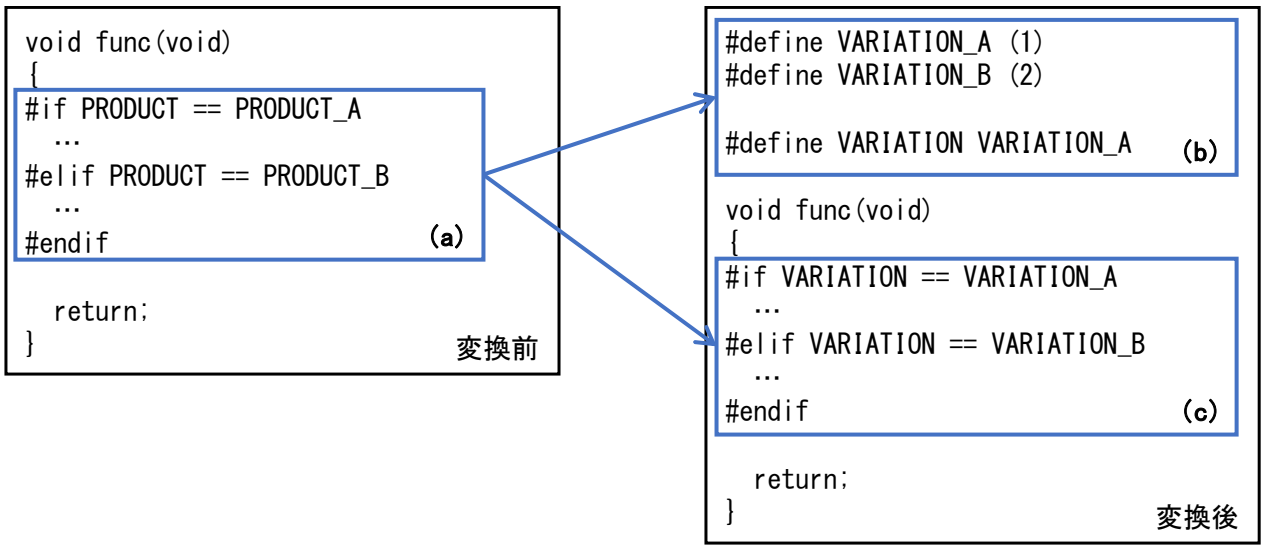

図 7.4 可変点の実装方法の変換ルール例

7.4.2　構成管理上の配置方法の変換ルール

構成管理上の配置方法の変換ルールでは，開発者が編集するためのソースファイルの配置方法の変換ルールを提供する．可変点の追加や変異体を決定，あるいは追加するときに，編集するファイルが衝突しないように，モジュールとしてファイルの配置を独立させる．

図 7.5 に，構成管理上の配置方法の変換ルール例を示す．本例は，図 7.4 で実施した，実装方法の変換ルールを適用し，変更された後の実装方法に基づいている．元の関数本体のファイルが収まるモジュールに変更はない．一方，機能としての変異体のパラメータの定義は，バリエーション共通ファイルとして独立定義される．また，製品ごとにバリエーションを指定するパラメータファイルは，製品レイヤに製品ごとに格納される．こうすることで，製品 A と製品 B が同時に開発されていても，各開発担当者は，独立したモジュールを編集するだけで，互いに独立した開発を維持することができる．

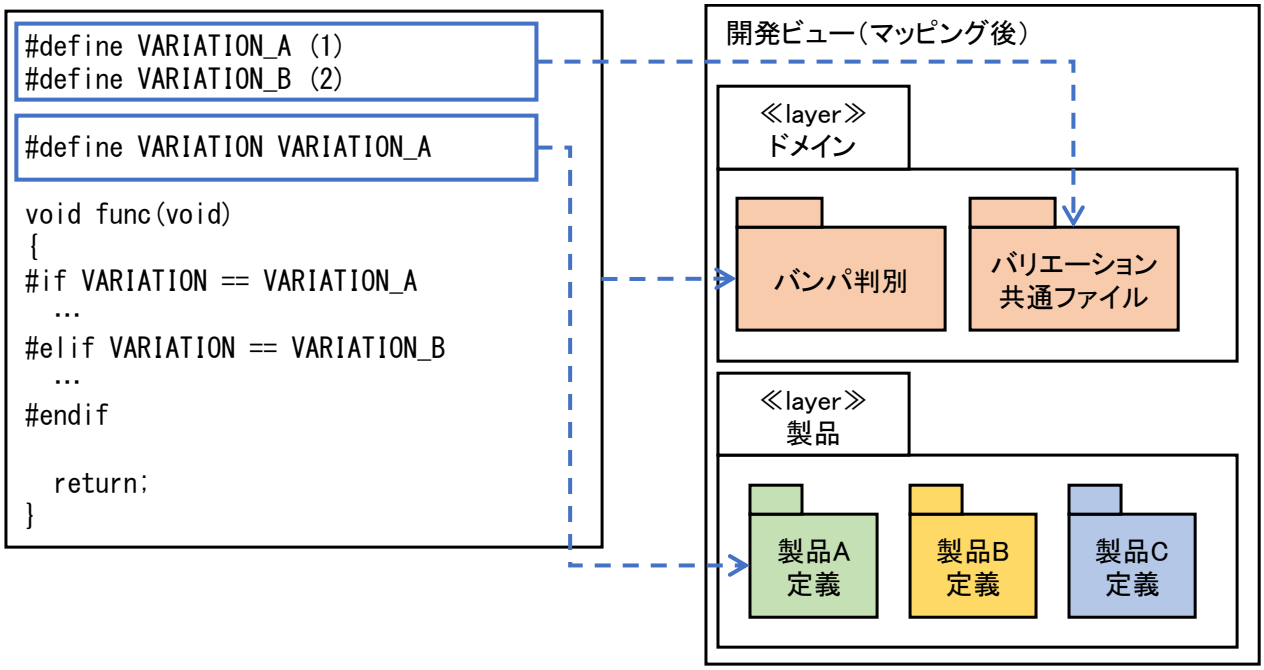

図 7.5 構成管理上の配置方法の変換ルール例

7.5 可変点の再配置

7.4 で定めたマッピングルールに則り，モジュールの構成を変更することを可変点の再配置と呼ぶ．マッピングルールは標準化されたルールであり，可変点を再配置するには，対象の可変点ごとにルールを具体化して適用する．

モジュール構成変更後は，変更前後でコア資産として変異体決定後の振る舞いが変わらないことの確認と，リファクタリングすべき可変点が漏れていないかを確認する．特に，対象とすべきリファクタリング対象が不足している場合，開発時に編集が衝突するファイルが存在することになり，開発の阻害要因となる．このため，同時並行開発が可能となった可変点が存在していたとしても，効果が限定的となるリスクが生じる．

8 自動車システム APLE への適用

本研究で提案した SPL の管理可能な開発方法論を，自動車システム APLE の実開発に適用し評価した．研究者は開発チームのリーダ，SPL を統括するマネージャとして参加し，データ収集と現場観察を実施した．

8.1 適用対象の自動車プロダクトライン開発のコンテキスト

図 8.1 に適用対象の自動車システム開発における開発組織と SPL 上の製品リリースの流れを示す．開発は 2 つの組織体から構成されている．コア製品開発チーム（以下，コアチーム）と派生製品開発チーム（以下、派生チーム）が，SPLE におけるドメイン開発とアプリケーション開発をそれぞれ分担している．

コアチームは，SPL に向けて可変点を織り込んだコア資産を開発する．また，SPL としてコア資産からいくつかの製品をアプリケーションとして開発する．その結果，2 つのコア製品 A と B をリリースする．派生チームはプロダクトライン 1 としてのコア資産を再利用して派生製品の集合 C，D，E を開発する．

コアチームは車両メーカの要求に対応すべく，コア製品の世代を成長させてコア資産を進化させる．例えば，主要機能の向上を果たしてコア製品 F をリリースする．派生チームは進化したプロダクトライン 2 を再利用して G と H という派生製品を 2 つリリースする．同様に，コアチームはさらにコア資産を進化させてコア製品 I と M をリリースする．派生チームはプロダクトライン 3 から m を再利用して派生製品 J から Q をリリースしていく．

自動車システムの多様な構成によって複数の SPL が生成される．加えて，プロダクトライン 1 とプロダクトライン 2 のように，複数の SPL を重なった期間の中で並行して開発することが生じ得る．プロダクトライン 2 とプロダクトライン 3 においてもまた複数の開発が並行に実行される．また，車両メーカの要求は年々多様化と俊敏化を推進しており，新たな SPL の立ち上げと派生製品を市場へ投入するまでの時期も短くなっている．その結果，コア資産の開発と派生製品としてのアプリケーション開発も並行化が進んでいる．

このような開発モデルでは，派生チームにおいて複数の SPL を対象とした短期間で並行する複数開発を束ねて包括管理することが課題となる．例えば，コアチームは年間 1 つか 2 つの SPL を生成する一方，派生チームは年間で 15～20 の派生製品をリリースする必要が生じる．加えて，ひと月の間にも派生チームは 1 から 3 の SPL を並行して開発対象としなくてはならない状況となっている．

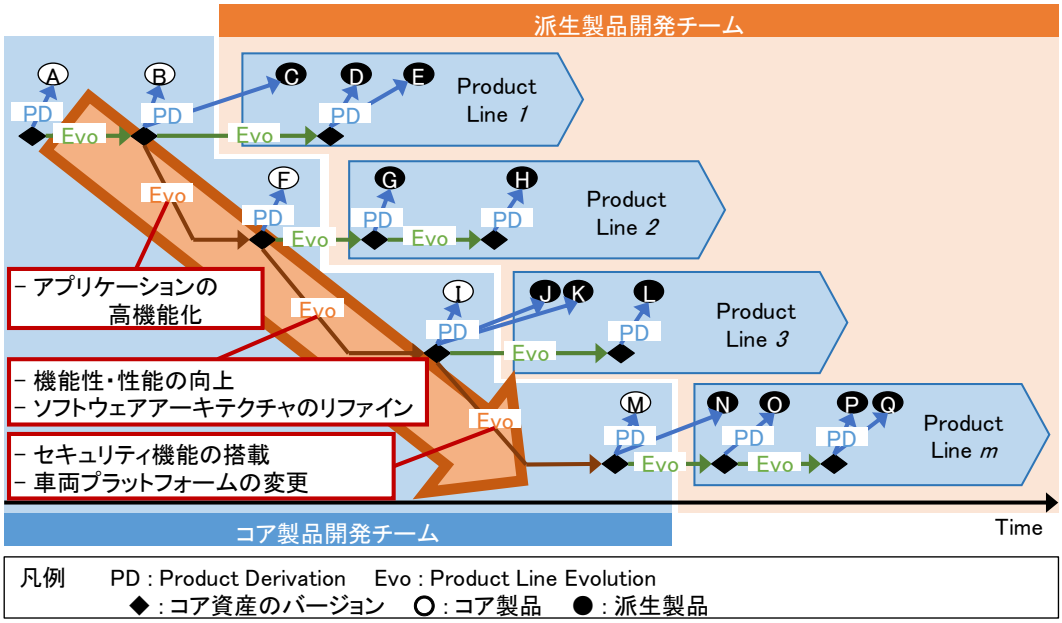

図 8.1 多様で並行な自動車システムの SPL 開発

8.2 適用対象のプロジェクト

本研究で提案した，SPLE のためのアジャイル開発方法（5 章）と可変性のモデル化と構造分析（6 章）に基づくアジャイルアプリケーション開発方法， SPL のアプリケーション開発の並行開発アーキテクチャ（7 章）を，対象の自動車システム APLE として段階的に適用した．以下にそれぞれの適用時期を示す．

8.2.1 SPLE のためのアジャイル開発方法の適用

適用期間は 2015 年 7 月～2016 年 4 月の 10 か月間である．1 スプリントは 2 週間としており，期間中に 22 スプリントを実行している. チームメンバは 3 スプリント目以降で増員しており, その後の人員の変動はない. 期間中に 11 のプロジェクトに取り組んだ．これらのプロジェクトで適用して得られたデータで本方法の有効性を評価した.

8.2.2 可変性のモデル化と構造分析に基づくアジャイルアプリケーション開発方法の適用

適用期間は 2016 年 12 月～2017 年 2 月の 3 か月間である．対象プロジェクトは 1 プロジェクトである．1 スプリントは 1 週間とし，9 スプリントで開発を終えている．開発人員は 3 名，テスト環境は 1 台である．評価の比較対象として，2016 年 8 月～10 月で開発した類似規模の製品開発を 1 プロジェクト選択した．比較対象プロジェクトは 8.2.1 と同様の開発方法で開発されたプロジェクトである．各プロジェクトで得られた実データで本方法の有効性を評価した.

8.2.3 SPL のアプリケーション開発の並行開発アーキテクチャの適用

適用期間は 2017 年 4 月～2017 年 10 月の 7 か月間である．この内，並行開発アーキテクチャへのリファクタリングの実施期間が 2017 年 4 月～2017 年 8 月の 5 か月間である．また，リファクタリング後のドメイン開発とアプリケーション開発の並行開発期間は 2017 年 9 月～2017 年 10 月の 2 か月間である．この期間中，コア製品開発はドメイン開発とアプリケーション開発 3 プロジェクトを同時開発しており，派生製品開発では，アプリケーション開発 2 プロジェクトを同時並行開発した．各プロジェクトで得られた実データで本方法の有効性を評価した.

リファクタリングは, 派生チームが担当している. リファクタリングを実行するプロセスとして, AR (3.4.2) を適用した．スプリントごとにリファクタリング対象のソフトウェアコンポーネントを選択し，コアチームの開発と衝突しないようにリファクタリング結果をソフトウェアに組み込む方法として適用している.

8.3 適用した開発環境

図 8.2 に，本適用におけるプロセス資産を活用した開発管理環境を示す．プロセス資産はチケット駆動開発ツールである JIRA[1]と構成管理ツールである Git[2]で収集される．ソースコード管理では Bitbucket[3]サービスを利用して JIRA と Git を連携させている．

プロセス資産モデルにおけるユニット名，分類名，見積り規模は JIRA のチケットに登録される．チケットとして作成されたプロセスユニットを利用して，プロダクト計画やプロダクト構築の管理を JIRA のプランニングボード上に展開する．JIRA の記録を参照することで，プロセスユニットに収集されたプロセス成果物や見積り規模を参照することが容易となる．

プロセス成果物は Git に登録される．プロセス定義は手順やガイドラインとして各成果物に埋め込まれ，プロセス成果物と共に Git 上に登録される．

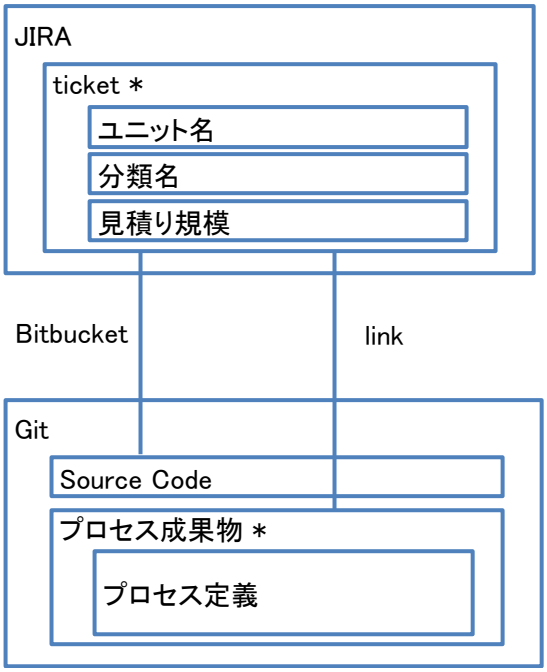

図 8.2 プロセス資産と開発管理環境

[1] JIRA: https://www.atlassian.com/software/jira.
[2] Git: https://git-scm.com/.
[3] Bitbucket: https://bitbucket.org/.

9 評価

8.2 で示した通り，SPL の開発全体を包括した管理可能な開発を目指し，各アプローチを順に加えて適用評価した．以下に，アプローチを加えるごとの開発管理性の向上についての評価を示す．

9.1 アジャイル開発フレームワーク評価

提案するアジャイル開発フレームワークの効果を以下の観点で評価した．

(1) プロセス資産の反復性

(2) 複数 SPL に対する管理性

(1)は本開発フレームワークの前提条件の検証であり，(2)は本開発フレームワークによる効果の検証に位置づく．

9.1.1 プロセス資産の反復性評価

試行した 11 のプロジェクトで実行されたプロセスユニットについて，プロセスユニット群として反復した回数の統計を図 9.1 に示す．本プロジェクトでは，プロセスユニットを要求開発，設計，実装，試験に分割しているため，分類ごとに集計している．

プロセス資産には，合計で 268 のプロセスユニットが登録された．その内，プロセスユニット群（単体のプロセスユニットを含む）は 15 種あった．15 種のプロセスユニットは，実行されたプロセスユニット全体の 84.7%を占めている．要求開発は，プロダクトを対象に実行されるため，プロジェクトの数が最大の反復回数となっている（A）．設計，試験は変異体を対象に実行され，かつ対象が異なっても類似のプロセスユニット群を反復することが多いため，プロジェクト数を超えて反復されている（E，F，K）．実装は変異体の固定で反復されるため，設計の反復回数よりは少ないが，同様にプロジェクト数を超えてプロセスユニット群が反復される結果が得られた（J）．

15.3%のプロセスユニットは，反復を伴わない一過性の調査や，プロトタイプ製品リリースのために派生プロダクトラインを進化させるフィーチャ開発であった．

自動車システム開発の SPL 開発において，プロセス資産に反復性があることが確認できた．

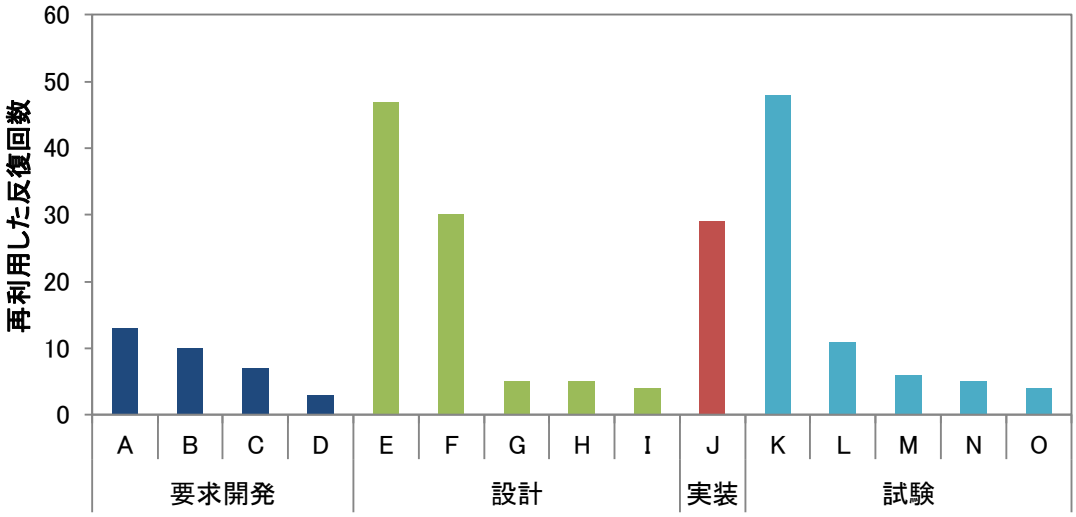

図 9.1 プロセスユニットの反復回数の統計

9.1.2　複数 SPL に対する管理性向上評価

複数 SPL の管理に本管理方法が有効であるかを確認するため，開発の安定性と見積りの正確性を評価した．

9.1.2.1　開発の安定性評価

開発管理の容易性の指標として生産性の予測可能性がある．予測可能性を評価する指標として，開発における生産性の変動性を式 9.1 に定義する．

$$\text{生産性の変動性} = \text{特定の時間単位における生産性の変化率} \tag{9.1}$$

生産性の変動性は，仮にあるスプリントで消化したプロセスユニットのストーリポイントの総和の変化率が小さければ，生産性の変化率もまた少ないことを意味する．開発の安定性はプロセスの生産性における変化率によって定義される．小さな変化率は安定性の高さを意味し，予測可能性が高いといえる．

スプリントごとの消化ポイントの推移を図 9.2 の上部に示す．青色のドット線は，7 スプリントごとの移動平均であり，生産性を示す．生産性が計算できる第 7 スプリント以降において，生産性の変動性が 20%以内である場合を，変動が少なく開発が安定しているとする．図 9.2 の下部に生産性の変動性の推移を示す．第 8～第 22 スプリントの 15 スプリントすべてが変化率 20%以下となっていた．その内，11 のスプリントは変化率 10%以下となっていた．変化率が 10%を上回る 4 スプリントの要因は以下である．

(1) 長期連休で稼働日が少数のスプリントを含む [1 回]
(2) フィーチャ開発など反復性のないプロセスユニットが多いスプリントを含む [2 回]
(3) 計画外プロセスが発生したスプリントを含む [1 回]

これらの要因を除くと，開発の反復性が高くなると，開発の安定性が高くなることが明らかになった．

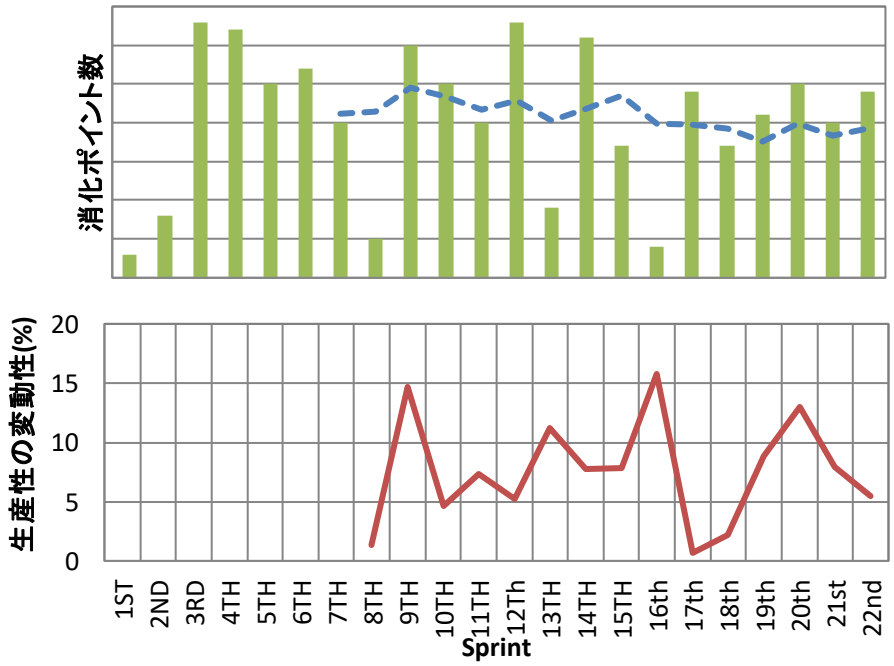

図 9.2 スプリントごとの消化ポイントと生産性の変動性の推移

9.1.2.2　見積りの正確性の評価

開発管理の容易性の指標として開発量の予測可能性がある．開発量の予測可能性を評価する指標として，開発量見積もりの正確性を式 9.2 に定義する．

$$\text{開発量見積りの正確性} = 1 - (\text{計画時の見積り量}) / (\text{完了時の実績量}) \qquad (9.2)$$

ポートフォリオ計画，プロダクト計画での規模見積りと，プロジェクト完了時の実績の比較を図 9.3 に示す．対象は 9.1.2.1 と同様であるが，開発中に顧客要因で要求が変化したプロジェクトを除外し，6 プロジェクトで評価した．誤差率は，開発量見積りの正確性の逆数の 100 分率である．誤差率が低いほど，開発量見積りの正確性は高い．

反復を重ねるごとに，プロジェクトの実績規模が減少した．これは，プロダクト開発が反復されることで，開発への経験不足で実施したムダな作業が省かれていったことと，開発要員が作業に習熟したことで，作業が効率化されプロセス規模が縮小したことによる．

ポートフォリオ計画とプロダクト計画では，プロダクト計画の正確性がより高い．誤差率が増加したプロジェクトもあるが，その後は減少した．ポートフォリオ計画ではほとんどのプロジェクトの誤差率が 30%以下（開発量見積りの正確性は 0.7 以上），プロダクト計画では誤差率が 15%以下（開発量見積りの正確性は 0.85 以上）であった．見積り誤差は初期のプロダクト定義において 60%（開発量見積りの正確性は 0.40），承認されたプロダクト定義において 25%（開発量見積りの正確性は 0.75）というデータ[9]と比較して，高い正確性が得られた．

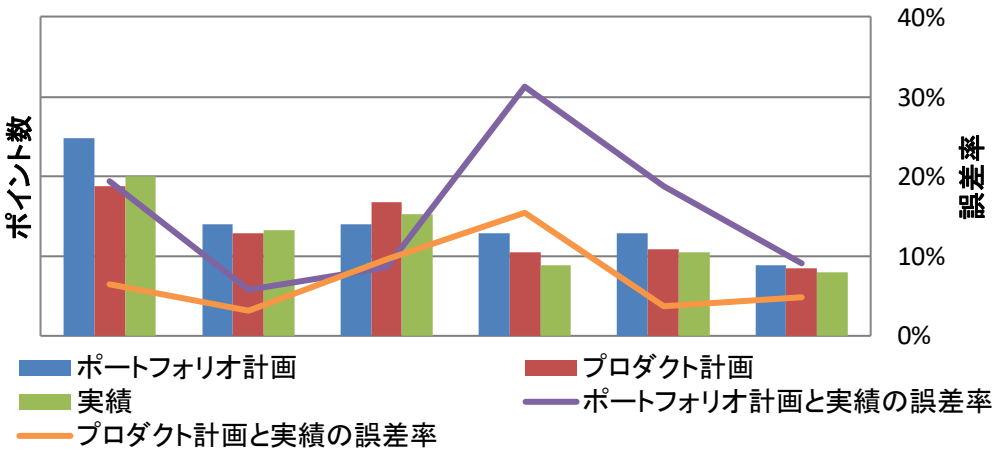

図 9.3 プロダクトの計画と実績の比較

9.2 可変性の構造分析に基づくアジャイルアプリケーション開発方法評価

可変性の構造分析に基づくアジャイルアプリケーション開発方法を以下の観点で評価した.

(1) バリューストリームの速度向上

(2) テスト工数とテスト環境の負荷平準

これらの評価で有効性が確認できることで，管理性向上に対する有効性を評価する.

9.2.1 バリューストリームの速度向上

開発管理の容易性の指標として開発アイテム（要件）ごとのコストと期間（バリューストリームの速度）の可制御性を規定する.開発アイテムのコストと期間の可制御性を評価する指標として,開発アイテムのコスト,期間の変動性を式 9.3，式 9.4 に定義する.

開発アイテムのコスト変動性 ＝ （開発工数の標準偏差）／（開発工数の期待値）(9.3)

開発アイテムの期間変動性 ＝（開発日数の標準偏差）／（開発日数の期待値）(9.4)

開発アイテムのコスト変動性が小さければ，開発アイテムのコストのばらつきが小さく，開発アイテム数からのコスト見積りからの変動が小さく管理が容易となる．開発アイテムの期間変動性が小さければ，バリューストリームの速度のばらつきが小さく，開発アイテムの開発完了が予測し易く管理が容易となる．開発アイテムの開発日数の期待値が小さければ， バリューストリームの速度が高く，各開発アイテムは短期間で開発を終えて仕掛かりが少ない．その結果，新たな要求に応じやすく，俊敏な開発が実現でき，仕掛かり数の少なさは在庫管理の冗長なコストが省かれることで管理性が高いといえる.

バリューストリームの速度向上効果として，提案方法の適用前後における，開発アイテムの要求分析からテスト完了までの開発工数と開発日数を比較して，3 つの指標を評価した．図 9.4 に，開発工数と開発日数を軸に取った，適用前後の開発アイテムごとの散布図を示す．また，表 9.1 に，同一のデータに対して，開発アイテムのコスト変動性と期間変動性，開発日数の期待値を求めた結果を示す.

適用前に対して適用後は，適用後の開発アイテム数が増加している．可変点を独立して開発できるようになったことで，開発アイテムをより小さく分割することができるようになっている．その結果，アイテムごとの開発工数の期待値は 31.5H から 20.0H と低減し，63.5%改善した．コスト変動性は 0.81 から 0.65 へ改善し，管理性が向上していることが確認できた.

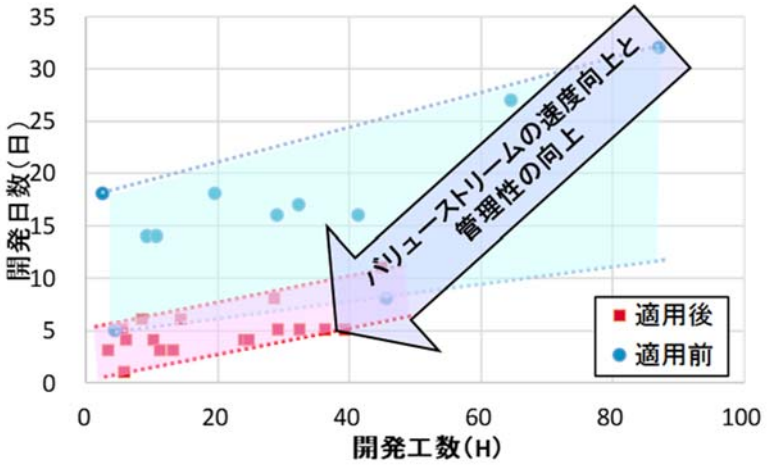

図 9.4 適用前後のバリューストリーム速度

表 9.1 開発アイテムのコスト変動性と期間変動性

	適用前	適用後
アイテム数	11	17
開発工数:期待値(H)	31.5	20.0
開発工数:標準偏差(H)	25.5	13.0
コスト変動性	0.81	0.65
開発日数:期待値(日)	16.8	4.8
開発日数:標準偏差(日)	7.2	2.2
期間変動性	0.43	0.46

バリューストリームの速度を示す開発日数の期待値は 16.8 日から 4.8 日と，3.5 倍に向上した．期間変動性は 0.43 から 0.46 になり，管理性としてのばらつきの程度に有意差はないが，絶対値としてのばらつきは 7.2 日から 2.2 日と 69.4%改善した．開発アイテムの開発規模が小さくなり，アイテムごとの開発工数，開発日数のばらつきが縮小し，バリューストリームの速度が 3.5 倍に向上する結果が得られた．

9.2.2　テスト工数とテスト環境の負荷平準

テスト工数とテスト環境の負荷平準効果として，テスト環境の使用率を計測し，テスト負荷のばらつきを評価した．テスト環境の負荷が一律であれば，テスト環境がボトルネックとなって開発の進捗が停滞することが少なくなり，開発の管理性が向上する．

図 9.5 に提案方法を適用する前後でのテスト環境使用率を示す．適用前では，開発の後半（16 日目以降）でテストが開始され，終盤に向けて使用率が増大している．適用後では，開発開始後 5 日目にはテストが開始され，開発終盤でやや高めになるものの，使用率は一定水準を保った．

表 9.2 にテスト環境使用率の平均値と標準偏差，最大値を示す．分母はテスト環境の使用日数であり，テストを実施した日だけを対象にデータを抽出して算出した．

適用前は，テスト環境を使用した日における環境使用率の平均値は 81.0%であり，標準偏差は 47.5%，最大使用率は 162.5%である．標準偏差と平均値，最大値から，開発環境が不足していることが明らかである．実際に，適用前の開発では，その他の開発チームから開発環境を一時的に借用して補っていた．

適用後の環境使用率の平均値は 48.9%と 32.1%低減し，標準偏差は 28.5%と 19%低減し，最大値も 106.3%へと 56.2%低減した．標準偏差と平均値，最大値の関係から，開発環境は十分であることがわかる．テスト工数は両プロジェクトで同等であり，テスト日数は適用後では 1.7 倍となった．これらの結果から，テスト環境の使用率が平準化され，テスト環境のボトルネック化が解消されたことが確認できた．

テスト工数には，報告書の作成工数など，テスト環境を使用する以外の工数も含まれる．しかし，テスト工数の多くは環境使用の工数であり，テスト環境使用率で近似できる．したがって，本評価結果から，提案方法によって，テスト工数の負荷が平準化され，開発の管理性が向上することが確認できた．

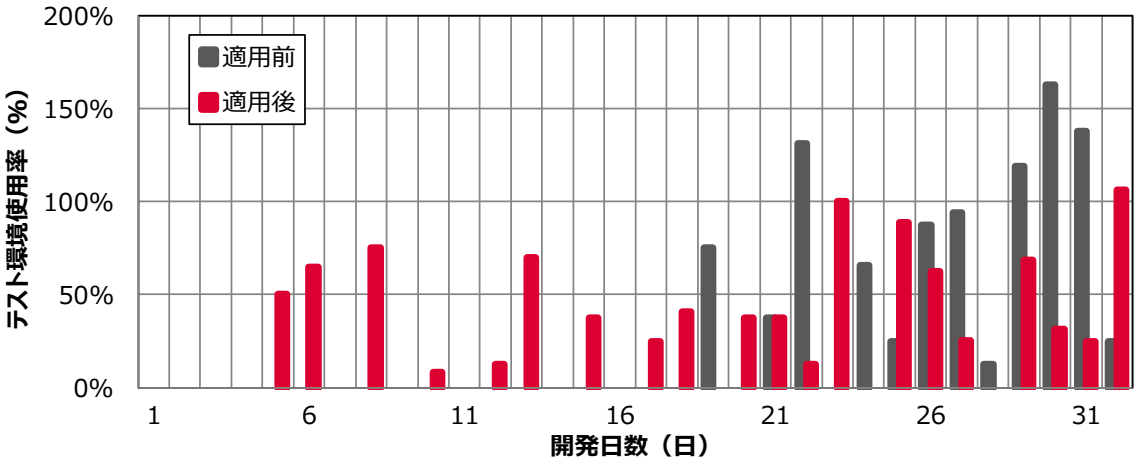

図 9.5 適用前後のテスト環境使用率

表 9.2 テスト環境使用率の平均と最大

	適用前	適用後
テスト日数(日)	12.0	20.0
テスト工数(H)	77.8	78.3
平均(%)	81.0	48.9
標準偏差(%)	47.5	28.5
最大(%)	162.5	106.3

9.3 アプリケーション開発の並行開発アーキテクチャ評価

SPL のアプリケーション開発の並行開発アーキテクチャ設計方法を以下の観点で評価した.

(1) 可変点のマッピングルールの有限性

(2) アプリケーション開発の並行性

これらの評価で有効性が確認できることで，管理性向上に対する有効性を評価する.

9.3.1 可変点のマッピングルールの有限性

可変点のマッピングルールの有限性を評価するため，適用プロジェクトで作成されたマッピングルールの数とパターンを測定した．表 9.3 に適用プロジェクトで作成されたマッピングルールの数とパターンの対応を示す．適用プロジェクトでは，3 パターンのマッピングルールの作成で，ルールを標準化することができた.

P2 のルールは，7.4.1 と 7.4.2 で示したルール例と同一である.

P1 のルールの可変点の実装方法の変換ルールを図 9.6 に示す．構成管理上の配置方法の変換ルールは，P2 と同じである．変換前の実装方法を拡張として分類し，Strategy パターンとして論理ビューで捉え，Strategy を実装上分離して再配置する.

P3 のルールの可変点の実装方法の変換ルールを図 9.7 に示す. 構成管理上の配置方法の変換ルールは, P1, P2 と同じである．変換前の実装方法をアスペクトとして分類し，アスペクト部を実装上分離して再配置する.

開発対象のソフトウェアシステムや開発組織により，実装のパターンは分化すると考えられる．SPL であっても，システムや組織体のバリエーションが少なければ，マッピングルールは有限性を示し，少ないパターンの抽出が可能となることが推測される．また，実装方法の変換ルールは，各実装によって様々であるが，構成管理上の配置方法の変換ルールは，基本ルールを定めれば十分である可能性が確認された.

表 9.3 マッピングルールの数とパターンの対応

分類	抽出	パターン	備考
継承	×	{}	－
拡張	○	{P1}	Strategy パターンとして実装を分離
構成	○	{P2}	コンポーネント代替に移行
パラメータ	○	{P2}	構成化してコンポーネント代替に移行
テンプレート	×	{}	－
生成	×	{}	－
コンポーネント代替	○	{}	リンク時の決定につきリファクタリング対象外
プラグイン	×	{}	－
アスペクト	○	{P3}	アスペクトを分離してコンポーネント代替に移行
実行時条件	○	{}	実行時の決定につきリファクタリング対象外

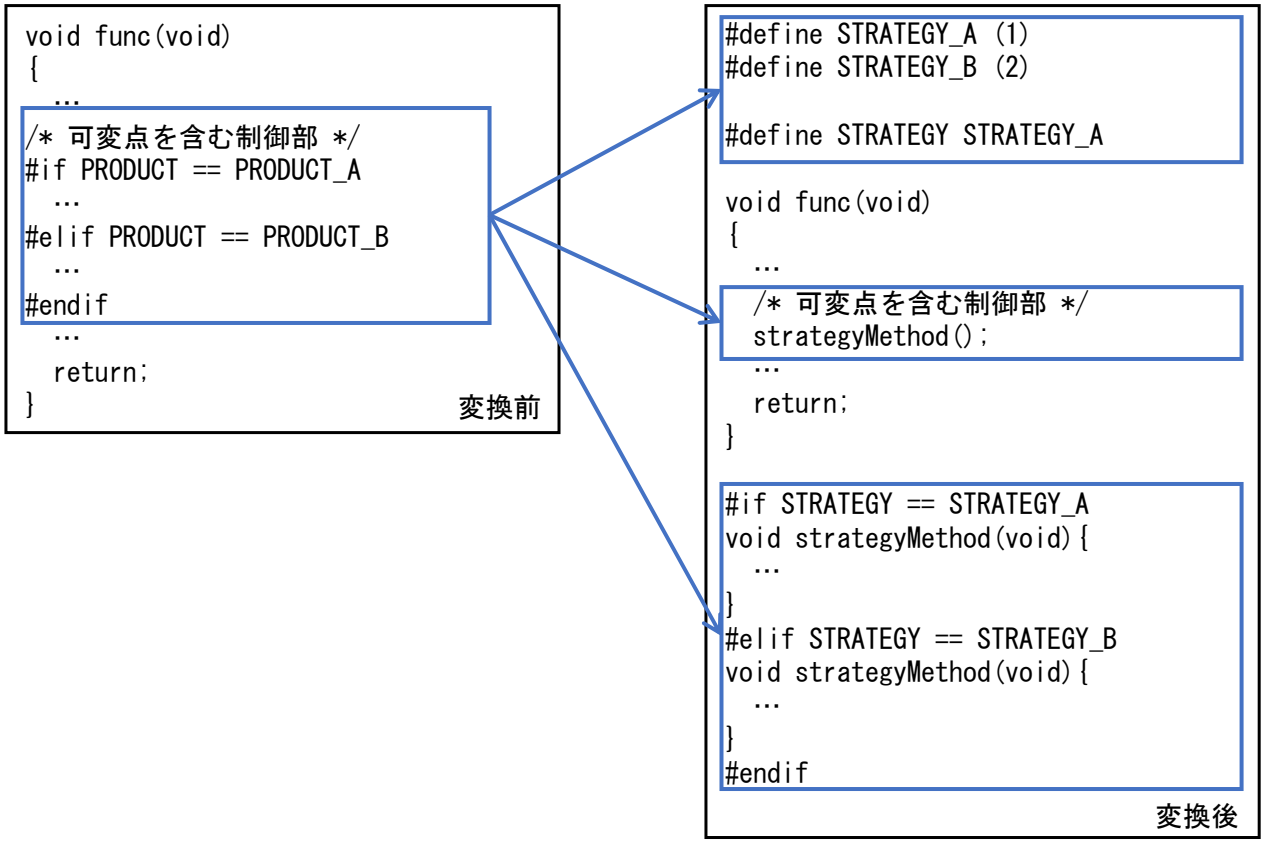

図 9.6 拡張における Strategy を分離するリファクタリングパターン

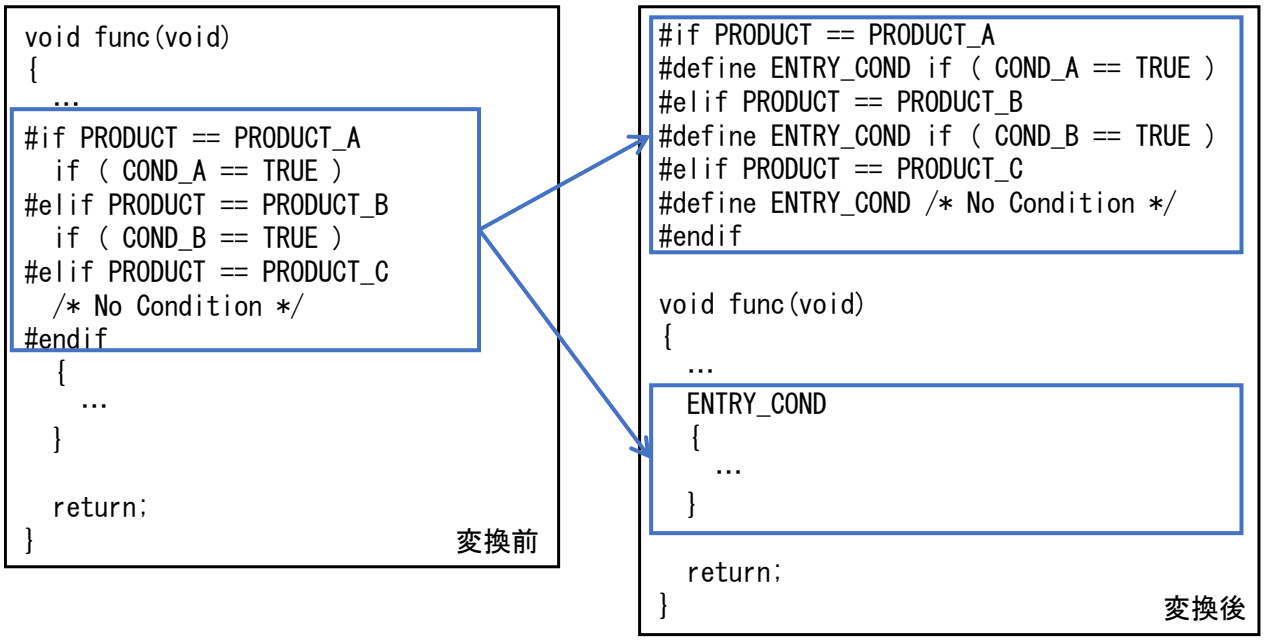

図 9.7 アスペクトを分離するリファクタリングパターン

9.3.2　アプリケーション開発の並行性

アプリケーション開発の並行性を評価するため，適用プロジェクトの実績としての開発スケジュールを作成した．図 9.8 に対象の SPL のコアチームと派生チームの開発スケジュール実績を示す．本開発スケジュールは，リファクタリング対象とした SPL のみ抽出している．コアチームは，この期間単一の SPL をコア資産として進化させながら，3 つのコア製品 A，B，C を開発し，インクリメンタルにリリースしている．派生チームはその他の SPL の開発も続けながら，対象の SPL のリファクタリングを実行し，派生製品 D，E を開発してリリースしている．

9.3.2.1　開発スケジュールの並行性

開発スケジュールの並行性の観点で評価を加える．対象の SPL の開発は，7 月末までがコア資産とコア製品 A の開発が主軸となっている．その後 8 月～10 月において，3 本のコア製品の開発が並行し，9 月～10 月では，派生製品として 2 本，両製品で最大 5 本の製品開発が並行している．

コア製品開発の並行化は，あらかじめコアチームが 3 製品を対象にコア資産を開発することで成立させている．従来の SPLE の開発方法といえる[39]．派生製品開発の並行化は，コア資産上に設計されていない変異体の追加も開発しながら同時並行的に開発することができている．

コア製品の開発が進行すると，可変性の開発は小さく，共通性の開発の洗練が中心となってくる．そのため，派生製品において可変部の開発が並行実施されても，開発ビューの構成管理上で分離されていれば，並行開発が阻害されないことが確認できた．

9.3.2.2　開発組織の並行性

本適用では，コアチームと派生チームと 2 つの組織体が，アプリケーション開発を同時並行実施することができている．また，4 月～7 月の期間においては，コア製品の開発をコアチームが，アーキテクチャのリファクタリングを派生チームが実行している．アーキテクチャのリファクタリング方法を確立し，ガイドラインとして示したことで，コアチームは共通性の開発に集中し，派生チームは可変性の開発に集中し，互いに並行かつ分担して開発することに成功した．開発組織の並行性としても有効であることが確認できた．

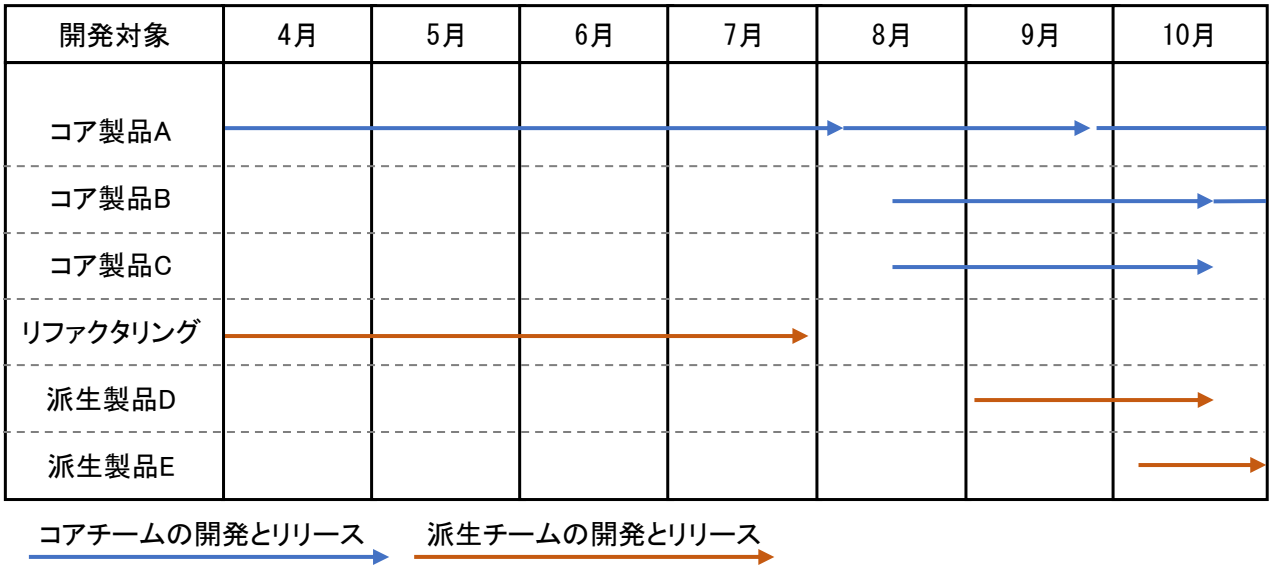

図 9.8 SPL の同時並行開発スケジュール

10 考察

10.1 SPL の管理可能な開発方法論の意義

従来の SPLE で，製品進化と同時並行的に多様な製品提供に対応する俊敏性を備えてソフトウェアを開発するためには次の 3 つの課題がある．

(1) SPL は包括的に管理困難である
(2) 個々の製品開発の安定性のばらつきが大きい
(3) 製品進化と多様な製品開発に順序制約がある

提案した SPL の管理可能な開発方法論は，上記の課題に対してポートフォリオマネジメントをドライバとして開発プロセス，プロダクト可変性，アーキテクチャを構築し直すことにより，SPL を包括的に管理可能な状態へと導く．開発プロセスを構築し直すことで，ポートフォリオをマネジメントするための，開発生産性を測定可能にし，個別の開発量の予測可能性を向上させる．プロダクト可変性を分析してコントロールすることで，個々の製品開発の安定性にばらつきが生じる阻害要因を取り除く．アーキテクチャを構築し直すことで，製品進化と多様な製品開発を同時並行的に開発することを可能とし，順序制約を軽減することで開発の待ちの発生を軽減する．

ポートフォリオマネジメントにより，管理可能かつ管理容易な状態に導くことで，提案した開発方法論により，市場の多様性と俊敏性の要求に対応したソフトウェア開発を実現可能とした．

10.2 SPLE のためのアジャイル開発方法の意義

従来の SPLE[10][16][23][43][44]では，SPL を戦略的に市場に投入するためのポートフォリオ戦略の立案については取り扱っているが，SPL の開発を包括管理する方法論は確立されていない．Business 領域としてポートフォリオ戦略を立案したら，Architecture を戦略に応じて設計し，Process を適合して Organization の責務分担を割り当てる BAPO モデル[20][31]で全体を設計するに留まる．そのため，SPL の開発が戦略に沿って活動できているかの監視や，どこで開発資源が枯渇するリスクが生じ得るのか，現有資源でどの程度顧客の要求に俊敏に対応できるのかといった組織状態の把握が困難である．

ASD ではポートフォリオマネジメントの研究は盛ん[25][29][32][40][47]であるが，学習とフィードバックに基づいているため，SPLE のアプリケーション開発のような短期開発プロジェクトの生産性を把握することを不得手としている．ASD ではフィーチャドリブン開発が中心であり，アーキテクチャやマネジメントを主体とした開発との方法論と統合する研究が少ない．SPLE と統合した APLE の研究分野でも統合方法がエンジニアリング領域の部分に限定されることが多い[11][17][18][37][42]．

提案した開発方法論では，SPLE のアプリケーション開発では，可変点を中心とした反復性の高いプロセスが実行されることに着目し，2 層のイテレーションプロセス構造を備えた SPLE のためのアジャイル開発方法を考案した．これにより，短期開発プロジェクトを集約して SPL を包括した長期開発プロジェクトと見做すことで，ASD における学習とフィードバックによる生産性の把握が可能となった．その結果，ポートフォリオマネジメントの管理性を向上させるための予測可能な開発量と安定性を備えた生産性を実現することが可能となった．アーキテクチャ中心，計画駆動な開発領域においても ASD を開発フレームワークとして統合することで管理性を向上することが可能であることを示した．

10.3　可変性のモデル化と構造分析方法の意義

従来の可変性モデルと分析方法[2][7][14][20][21][22][27][34][38][39]では，内部可変性や外部可変性をモデル化し，製品設計やアーキテクチャ設計の構造に反映させることを目的としている．そのため，可変性をモデル化し構造を分析することは，品質の確保やコストの低減に寄与しても，開発の管理性の向上に寄与するよう活用されていない．

提案した可変性のモデル化と構造分析方法は，分析を通して可変点の開発の順序制約を明らかにする．開発の順序制約が明らかになることで，開発アイテム同士が独立して開発できるものか，依存するために回帰的に検証を求めるものかが判別可能となる．その結果，従来であればアプリケーションに対する要求をすべて分析し，すべての可変点を一度に構成し，テストと検証も一括で実施するシーケンシャルな開発プロセスを分解することができるようになる．順序制約を備えるものの，分割開発可能となった可変点の集合は，開発方法に自由度を与えることができ，アジャイル開発方法と組み合わせることで，検査環境使用の平準化や，バリューストリームの速度向上による開発の俊敏性の向上を実現可能とした．

10.4　SPL のアプリケーション開発の並行開発アーキテクチャの意義

従来の SPLE における可変性への対応[4][8][10][31][39]では，ドメイン開発において可変点をどのように設計すべきかをスコープとしている．そのため，ドメイン開発とアプリケーション開発を並行して開発することが考慮されていない．アーキテクチャ設計のビュー[26][33][41]においても，進化パースペクティブなど，ビューを横断して分析する方法は提案されているが，組織活動としてのプロセスの並行性を考慮する方法が確立されていない．

提案した並行開発アーキテクチャの設計方法は，可変点の実装方法を開発ビューで抽出する．論理ビューと開発ビューで分析し直すことで，異なる製品開発において可変点を開発するためのアクティビティが相互依存しないようソフトウェア構成をリファクタリングする．その結果，組織活動としてのプロセスの並行性の向上を実現している．また，リファクタリング方法をガイドラインとして提供することで，コア資産を開発するチームに限定したアクティビティの制約を軽減できる．大規模アジャイル開発の SAFe フレームワークにおける Architectural Runway[28][29][30]の考え方を応用することで，組織の人的資源を効率的に割り当てることができるようになり，SPL の包括管理の管理性の向上を実現可能とした．

11 今後の課題

本研究を発展させていくために，次の 3 点を今後の課題としてあげる．

11.1 ドメイン開発における俊敏な連携方法の検討

提案した SPL の管理可能な開発方法論は，SPLE のアプリケーション開発を中心に方法論を展開している．これは，SPLE においてドメイン開発よりアプリケーション開発の方が開発プロジェクト数は多く，管理性の低下が発生し易いためである．

システムの高度化，複雑化によって，ドメイン開発に求められる進化の俊敏性も高くなっている．ドメイン開発が常に進化し続ける場合，提案した開発方法論においてドメイン開発とアプリケーション開発の並行開発が可能となっても，管理限界に到達するリスクが生じる．提案方法では，ドメイン開発は基本的に独立先行して断続して進化する開発であり，アプリケーション開発は進化を追従発展させる形を前提としているためである．ドメイン開発が連続して進化し続ける状況において，俊敏に連携するアーキテクチャと開発方法を確立できれば，より多様性と俊敏性の高い開発方法の設計が可能となる．

11.2 スケーラビリティへの対応

提案した SPLE のためのアジャイル開発方法を適用した自動車システムの開発は，開発規模が中規模の開発である．ドメイン開発は 100 人規模，アプリケーション開発は 1 チームで 10 人規模の開発であった．また，システムとしては単一システムにおける開発に対する適用である．システムはより複雑化を進めており，1,000 人規模以上の開発や，システムオブシステムズのように複合的なシステム開発が増加している．

1,000 人規模以上の大規模開発に対しては，アプリケーション開発においても複数チームで 100 人規模の開発に発展することが想定される．その場合，SPL の包括管理もより複雑化するため，大規模アジャイル開発である SAFe[29]などの開発方法などの研究を参考にし，開発チームの包括管理にも併せて取り組む必要がある．

システムオブシステムズにおいては，システム開発間の資源の割り当てに柔軟性が欠けるため，システムズ全体での平準化が困難となる．また，システム間での可変性の管理は，開発ステークホルダーの複雑化と共により困難化する．こうしたスケーラビリティへ対応できれば，多様性と俊敏性だけでなく，柔軟性の高い開発方法の設計が可能となる．

11.3 開発方法論の他分野への適用

本論文では，提案した SPL の管理可能な開発方法論を，自動車システムのソフトウェア開発に適用し，その有効性を実プロジェクトで評価した．本開発方法論をより発展させるために，他分野への適用と評価が必要である．

組込ソフトウェア開発と，エンタープライズ系ソフトウェア開発，Web 系ソフトウェア開発にはその特徴に大きな差がある．組込ソフトウェア開発では，ソフトウェアのハードウェアによる制約が強く，またソフトウェア製品の更新容易性の設計も発展途上である．そのため，SPL の世代交代が完全には果たされず，複数の SPL が生じやすい環境となっている．その他の分野では，ソフトウェアが容易に更新し易かったり，自動テスト環境が構築し易かったりする反面，求められる俊敏性はより高くなることが考えられる．開発環境の異なる他分野の適用を通じて，本開発方法論を発展させれば，より包括的な開発方法論へ発展させ，より多くの分野に貢献することが可能となる．

12まとめ

本研究では，製品進化と多様な製品提供を同時並行的に実現させることを目指して，SPL の開発全体を包括した管理可能な開発を実現させる開発方法論を提案した．提案した開発方法論は，ポートフォリオマネジメントをドライバとして，SPLE における開発プロセス，プロダクト可変性，アーキテクチャを構築し直すことにより，予測可能なソフトウェア製品の開発量と，安定した生産性を実現して，複数の SPL を対象とした開発の管理性を向上する．

SPLE のアプリケーション開発では, 可変点を中心とした反復性の高いプロセスが実行されることに着目し，2 層のイテレーションプロセス構造を備えた SPLE のためのアジャイル開発方法を提案した．可変性のモデル化と構造分析では，分析を通して可変点の開発の順序制約を明らかにし，独立した開発アイテムの集合にプロジェクトを分割する方法を提案した．アプリケーション開発の並行開発アーキテクチャでは，可変性を開発ビューのレベルでモジュール分離する設計ガイドラインを提案した．

提案した SPL の管理可能な開発方法論を自動車システムのソフトウェア開発の実プロジェクトに適用し, その有効性を示した．評価結果から，提案した開発方法論は，多様な製品を俊敏に開発する SPLE に有用であることを確認した．

本論文で提案した開発方法論は，関連研究と照らして，次の点でソフトウェア工学において意義があると考えられる．SPLE において複数の SPL を包括的に管理する方法を提示し，ASD のマネジメント方法と組合せ可能であることを示した．また，ASD を短期開発プロジェクトだけでなく，反復性を備えた類似の複数プロジェクトを対象に適用できることを示した．これは，両開発方法論を融合させた APLE の発展に貢献するものといえる．

謝辞

本研究は，南山大学大学院理工学研究科ソフトウェア工学専攻青山研究室において，青山幹雄教授のご指導の下に実施されたものです．本研究の遂行にあたり，終始手篤いご指導，ご援助を賜りました．青山幹雄教授に心より厚く御礼申し上げます．

本論文の審査をいただきました，南山大学大学院理工学研究科ソフトウェア工学専攻，野呂昌満教授，阿草清滋教授におかれましては，本論文について詳細なアドバイスを頂きました．ここに深甚な謝意を表します．

本論文について共に研究し，提案方法論の議論に参加して下さいました，株式会社デンソーの古畑慶次担当課長には，深く感謝の意を表します．さらに，本研究について新たな開発方法にチャレンジしてくれたプロジェクトの開発メンバの面々に心から感謝します．

参考文献

[1] D. J. Anderson, KANBAN, Blue Hole Press, 2010.

[2] M. Aoyama, An Extended Orthogonal Variability Model for Metadata-Driven Multitenant Cloud Services, Proc. of APSEC 2013, IEEE, Dec. 2013, pp. 339-346.

[3] M. Aoyama, Continuous and Discontinuous Software Evolution, Proc. of IWPSE 2001, ACM, Sep. 2001, pp. 87-90.

[4] S. Apel, et al., Feature-Oriented Software Product Lines, Springer, 2013.

[5] K. Beck, Extreme Programming Explained, Addison-Wesley, 2000.

[6] W. Bekkers, et al., A Framework for Process Improvement in Software Product Management, Proc. of EuroSPI 2010, Springer, Sep. 2010, pp. 1-12.

[7] D. Benavides, et al., Automated Reasoning on Feature Models, Proc. of CAiSE 2005, Springer, Jun. 2005, pp. 491-503.

[8] S. Bühne, et al., Why is it not Sufficient to Model Requirements Variability with Feature Models?, Proc. of AuRE 2004, IEEE, Sep. 2004, pp. 5-12.

[9] M. Cohn, Agile Estimating and Planning, Prentice Hall, 2005.

[10] S. Deelstra, et al., Product Derivation in Software Product Families, J. of Systems and Software, Vol. 74, No. 2, 2005, pp.173-194.

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

[12] C. Ebert and J. Favaro, Automotive Software, IEEE Software, Vol. 34, No. 3, pp. 33-39, May-Jun. 2017.

[13] C. Ebert, The Impacts of Software Product Management, J. of Systems and Software, Vol. 80, No. 6, Jun. 2007, pp. 850-861.

[14] F. R. Frantz, Quality-Aware Analysis in Product Line Engineering with the Orthogonal Variability Model, Software Quality J., Vol. 20, No. 3-4, Sep. 2012, pp. 519-565.

[15] E. Jagroep, et al., Framework for Implementing Product Portfolio Management in Software Business, G. Ruhe and C. Wohlin (eds.), Software Project Management in a Changing World, Springer, 2014, pp. 193-221.

[16] L. G. Jones, Software Process Improvement and product Line Practice: Building on Your Process Improvement Infrastructure, Software Engineering Institute, 2004.

[17] G. K. Hanssen, et al., Process Fusion, J. of Systems and Software, Vol. 81, No. 6, Jun. 2008, pp. 843-854.

[18] P. Hohl, et al., Searching for Common Ground, Proc. of ICSSP 2017, ACM, Jul. 2017, pp. 70-79.

[19] 石川 馨, 新編品質管理入門 B 編, 日科技連出版社, 1966.

[20] K. Kang, et al., Applied Software Product Line Engineering, Auerbach Publications, 2009.

[21] K. Kang, et al., Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[22] K. Kang, et al., Feature-Oriented Product Line Engineering, IEEE Software, Vol. 19, No. 4, Jul.-Aug. 2002, pp. 58-65.

[23] S. Kato, et al., Variation Management for Software Product Lines with Cumulative Coverage of Feature Interactions, Proc. of SPLC 2011, IEEE, Aug. 2011, pp. 140-149.

[24] 小暮 正夫, 工程能力の理論とその応用, 日科技連出版社, 1975.

[25] J. Krebs, Agile Portfolio Management, Microsoft Press, 2008.

[26] P. Kruchten, Architectural Blueprints:The 4+1 View Model of Software Architecture, IEEE Software, Vol. 12, No. 6, Nov. 1995, pp. 42-50.

[27] K. Lee, et al., Concepts and Guidelines of Feature Modeling for Product Line Software Engineering,

Proc. of ICSR-7, Springer, 2002, pp. 62-77.
[28] D. Leffingwell, Agile Software Requirements, Addison-Wesley, 2011.
[29] D. Leffingwell, SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering, Addison-Wesley, 2016.
[30] D. Leffingwell, Scaling Software Agility, Addison-Wesley, 2007.
[31] F. van der Linden, et al., Software Product Family Evaluation, Proc. of SPLC 2004, LNCS Vol. 3154, Springer, Aug.-Sep. 2004, pp. 110-129.
[32] R. C. Martin, Agile Software Development, Prentice Hall PTR, 2003.
[33] M. Matinlassi, Comparison of Software Product Line Architecture Design Methods, Proc. of ICSE 2004, IEEE, May 2004, pp. 127-136.
[34] H. M. Mærsk-Møller, Cardinality-Dependent Variability in Orthogonal Variability Models, Proc. of VaMos'12, ACM, Jan. 2012, pp. 165-172.
[35] A. Metzger, et al., Software Product Line Engineering and Variability Management, Proc. of FOSE 2014 (ICSE 2014), ACM, May-Jun. 2014, pp. 70-84.
[36] 野田 夏子, プロダクトラインの可変性管理, 情報処理, Vol. 50, No. 4, Apr. 2009, pp. 274-279.
[37] M. A. Noor, et al., Agile Product Line Planning, J. of Systems and Software, Vol. 81, No. 6, 2008, pp. 868-882.
[38] O. Oliinyk, et al., Structuring Automotive Product Lines and Feature Models, Requirements Eng. J., Vol. 22, 2017, pp. 105-135.
[39] K. Pohl, et al., Software Product Line Engineering, Springer, 2005.
[40] K. Rautiainen, et al., Supporting Scaling Agile with Portfolio Management, Proc. of HICSS 2011, IEEE, Jan. 2011, pp. 1-10.
[41] N. Rozanski and E. Woods, Software Systems Architecture, 2nd ed., Addison Wesley, 2011.
[42] B. Rumpe, et al., Agile Synchronization between a Software Product Line and its Products, Proc. of Informatik 2015, LNI Vol. 246, Springer, Sep.-Oct. 2015, pp. 1687-1698.
[43] J. Savolainen, et al., Combining Different Product Line Models to Balance Needs of Product Differentiation and Reuse, High Confidence Software Reuse in Large Systems, Proc. of ICSR 2008, LNCS Vol. 5030, Springer, May 2008, pp.116-129.
[44] K. Schmid, Scoping Software Product Lines: An Analysis of an Emerging Technology, Proc. of the 1st Software Product Lines Conference, Kluwer Academic, Aug. 2000, pp.513-532.
[45] K. Schwaber, Scrum Development Process, Business Object Design and Implementation, Springer, Oct. 1995, pp. 117-134.
[46] M. Steger, et al, Introducing PLA at Bosch Gasoline Systems: Experiences and Practices, Proc. of SPLC 2004, LNCS Vol. 3154, Springer, Aug.-Sep. 2004, pp. 34-50.
[47] C. J. Stettina, et al., Agile Portfolio Management: An Empirical Perspective on the Practice in Use, Int'l J. of Project Management, Vol. 33, No. 1, Jan. 2015, pp. 140-152.
[48] B. Tekinerdogan, et al., Supporting Incremental Product Development using Multiple Product Line Architecture, Int'l J. of Knowledge and Systems Science, Vol. 5, No. 4, Oct.-Dec. 2014, pp. 1-16.
[49] C. Tischer, et al., Bosch Gasoline Systems, F. van der Linden, et al. (Eds.), Software Product Line in Action, Springer, 2007, pp. 133-148.
[50] M. Turek, et al., Multi-Project Scrum Methodology for Projects Using Software Product Lines, Proc. of ISAT 2015 - Part III, AISC Vol. 431, Springer, 2016, pp. 189-199.
[51] L. Wozniak, et al., How Automotive Engineering is Taking Product Line Engineering to the Extreme, Proc. of SPLC 2015, ACM, Jul. 2017, pp. 327-336.
[52] J. M. Yuran, Quality Control Handbook, McGraw-Hill, 1951.

研究業績

[1] 林 健吾, コンカレントフィードバック開発方法の車載ソフトウェア開発への適用, SES（ソフトウェアエンジニアリングシンポジウム) 2015 論文集, 情報処理学会, Aug. 2015, pp. 48 - 56 [最優秀論文賞（実践論文部門）, 企業賞, 情報処理学会 2016 年度山下記念研究賞 受賞].

[2] Kengo Hayashi, Mikio Aoyama, and Keiji Kobata, A Concurrent Feedback Development Method and Its Application to Automotive Software Development, Proc. of 22nd Asia-Pacific Software Engineering Conference (APSEC 2015), IEEE Conference Publishing Services, Dec. 2015, Delhi, India, pp. 362-369.

[3] 林 健吾, 青山 幹雄, 古畑 慶次, マルチプロダクトライン開発のための反復型プロセスモデルと管理方法の提案と適用評価, SES (ソフトウェアエンジニアリングシンポジウム) 2016 論文集, 情報処理学会, Sep. 2016, pp. 195-202 [企業賞受賞].

[4] 林 健吾, 青山 幹雄, マルチプロダクトライン開発における可変性の構造分析に基づくアジャイルアプリケーション開発方法の提案と評価, SES (ソフトウェアエンジニアリングシンポジウム) 2017 論文集, 情報処理学会, Aug.-Sep. 2017, pp. 190-197.

[5] 白木 徹, 林 健吾, 青山 幹雄, GUI 開発におけるアーキテクチャドリブンなインクリメンタル開発の提案と自動車 HUD ソフトウェア開発への適用評価, SES (ソフトウェアエンジニアリングシンポジウム) 2017 論文集, 情報処理学会, Aug.-Sep. 2017, pp. 198-203.

[6] Kengo Hayashi, Mikio Aoyama, and Keiji Kobata, Agile Tames Product Line Variability: An Agile Development Method for Multiple Product Lines of Automotive Software Systems, Proc. of 21st International Systems and Software Product Line Conference (SPLC 2017), ACM, Sep. 2017, Sevilla, Spain, pp. 180-189.

[7] 林 健吾, 青山 幹雄, 古畑 慶次, コンカレントフィードバック開発方法の自動車ソフトウェア開発への適用, 情報処理学会論文誌, 15 pages.