プロダクトライン進化に関する調査研究報告書

第 1.2 版（2015 年 5 月 6 日）

平成 26 年度 産学戦略的研究フォーラム

執筆者一覧

丸山 勝久　（立命館大学）
沢田 篤史　（南山大学）
小林 隆志　（東京工業大学）
林 晋平　（東京工業大学）
位野木 万里　（工学院大学）
吉田 則裕　（名古屋大学）
角田 雅照　（近畿大学）

林 千博　（株式会社 とめ研究所）
岩政 幹人　（株式会社 東芝）
今井 健男　（株式会社 東芝）
神代 剛典　（株式会社 東芝）
白石 崇　（東芝ソリューション 株式会社）
長岡 武志　（東芝ソリューション 株式会社）
小川 秀人　（株式会社 日立製作所）
島袋 潤　（株式会社 日立製作所）
加藤 正恭　（株式会社 日立製作所）
三部 良太　（株式会社 日立製作所）
渡邊 結　（株式会社 日立製作所）
大島 敬志　（株式会社 日立製作所）

目 次

1　はじめに

市場や技術が急速に変化するビジネス環境において，顧客や利用者の要求を満たすソフトウェアをいかに迅速に提供できるのかが IT 事業を成功させる鍵である．特に，信頼性の高いソフトウェアを効率的に開発し，IT 事業で利益を創出し続けるためには，既存の資産を最大限に活用することが重要である．しかしながら，実際，さまざまなシステムが，仕様変更への対応の繰り返しの結果として大規模化複雑化し，保守が困難となる状況に直面している．このような状況において，何の戦略を持たないまま既存資産を活用することは非常に困難である．

このような事態を解決する方法の一つとして，従来の新規開発や派生開発から，プロダクトライン開発 [65] への移行が考えられる．ソフトウェアプロダクトライン（以下，SPL: Software Product Line）とは，共通の特性を持ち，特定の市場やミッションのために，共通の再利用資産（reusable asset）から規定された方法で作られる，ソフトウェア中心システムの集合を指す [18, 65].

ここで，SPL もソフトウェアシステムである以上，その進化は避けられない．残念ながら，SPL におけるプロダクト（製品）開発は通常のアプリケーション開発と異なり，SPL の進化をシステム単体の進化を同じように扱うことはできない [16]．このような状況において，実際の開発現場や保守現場で SPL 進化を実践するためには，理解しやすい進化モデルの構築が必須である．

このような観点から，本研究調査報告書では，SPL における進化プロセスのモデルを示すことで，SPL 進化に対する理解の促進を目指す．さらに，その SPL 進化プロセスを支える技術を明確にすることで，SPL 進化の開発現場および保守現場への普及を加速させる．

2　ソフトウェアプロダクトライン

通常，工業製品としてのソフトウェアを提供する際には，類似したソフトウェアを同時に，あるいは，時系列的に開発することが多い．たとえば，同じ製品であっても機能の一部を限定することでグレードを変えて提供したり，同じ製品を異なるプラットフォーム（ハードウェアや OS など）で稼働するようにしたりする [43].

このような状況において，製品開発の生産性を向上させるためには，各製品で共通な部分を特定し，その共通部分を最大限に再利用することが望まれる．共通部分に対して，各製品に個別な特徴を付与することで，最終的な製品を導出する．ここでは，再利用可能な成果物（artifact）のことを SPL 資産あるいはコア資産（core asset）と呼ぶ．SPL 資産は，複数の製品系列に対して適用可能なように，あらかじめ共通部分と可変部分を織り込んで構築される．

このように SPL 開発では，再利用を強く推し進めることで，以下に示すことを目指している．

開発コストの削減　通常，プロダクトを個別に開発する場合，プロダクトを開発するたびに開発コストが増加する．これに対して，SPL 資産を整備することにより，プロダクト群全体で開発コストを低く抑えることが期待できる [65]．ただし，SPL 資産を最初に開発するためのコストが必要になる．この様子を図 1 に示す．

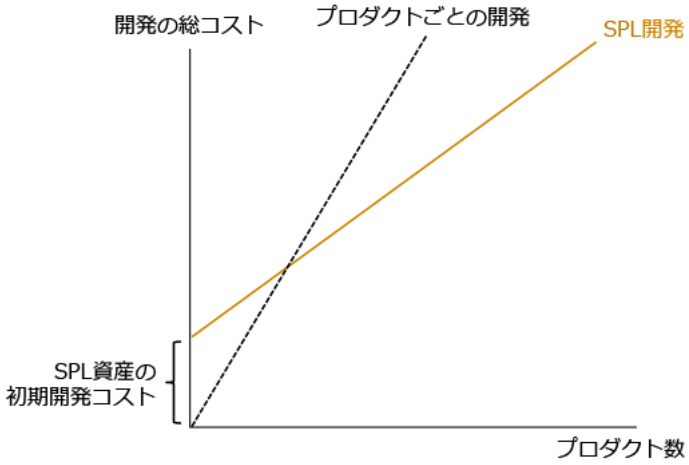

図 1: SPL 開発における開発コストの削減

品質の向上　プロダクトラインにおいて作成される成果物に対するレビューやテストは，単一のプロダクトではなく複数のプロダクトにおいて実施される．このため，不具合が発見されたり，それが修正される機会は増加する．結果的に，SPL で開発されたプロダクト全体の品質は向上する．

出荷までの時間の短縮　再利用なしでプロダクトを毎回開発するやり方では，開発するプロダクトの数が増えても各プロダクトに費やす開発時間はほぼ同じであり，それぞれのプロダクトの出荷までの時間はほぼ一定となる [65]．それに対して，SPL では

SPL 資産を整備する分だけ開発の初期に遅れがでるものの，それを乗り越えた後ではSPL 資産に含まれる共通部品を開発する分の遅れがなくなるため，それ以降のプロダクトの出荷までの時間を短縮することが期待できる．この様子を図 2 に示す．

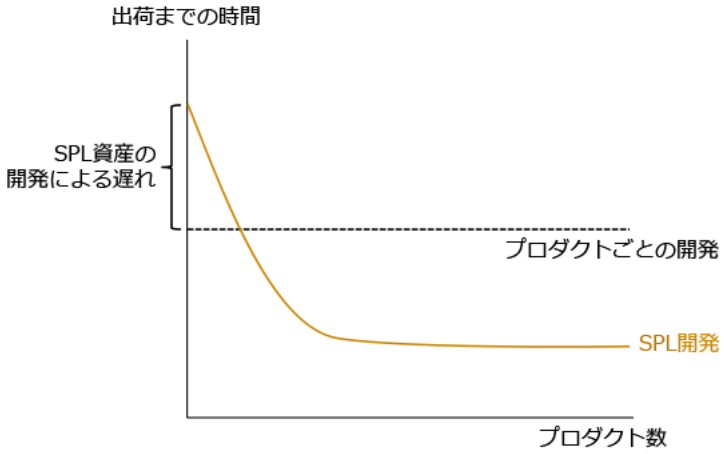

図 2: SPL 開発における出荷までの時間の短縮

次に，SPL 開発の概要を示す．図 3 に示すように，SPL の活動は大きく，ドメインエンジニアリング（domain engineering）とアプリケーションエンジニアリング（application engineering）に分けられる [65]．ここで，文献 [18, 61] では，これら 2 つの活動をそれぞれコア資産開発（core asset development）とプロダクト開発（product development）と呼ぶ．また，これら 2 つの活動を支援する技術的かつ組織的な管理（management）活動も追加している．ただし，本成果報告書では管理活動を対象としない．

ドメインエンジニアリング活動では，SPL 資産の開発を指す．SPL アーキテクトは，SPL に対する要求を獲得し，プロダクトラインアーキテクチャ（SPLA: Software Product Line Architecture）を決定し，再利用可能なコンポーネントを実現する．SPL 資産の開発において重要な作業にスコープの定義（スコーピング）がある．これは，SPL からどのプロダクトを導出するのか，どのプロダクトを導出しないのかの境界を明確にすることを指す．導出される可能性のあるプロダクト系列において可変な部分は，可変モデルとして実装される．可変モデルにおいて，特定のプロダクト系列に固有な部分とすべてのプロダクト系列に共通な部分との分岐点を可変ポイントと呼ぶ．また，バリアント（variant）とは，可変ポイントに対する選択肢を指す．

プロダクトライン開発においてドメインにおける共通性や可変性を分析および定義するための代表的な手法に，フィーチャモデリング（feature modeling）[38, 39] がある．フィーチャ（feature）とは，プロダクトが提供するサービス，操作性，性能など，そのプロダクトに関する代表的かつ観測可能な特徴のことを指す．

フィーチャモデル [71] とは，フィーチャモデリングによって構築された可変モデルを指す．図 4 にフィーチャモデルの例を示す．この例において，すべてのプロダクト（Wiki）は必ずコンテンツタイプ（Content Type）を持ち，それは必ずテキスト（Text）を備える．画像（Image）については，備えていても備えていなくともよい．また，検索機能（Search）を

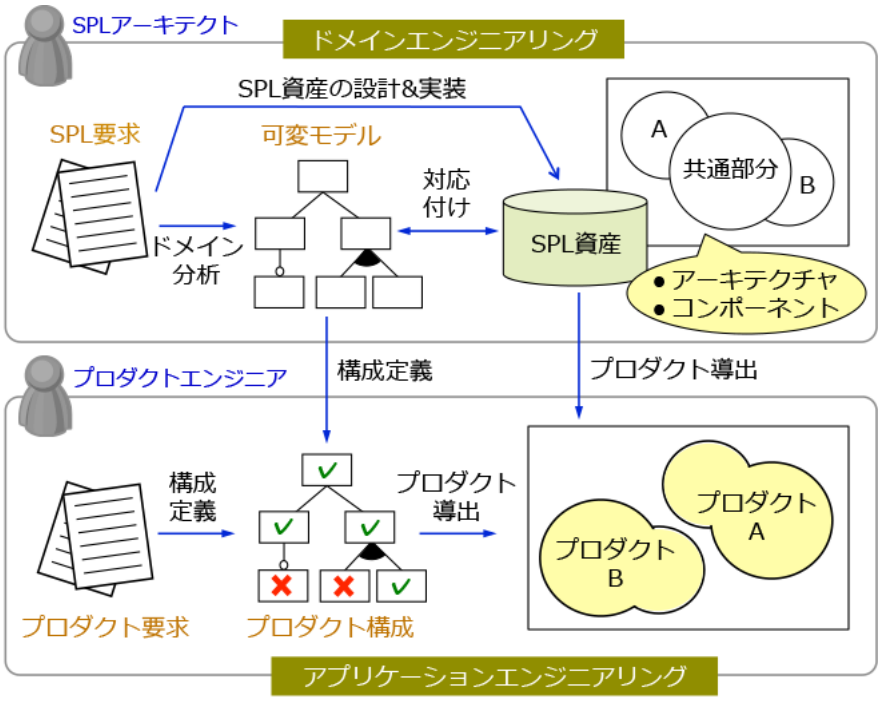

図 3: SPL 開発の概要

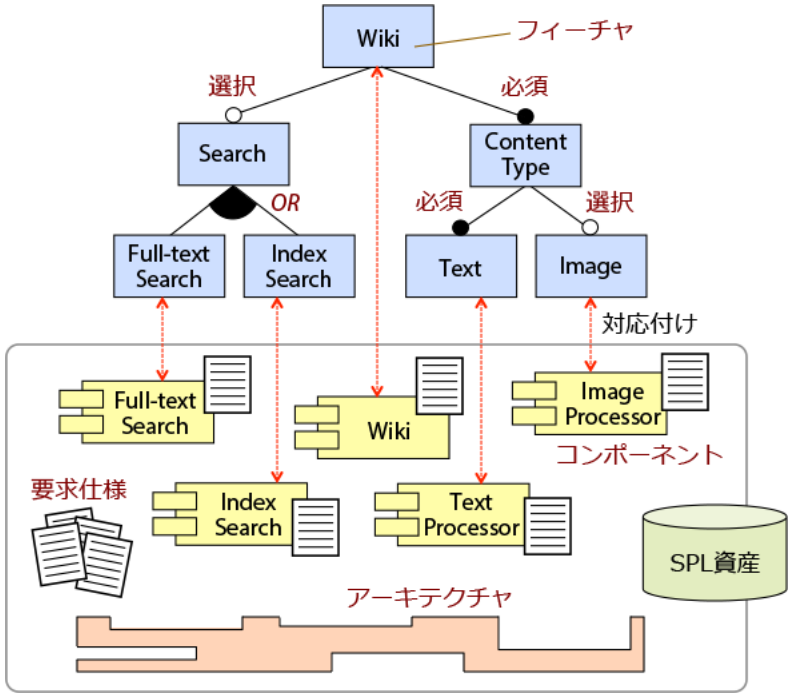

図 4: 可変モデルの例: フィーチャモデルと SPL 資産の対応

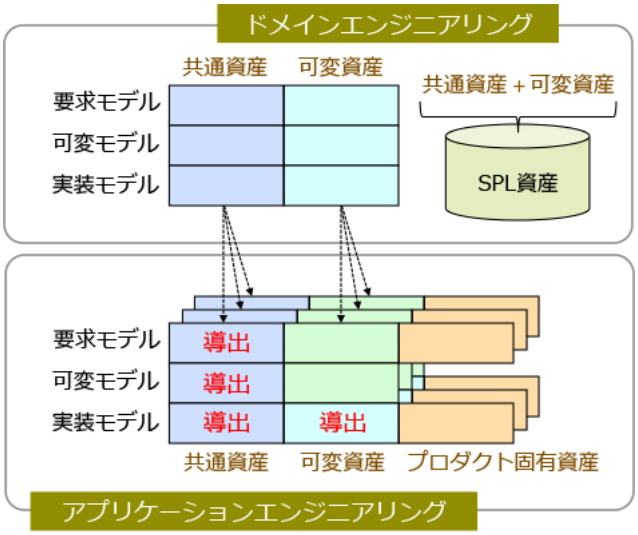

図 5: SPL における資産

持つかどうかは各プロダクトに依存し（検索機能を持たないプロダクトが存在することもあり），それは全文探索（Full-text Search）あるいは見出し検索（Index Search）のどちらかあるいは両方を備えている．

ここでは，単純に 1 つのフィーチャが 1 つの再利用可能コンポーネントに対応している．しかしながら，実際のフィーチャモデルではさまざまな SPL 資産（要求仕様，実装コード，テストケース，アーキテクチャ，ドキュメント）およびそれらを組み合わせたものがフィーチャに対応することがあることに注意する必要がある．

アプリケーションエンジニアリング活動では，プロダクト要求を満たすように設定されたプロダクト構成に基づき，SPL 資産からプロダクトを導出し，各プロダクトで必要となる機能や特性を実現する．可変モデルとしてフィーチャモデルを採用し，フィーチャと SPL 資産やバリアントとの対応がとれている場合には，適切なフィーチャを選択あるいは削除することで，プロダクトの実装を得ることができる．プロダクト要求に対してフィーチャの選択や削除だけで対処できない場合は，プロダクトに特化した実装を追加することになる．

ドメインエンジニアリング活動とアプリケーションエンジニアリング活動に現れる資産を図 5 に示す [71, 16]．SPL およびプロダクトにはそれぞれ異なる抽象度（要求モデル，可変モデル，実装モデル）の成果物が存在する．また，それぞれの成果物は，次の 3 種類に分類できる．

共通資産 すべてのプロダクトに共通する資産（common asset）を指す．これは，ドメインエンジニアリング活動において，あらかじめ SPL 資産として定義する．アプリケーションエンジニアリング活動では，要求モデル，可変モデル，実装モデルのすべてが SPL 資産から直接導出される．

可変資産 個々のプロダクト構成に依存する資産（variable asset）を指す．共通資産と同様に，ドメインエンジニアリング活動において，あらかじめ SPL 資産として定義する．

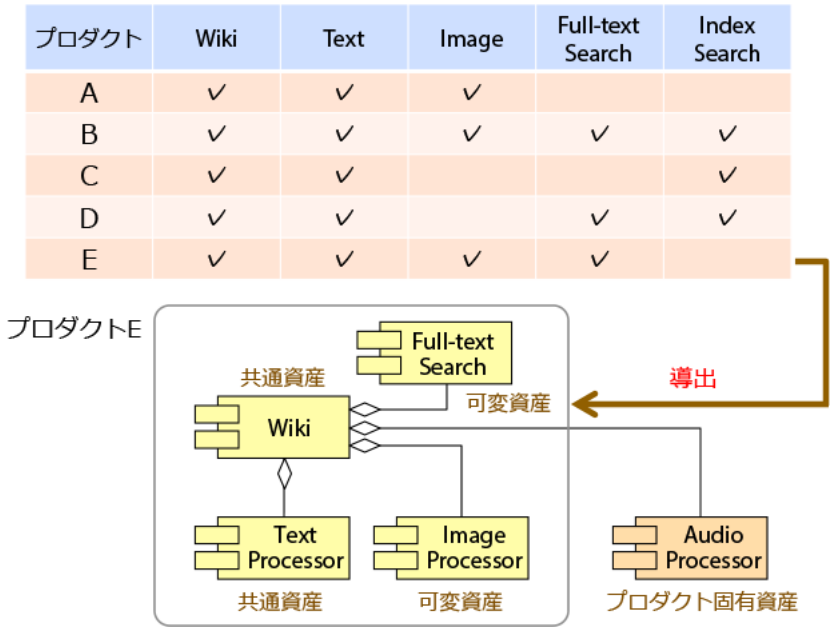

図 6: フィーチャの選択とプロダクトの開発

アプリケーションエンジニアリング活動では，個々のプロダクトに対する要求モデルに基づき，可変モデルにおいて適切なバリアントを選択する．個々のプロダクトに対するバリアントが選択されると，バリアントに対応する実装モデルが SPL 資産から導出される．

プロダクト固有資産 SPL 資産に存在しない機能や特性を実現するために，個々のプロダクトに応じて用意する資産（product-specific asset）を指す．アプリケーションエンジニアリング活動において，要求モデルに基づき実装モデルを作成する．また，要求モデルの定義や実装モデルの作成はプロダクトごとに実施されるため，プロダクトに対する可変性を考慮する必要はない．

図 6 に，アプリケーションエンジニアリング活動において，プロダクトを開発する例を示す．まず，フィーチャを選択することで SPL 資産から共通資産と可変資産を導出する．Wiki および Text Processor は共通資産として導出されたコンポーネント（要求モデル，可変モデル，実装モデル），Full-text Search および Image Processor は可変資産として導出されたコンポーネント（実装モデル）を指す．これらのコンポーネントは，プロダクトラインアーキテクチャ上において結合される．次に，Audio Processor コンポーネント（要求モデル，実装モデル）を追加することで，プロダクト E を作成する．Audio Processor コンポーネントはアプリケーションエンジニアリング活動におけるプロダクト固有資産となる．

3　ソフトウェアプロダクトラインの進化

本章では，まずソフトウェア進化について述べる．その後，SPL の導入戦略と SPL 進化を概要する．最後に，SPL 進化プロセスのモデルを示し，進化プロセスに関連する技術を説明する．

3.1　ソフトウェア進化

ソフトウェア進化とは，一旦出荷されたソフトウェアに対する変更を受け入れる仕組みや活動を指す [53, 56, 63]．通常，個体（生物など）に対して，それ自体が変化することを進化とは呼ばない．進化とは，互いに似ている一群の個体を指す種（species）に対して，世代を越えて発生する現象である．

ここで，ソフトウェアにおける種を同一の SPL 資産から導出されるプロダクト群で捉えることで，プロダクトの導出は世代を超えて発生していると見なせる [35]．一方，SPL 資産や導出されたプロダクトの保守においては，それらの一部だけを書き直したり入れ替えたりすることで，新たな SPL 資産やプロダクトを作成していることが多い．このように作成された SPL 資産やプロダクトが世代を超えていると見なせるかどうかは開発者や利用者の意識に依存する．

そこで，本成果報告書では，SPL 資産やプロダクトの変更が世代を超えているかどうかに関わらず，進化と捉えることにした．

3.2　ソフトウェアプロダクトラインの導入

SPL の導入（adoption）には，以下の 4 つの戦略がある [70]．

独立型（Independent） プロダクトの開発を始める前に，既存のプロダクトとは独立に，SPL 資産を準備する戦略を指す．最初に大きな投資が必要なため，今後開発するプロダクトに対する一定の見通しが明確な場合に有効である．Pure PL [21]，Proactive [47] とも呼ばれる．

借入型（Leveraged） すでに存在する SPL 実装から活用可能な部分を取り出し，新たな SPL 資産を作成する戦略を指す．活用可能な SPL 実装が容易に見つかり，それに対する深い知識がある場合には有効である．

プロジェクト組込み型（Project-integrating） プロジェクトの遂行時に，プロダクト開発と SPL 資産の開発を同時に行う戦略を指す．Project-integrating PL [21]，Reactive [47] とも呼ばれる．プロダクトの要求が，今後開発するプロダクトに対しても同様に発生すると判断された場合，SPL 資産の更新を行う．SPL 資産をインクリメンタル（incremental）に開発するため，その初期開発コスト（初期投資）を削減することが可能である．

リエンジニアリング型（Reengineering-driven） 既存のプロダクト群を分析し，それらの間の共通性や可変性を識別することで，SPL 資産を抽出する戦略を指す．既存プロダ

クトと同様のプロダクトの作成が今後も見込まれる場合に有効である．Reengineering-enabled PL [21]，Extractive [47] とも呼ばれる．

一般的に，独立型と借入型は革命的な（revolutionary）戦略，プロジェクト組込み型は進化的な（evolutionary）戦略である．また，既存の SPL が存在しない場合のリエンジニアリング型は革命的な戦略，既存の SPL に外部資産を組み入れていくリエンジニアリング型は進化的な戦略である．本成果報告書では，プロジェクト組込み型とリエンジニアリング型を SPL の進化として扱う．

3.3　ソフトウェアプロダクトライン進化に対する動機

図 1 と図 2 において，SPL 開発がプロダクト群全体の開発コストを引き下げる可能性と出荷までの時間を短縮できる可能性を持つことを説明した．

開発コストの削減という観点では，SPL 資産の準備にかかる初期開発コストをいかに削減するかということと，個々のプロダクトの作成にかかる開発コストをいかに削減するかということが重要である．SPL 進化では，独立型戦略により膨大な SPL 資産をあらかじめ準備するのではなく，プロジェクト組み入れ型戦略やリエンジニアリング型戦略により段階的に SPL 資産を増やしていくことで，前者のコスト削減が期待できる [70]．また，SPL 進化により SPL 資産を改善することで，後者のコスト削減も期待できる．たとえば，ある SPL 資産から導出されたプロダクト群に同じフィーチャが何度も固有に実装されている場合を考える．このようなフィーチャを共有化できるように SPL 資産を進化させることにより，次のプロダクト開発において実装コストを抑えることができる．SPL 資産の進化が開発コストを削減させる様子を図 7 に示す．

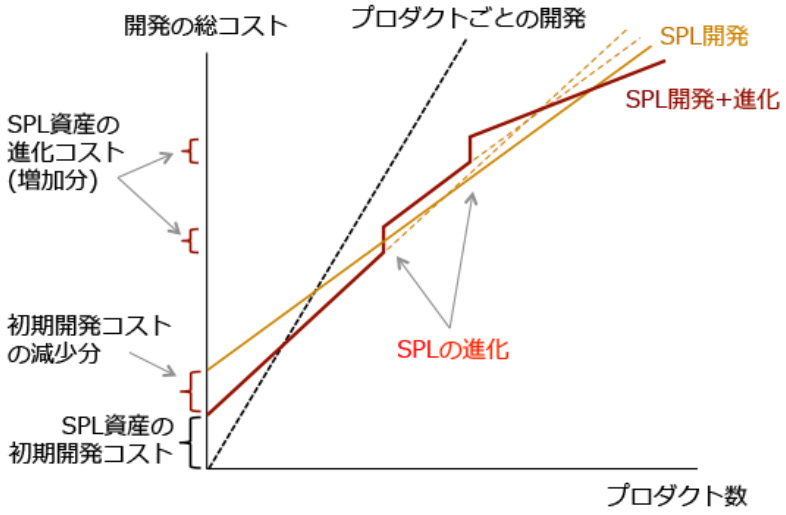

図 7: SPL の進化による開発コストの削減

出荷までの時間の短縮という観点では，SPL 資産の準備による遅れをいかに短縮するかということと，個々のプロダクトの作成にかかる開発時間をいかに削減するかということ

が重要である．SPL 進化では，開発コストの削減と同様に，段階的に SPL 資産を増やしていくことで前者の時間短縮，その SPL 資産を段階的に改善することで後者の時間短縮が期待できる．SPL 資産の進化が出荷までの時間を短縮させる様子を図 8 に示す．

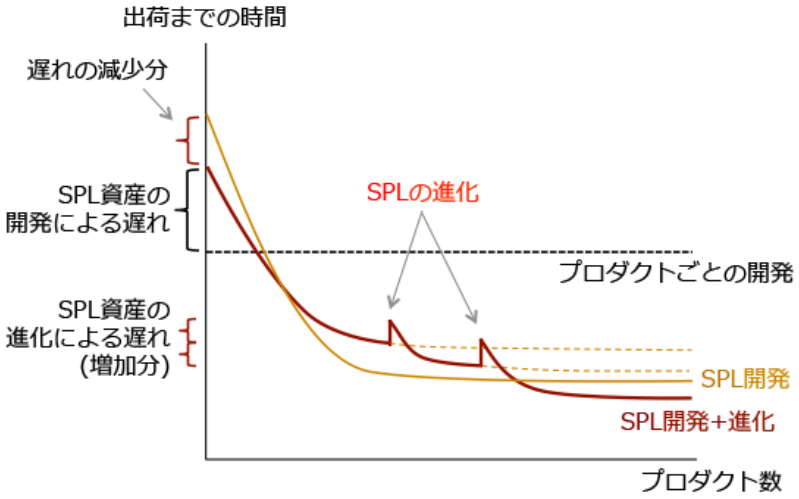

図 8: SPL の進化による出荷までの時間の短縮

3.4　ソフトウェアプロダクトライン進化の特徴

SPL 進化による開発の総コスト削減と出荷までの時間短縮は魅力的である．その反面，Botterweck と Pleuss は，SPL の進化は以下に示す 3 つの課題に直面していることを指摘している [16].

- 長寿命である．さらなる利益を追求するために SPL が長期投資になればなるほど，SPL から導出されるプロダクトも数が多くなる．また，SPL から導出されたプロダクトに対する新規および変更要求に応えるために，SPL は進化しなければならない．結果的に，SPL は単体のプロダクトに比べて，より大きな拡張に向かって，非常に長い期間にわたり，頻繁に進化が強要される．

- 大規模かつ複雑である．SPL はプロダクトの全系列を表現することになり，個々のプロダクトに比べて，その規模はより大きく，より複雑である．一般的に，複数のチームが SPL の開発や保守に関与する．このため，SPL に関する知識は分散し，異なる部分の進化が異なる早さで発生する．

- 相互依存性が強い．SPL における系統的な再利用により，ソフトウェア資産間の相互依存性が強くなる．たとえば，SPL レベルの変化 (SPL 資産におけるバグ修正など) は，その SPL から導出された数多くの個別のプロダクトに影響を与える．さらに，個々のプロダクトレベルにおける新規の要求は SPL 全体に対する変化 (SPL 資産の置き換えなど) を要求する可能性がある．

SPL 開発は通常のプロダクト開発と異なるため，SPL の進化をプロダクト単体の進化を同じように実現することはできない．特に，SPL 開発では，ドメインエンジニアリング活動とアプリケーションエンジニアリング活動が明確に区別され，あらかじめ用意しておいた SPL 資産から要求に応じたプロダクトの作成が行われる．よって，SPL 進化では，SPL 資産の進化とプロダクトの進化を考える必要がある．

我々は，これら 2 つの進化は密接に結びついており，共進化（co-evolution）を引き起こすと考えている．たとえば，ビジネス戦略やビジネス形態の変化により，SPL 資産が変更された場合，それを用いて作成されているプロダクトにも変更が発生する可能性は高い．反対に，変更要求に合わせてプロダクトが書き換えられると，プロダクト群に共通部分が生み出されたり，既存の共通部分が消えたりすることがある．このような状況において，プロダクトを効率的に開発するためには，共通部分の SPL 資産への組み込みや SPL 資産の改訂を行うことは当然である．

SPL 進化における課題を克服し，現実の開発現場や保守現場において SPL 進化を適切に実践していくための第一歩として，SPL 進化におけるプロセスモデルを理解しておくことは重要である．

3.5　ソフトウェアプロダクトライン進化の種類

Schmid と Eichelberger は，SPL 進化における資産の変化や変化による影響を以下のようにまとめている [69]．ここで，SPL 資産とは共通資産と可変資産を指す．

要求レベルの変化 ビジネスゴールの変更などに応じて，ドメインエンジニアリング活動において作成する SPL 資産に対する要求や，アプリケーションエンジニアリング活動において作成する個々のプロダクトに対する要求が変化することを指す．この変化は，図 4 における共通資産，可変資産，プロダクト固有資産に影響を与える．

プロダクトレベルの変化 SPL 資産からプロダクトを導出したり，不要なプロダクトを削除したりすることを指す．一般的に，この変化は，図 4 における共通資産，可変資産，プロダクト固有資産に影響を与えない．ただし，フィーチャインタラクション（feature interaction）[48] による影響のため，SPL 資産からプロダクトを導出する場合に，特定の組み合わせのフィーチャに対応する実装の変更が要求されることがある．

SPL レベルの変化 新規に SPL が構築されたり，不要な SPL が削除されたり，複数の SPL が合併したり，単一の SPL が分割されることを指す．この変化は，SPL の階層化などにより発生する．この変化は，図 4 における共通資産，可変資産，プロダクト固有資産に影響を与える．

ここで，本調査研究では，単一の SPL に関する進化に焦点を当てる．つまり，SPL レベルの変化は対象としない．また，図 4 における共通資産，可変資産，プロダクト固有資産に影響を与える SPL 進化を扱うため，プロダクトレベルの変化も対象としない．結果的に，本調査研究では要求レベルの変化を対象とする．

図 5 における共通資産，可変資産，プロダクト固有資産において発生する要求レベルの変化とそれらの影響を図 9 に示す．資産間の関係において，実線は要求レベルの変化がす

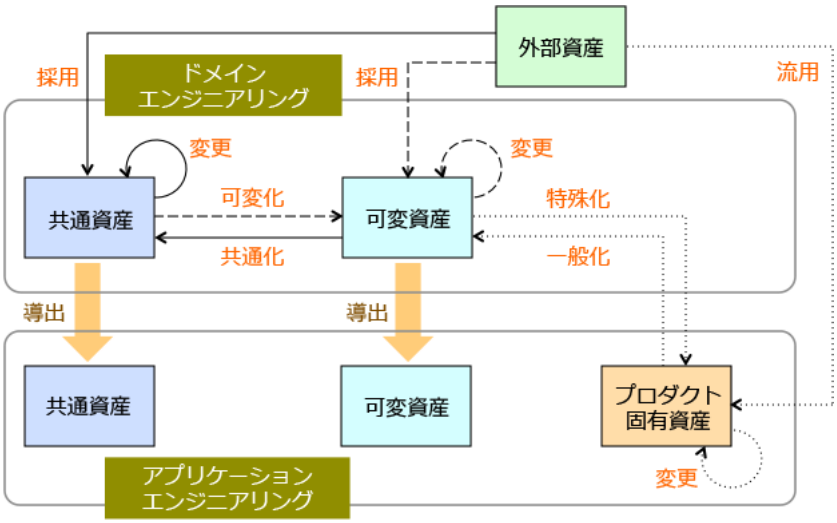

図 9: 資産に対する要求レベルの変化とその影響

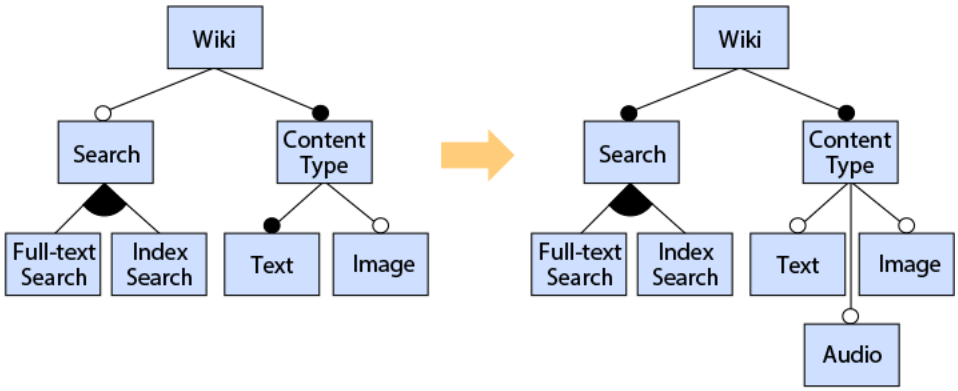

図 10: SPL 進化におけるフィーチャモデルの変更

べてのプロダクトに影響を与えること，破線は要求レベルの変化がいくつかの（複数の）プロダクトに影響を与えること，点線は要求レベルの変化がそれに関係する個々のプロダクトだけに影響を与えることを指す．

ここで，図 9 に示した資産間の関係を，例を用いて説明する．いま進化において，図 4 のフィーチャモデル（図 10 の矢印の左側）が図 10 の矢印の右側のフィーチャモデルに変更された場合を考える．

図 10 に示すように，Wiki フィーチャと Search フィーチャの関係が「選択」から「必須」に変化した場合，可変資産の一部に「共通化」が適用され，共通資産に変化したと見なせる．この場合，すべてのプロダクトが必ず Search フィーチャを備えることになる．これは，すべてのプロダクトに対して要求レベルの変化が影響することを指す．

これに対して，Content Type フィーチャと Text フィーチャとの関係が「必須」から「選択」に変更された場合を考える．これは，共通資産の一部に「可変化」が発生したと見な

せる．この場合，すべてのプロダクトが Text フィーチャを備えているという前提が取り除かれ，いくつかのプロダクトでは Text フィーチャを持たないことが許されるようになる．この変化による影響は，すべてのプロダクトではなく，一部のプロダクトに限定される．

「共通化」と「可変化」は，共通資産と可変資産の間の関係であるのに対して,「一般化」と「特殊化」は可変資産とプロダクト固有資産の関係である．いま，アプリケーションエンジニアリング活動において Audio 機能を持つプロダクトが作成された場合，Audio 機能に関する要求と実装がプロダクト固有資産として蓄積される．図 10 では，プロダクト固有資産に対して「一般化」を適用することで，Audio フィーチャが Content Type フィーチャの「選択」関係を持つ子要素として追加されていることを表している．これとは反対に,「特殊化」により，可変資産に含まれるフィーチャがプロダクト固有資産に変わる可能性もある．

また，SPL 進化において，それぞれの資産内における要求の変更も発生する．さらには，外部資産の一部を採用したり，流用したりすることもある．

3.6 プロダクトライン進化の計画

プロダクトライン進化の計画とは，次の SPL のバージョンアップに対して何を変化させるのかを決めることであり，事業の成功に本質的な影響を及ぼす意思決定をすることに相当する [16]．プロダクトライン進化の過程では，年次モデル，グレード，仕向け地ごとにさまざまな製品バリエーションを，長期間に渡り維持管理し，関係するステークホルダも変化することが予測される．プロダクトラインの進化を無計画に進めると，さまざまなリスクの発生が考えられる．想定されるリスクとしては，製品開発ロードマップに描かれた製品を開発するために必要な要求がコア資産から漏れる，コア資産や製品開発のリソースが不足することなどがある [68].

プロダクトラインの進化の計画立案では，市場，技術，組織の変化を考慮し，要求の範囲とタイミングを決定し，ステークホルダ間で合意形成する．この計画立案において，開発のスコープ，つまり何を開発すべきかを定める要求定義は重要であり，限られたリソースで開発する場合には，要求の優先度を定義することが必要である [54].

要求の優先度を決めるために，既にさまざまな手法が考案されている．要求の優先度を決めるためのシンプルな方法として，MosCow [32]，要求に 1〜5 などの数値（ポイント）を付与する方法 [66]，100 ドルテスト [49] がある．また，複数の視点を組み合わせて要求を評価する手法として，コスト−バリューアプローチ [41] や，重要度と緊急度による 4 象限方式 [82] などがある．これらの手法は，各要求を複数の軸で定量化し，図面に可視化して評価結果を示し，複数のステークホルダでも直観的に共通理解に到達しやすくする．しかし，プロダクトラインの進化に対する要求の優先度定義では，さまざまなシリーズ製品を多くのステークホルダにより長期間に渡り維持管理する状況にあり，ステークホルダの要求への優先度そのものが変化する可能性もあるため，既存の要求の優先度手法をそのまま活用するだけでは有効ではない.

Rule は，release planning と呼ぶシリーズ製品開発における要求の優先度定義手法を提案している [68]．この手法では，要求を評価する基準，ステークホルダ，各要求への制約，要求間の依存関係などの複数の観点から要求を評価し，要求の範囲を決定する．要求を評価する基

準の例として，顧客満足度（customer satisfaction），顧客不満度（customer dissatisfaction），開発リスク（risk of implementation），顧客の受容リスク（risk of acceptance），財務的価値（financial value），コスト（cost），緊急度（urgency），準備度（readiness），使用頻度（frequency of use），使いやすさ（ease of use）などが提案されている．また，ツール EVOLVE [59] により，要求の優先度を可視化し，優先度の計算の自動化を支援している．

プロダクトライン進化の期間は，単一システムの開発と保守の期間よりも，長期に渡ると考えられる．プロダクトライン進化の過程で，開発の意思決定に関わったステークホルダも変化することで，過去のプロダクトライン開発における意思決定の根拠があいまいになり，その後の意思決定が適切に行われないリスクが高まると考えられる．このようなリスク回避のため，Tang らは，設計に関わる意思決定の根拠を蓄積し共有することを提案している [79]．さらに，Schubanz らは，プロダクトライン進化の計画に関係する情報である，組織のゴール，要求の評価基準，意思決定の根拠そのものも進化すると考え，体系化された手法に基づきプロダクトライン進化を扱うことの重要性を指摘している [72]．

3.7 ソフトウェアプロダクトライン進化のプロセスモデル

ここでは，SPL 進化におけるプロセスモデルを示す．我々は，図 3 に示した SPL 開発の一般的なプロセスに基づき，図 9 に示す資産間の関係を考慮して，進化プロセスのモデルを構築した．モデルの構築においては，文献 [16] を参考にした．図 11 に SPL 進化のプロセスモデルを示す．

ここで，SPL 進化は SPL 開発を内包しており，大きくドメインエンジニアリング活動とアプリケーションエンジニアリング活動に分けられる．実際には，SPL 進化には，プロセスそのものを改善するメソッドエンジニアリング活動も含まれるが，本調査研究では検討していないので省略する．四角形は成果物，楕円形は活動（アクティビティ），六角形は評価による判断を必要とする活動を表す．

ドメインエンジニアリング活動では，SPL 要求を受けてドメイン分析が行われ，可変モデルが作成される．その後，SPL 要求と可変モデルに基づきドメイン実装が行われ，SPL 資産（SPL 実装）が構築される．図 4 に示したように，可変モデルのフィーチャと SPL 実装に含まれる要素との間には対応付けが行われている．

アプリケーションエンジニアリング活動では，プロダクト要求を受けて，プロダクトの構成定義が実施される．これによりプロダクト構成が構築され，それに基づきプロダクトの部分実装の導出が行われる．同時に，SPL 資産（共有資産および可変資産）で未対応であったプロダクト固有部分に対する実装が行われる．この実装は，プロダクト固有資産として蓄積される．最後に，プロダクトの組み立てにおいて，プロダクトの部分実装とプロダクト固有実装が統合され，プロダクトの実装が完成する．

SPL 開発は，一般的にこのようなプロセスにより実施される．我々は，このような SPL 開発において，5 つの進化プロセス（図 11 の $E_1 \sim E_5$）をモデル化した．以下，進化プロセス $E_1 \sim E_5$ をそれぞれ説明する．

進化プロセス E_1 既存の外部資産（プロダクト群）やプロダクト固有資産がある場合，それらを SPL 資産として採用するかどうかを判断し，採用すると判断した場合に実施する SPL 資産の進化を指す．これは，3.2 節で述べたリエンジニアリング型戦略に

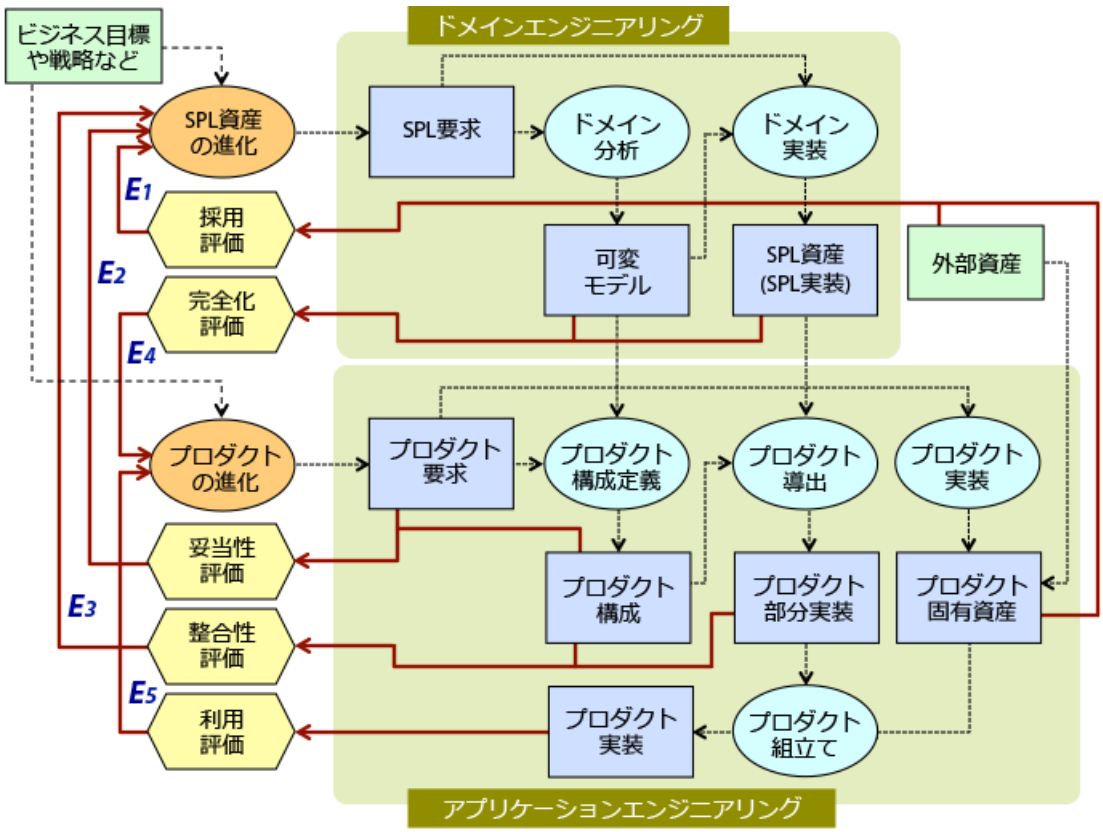

図 11: SPL 進化のプロセスモデル

おける進化に対応する．この進化では，既存の外部資産やプロダクト固有資産における共通性や可変性を識別し，それらを SPL 資産として抽出することが主な活動になる．

この進化プロセスを実現する方法として，RE-PLACE [6], OAR [8], MAP [77, 62], Evolutionary Introduction [76] などがある．また，She らは抽出したフィーチャ間の依存関係に基づき，フィーチャモデルの階層や条件を見つけ出す手法を提案している [75]．さらに，フィーチャ特定（feature location）[10, 83, 24] が，この進化を支援する．フィーチャ特定とは，着目するフィーチャを（直接的に）実装しているコード（の一部）を特定することを指す．

進化プロセス E_2 プロダクトの要求と可変モデルから導出されるプロダクト構成を照らし合わせ，その妥当性（特定のフィーチャをプロダクト固有資産で実装することが妥当であるかどうか）により実施する SPL 資産の進化を指す．これは，3.2 節で述べたプロジェクト組込み型戦略における進化に対応する．いま，新たなプロダクトを開発しようとした場合を考える．その際，まずプロダクト要求がプロダクトに固有なものであるのか，あるいは今後開発されるプロダクトにおいても発生する可能性があるのかを検討する．今後も発生する可能性が高いと判断された場合は，プロダクト

の開発を一旦中止し，プロダクト要求に合わせて SPL の進化を実施する．SPL の進化が完了した後，もとのプロダクトの開発に戻る．今後も発生する可能性が低いと判断された場合は，プロダクトの要求をプロダクト固有資産で吸収する．

この進化プロセスを実現する手法として，PuLSE-Eco [22] や PuLSE-EM [5], Rapid Feedback [29], Issue-based Variability Management [81] などがある．

進化プロセス E_3 プロダクトの構成とプロダクトの部分実装を参照して，それらの間の整合性（一貫性）を検証し，不整合な部分が存在する場合にそれを取り除くための実施される SPL 資産の進化を指す．たとえば，プロダクト構成に現れるフィーチャ間に重複や干渉が発生した場合，それを取り除くように SPL 資産を再構成する．また，プロダクト構成に現れるフィーチャ群とそれらに基づき導出されたプロダクトの部分実装の振る舞いに矛盾が発生する場合，矛盾を取り除くように SPL 資産を修正する作業を指す．

この進化プロセスを実現する方法として，モデル検査に基づく手法 [28, 19] やクローン検出に基づく手法 [55] がある．他にも，文献 [60] では，SPL 進化において利用な形式手法（formal method）（フィーチャモデルの形式化と検証 [7] など）が紹介されている．

進化プロセス E_4 可変モデルや SPL 実装を更新したことで実施するプロダクトの進化を指す．たとえば，稼働中のプロダクトの実装においてプロダクト固有資産で実現していたフィーチャが SPL 資産で提供されるように変更された場合，プロダクトを再開発した方が将来の保守コストの削減が見込める可能性がある．また，稼働中のプロダクト実装では見送られていたフィーチャが SPL 資産で提供されるようになることもある．このような場合，更新後の SPL 資産を利用してプロダクトの再開発が実施される．この進化プロセスは，ISO14764 [34] の完全化保守（perfective maintenance）に対応する．

進化プロセス E_5 稼働中のプロダクトの実装に対する要求の変化や実行環境の変化に応じて実施するプロダクトの進化を指す．この進化プロセスは，ISO14764 [34] の適応保守（adaptive maintenance）や完全化保守に対応する．

ここで，進化プロセス E_1，E_2，E_3 に共通する支援技術にリファクタリング（refactoring）がある．リファクタリングとは，既存ソフトウェアの外部的挙動を保存したままで内部構造を改善することを指す．また，進化プロセス E_2 と E_3 を支援する技術に変更影響分析（change impact analysis）がある．変更影響分析とは，ソフトウェアの一部が変更されたとき，その変更の影響範囲を正確に把握することを指す．さらに，進化を適切に管理するためには，過去に実施された進化を分析する必要がある，このための方法のひとつにソフトウェアリポジトリマイニング（MSR: Mining Software Repository）がある．SPL 進化におけるリファクタリング，変更影響分析，ソフトウェアリポジトリマイニングの活用については，それぞれ 3.9.1 節，3.9.2 節，3.9.3 節で述べる．

我々の SPL 進化プロセスモデルとは別に，Scmid と Verlage は SPL の進化を以下の 3 つに分類している [70].

インフラストラクチャベース（infrastructure-based） プロダクトに対する新規要求に応じて，すぐに SPL 資産の汎化（generalization）を実施する進化を指す．これは，図 11 の進化プロセス E_2 に相当する．

分岐/統合（branch-and-unite） プロダクトに対する新規要求に応じて，すぐに SPL 資産の汎化を実施するのではなく，SPL 資産に対して新たなブランチ（SPL 資産の複製）を作成することで対処する進化を指す．プロダクトの実装が完了した後で，プロダクト固有資産を SPL 資産に取り込む．これは，図 11 の進化プロセス E_1 に相当する．

一括（bulk） プロダクトの実装が完了するごとに SPL 資産への取り込みを実施するのではなく，ある程度の量のプロダクト固有資産が蓄積された後で，一括してプロダクト固有資産を SPL 資産に取り込む進化を指す．これは，図 11 の進化プロセス E_1 に相当する．

3.8 進化の例

図 11 に示した進化プロセス $E_1 \sim E_5$ をまとめたものを図 12 に示す．E_1 と E_2 はプロダクト進化により発生する SPL 進化であることが分かる．また，E_4 は SPL 進化により発生するプロダクト進化であることが分かる．E_3 と E_5 は，それぞれ SPL 進化とプロダクト進化に閉じている．これらの進化は単独で発生することもあれば，連続して発生することもある．

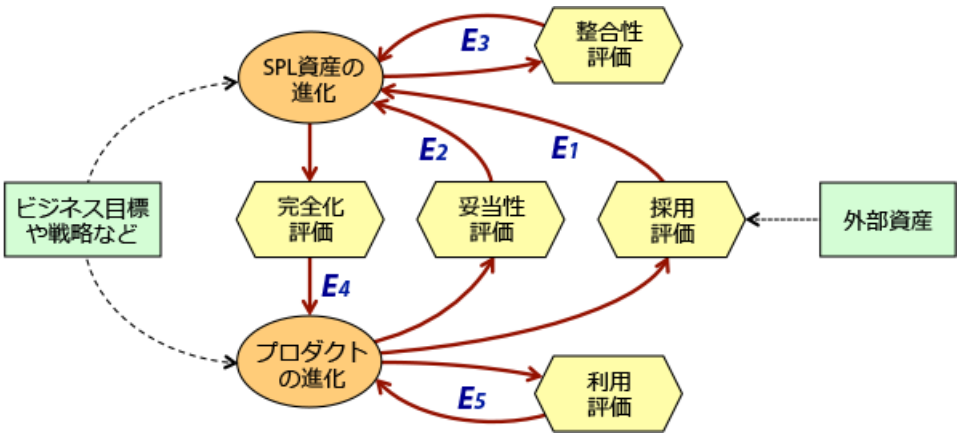

図 12: SPL 進化における進化プロセス

この様子を，図 13 に示す例を用いて説明する．

(1) 新たなビジネス要求が発生したため，SPL 実装 ca3 を利用してプロダクトの開発を開始する．プロダクト構成 pa3-1 を導出し，それとプロダクト要求に対して妥当性評価を行う．妥当性評価の結果により，進化プロセス E_2 が実行される（SPL 実装 ca4 が作成される）．SPL 進化が完了した後にプロダクト開発を再開し，プロダクト pa4-1 を作成する．

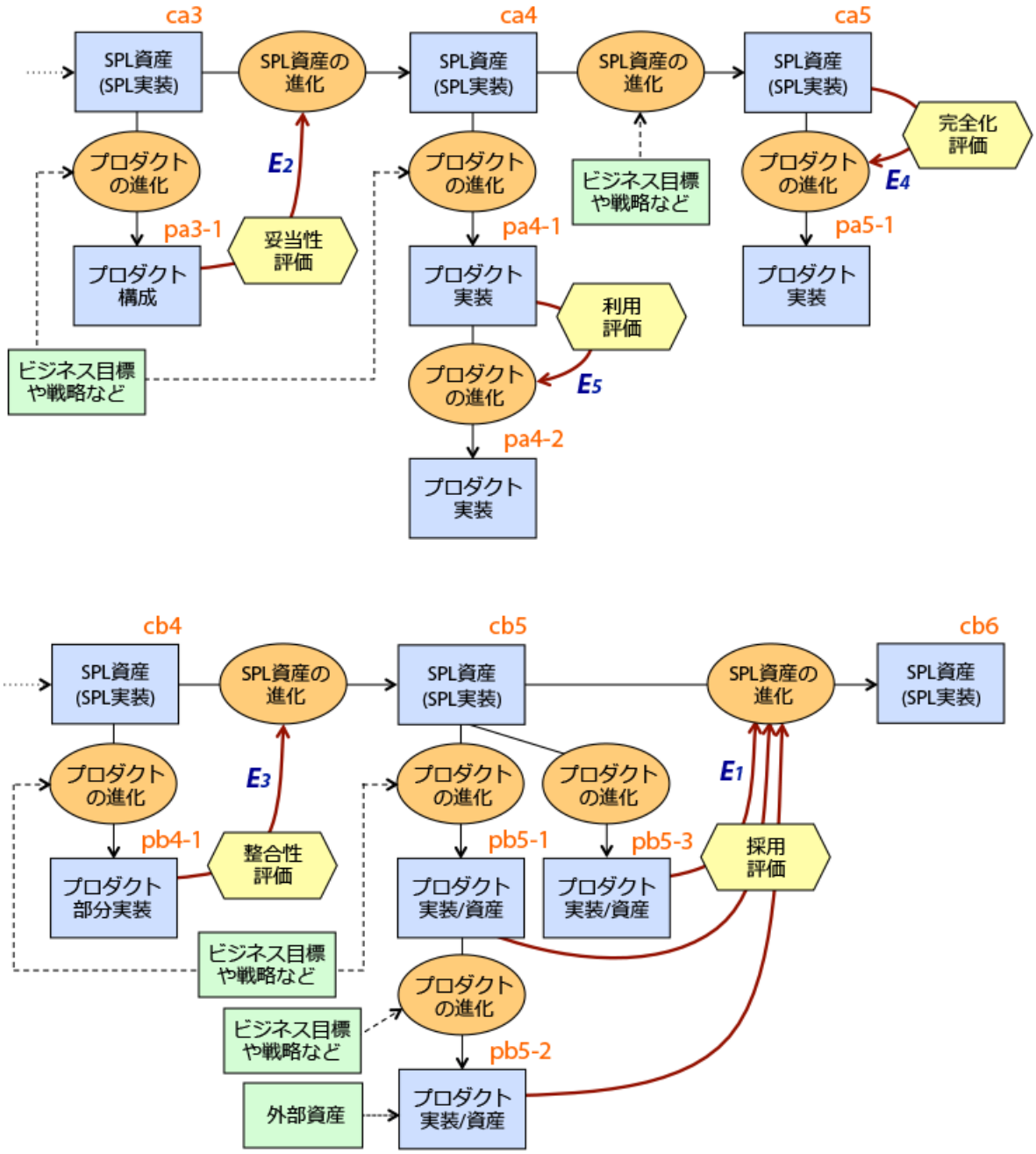

図 13: SPL 進化の例

(2) このプロダクト pa4-1 に対して利用評価が行われ，その結果により進化プロセス E_5 が実行される（プロダクト pa4-2 が作成される）.

(3) ビジネス戦略の変更により，SPL 実装 ca4 への進化が実施され，SPL 実装 ca5 が作成されている．この SPL 進化に対して完全化評価が行われ，その結果により進化プロセス E_4 が実行される（プロダクト pa5-1 が作成される）.

(4) 新たなビジネス要求が発生したため，SPL 実装 cb4 を利用してプロダクトの開発を開始する．プロダクト部分実装 pb4-1 を導出し，それとプロダクト構成に対して整合性評価を行う．整合性評価の結果により，進化プロセス E_3 が実行される（SPL 実装 cb5 が作成される）．SPL 進化が完了した後にプロダクト開発を再開し，プロダクト pb5-1 を作成する．

(5) 新たなビジネス要求により，プロダクト pb5-1 に対して進化が実施され，プロダクト pb5-2 が作成される．また，要求によっては，SPL 実装 cb5 から別のプロダクト pb5-3 が作成される．このようなプロダクトの開発により，プロダクト固有資産が蓄積される．その結果として SPL 実装 cb5 に対して進化プロセス E_1 が実行される（SPL 実装 cb6 が作成される）．

SPL 進化の事例は，文献 [78, 52, 23, 40, 42, 33] などで報告されている．また，SPL の進化にはさまざまな成果物が関係するため，それを実現するためのツール支援は必須である．たとえば，EvoFM [64] では，フィーチャモデルにおいて同時に追加あるいは削除される可能性のあるフィーチャ集合をフラグメントとしてまとめる．フラグメントに対する変更とフラグメント内の変更を明確に区別することでフィーチャの変更が扱いやすくなり，SPL の進化計画の容易化が期待できる．Botterweck らは，EvoFM に基づくツールを提案されている [15, 14]．また，Heider らは，過去のプロダクト構成定義を記録しておくことで，プロダクトを自動導出するツールを提案されている [30]．このツールでは，SPL 資産が更新された場合に，過去のプロダクト構成定義を可変モデルに適用し，新たな版の SPL 資産から新たな版のプロダクトを導出する．

3.9 ソフトウェアプロダクトラインの進化を支える技術

ここでは，SPL 進化を支える技術として，リファクタリング，変更影響分析，ソフトウェアリポジトリマイニングについて概説する．

3.9.1 リファクタリング

リファクタリングとは，既存ソフトウェアの理解性や変更容易性を向上させることを目的とした上で，外部的挙動を保存したままで内部構造を改善することを指す [25]．一般的には，リファクタリング前後で，同一の入力値集合（外部入力）に対して同一の出力値集合（外部出力）が得られることを，外部的挙動が保存されているという．また，リファクタリングでは，大きな設計変更を小さな変換の繰り返しで実現することで，挙動の保存を保証するのが特徴である．

SPL の進化では，既存プロダクトから SPL 資産を構築するためのリファクタリング（SPL 資産を抽出したり SPL 資産に変形させたりするリファクタリング）と，すでに存在する SPL 資産を再構築するリファクタリングを区別して考える必要がある [16]．

まず，SPL 資産を構築するためのリファクタリングを紹介する．Kolb らは，PuLSE-DSSA [6] において既存のソフトウェアコンポーネントから SPL 実装を作成する際にリファクタリングを適用した事例を報告している [44]．基本リファクタリングとより高度

なリファクタリングを適用することでソースコードを再構成し，SPL 実装の改善に成功している．また，Liu らは，既存アプリケーションのコードを分割するリファクタリング（feature-oriented refactoring）を提案している [51]．このリファクタリングでは，アプリケーションの提供するフィーチャをモジュールの合成として捉え，モジュール関係を再構成することで，もとのアプリケーションの振る舞いを保存したままでフィーチャを抽出する．さらに，Rubin と Chechik は，類似する複数のプロダクトのモデル（UML のステートチャートなど）の性質に基づき，SPL における共通資産の抽出を支援する手法を提案している [67]．この手法では，プロダクトのモデルを合成して作成したモデルが，もとのプロダクトのモデルの性質を変えない（合成したステートチャートがもとのステートチャートにおける状態遷移をすべて満たしている）場合をリファクタリングと見なしている．

これらのリファクタリングは主に進化プロセス E_1 を支援する．ここで，既存のプロダクトから SPL 資産を抽出するためには，既存プロダクトに対する十分な理解が必要である．よって，図 11 の進化プロセス E_1 の実施においては，従来のコードリファクタリング [25] も頻繁に利用される．

次に，すでに存在する SPL 資産を再構築するリファクタリングを紹介する．Critchlow らは，プロダクトラインアーキテクチャに対するリファクタリングを提案している [20]．各コンポーネントが提供するサービスと各コンポーネントが要求するサービスに対するメトリクス値を求めることで，プロダクトラインアーキテクチャに関する構造の欠陥を検出する．この欠陥を除去するためのリファクタリングが定義されている．これに対して，Avles らは，フィーチャに着目したリファクタリングを提案されている [2]．このリファクタリングは，変換前後のフィーチャモデルから導出されるプロダクト群の構成が同じであることを保証し，その上でプロダクトの構成可能性（configurability）の改善を目指している．また，Schulze らは，フィーチャ組み合わせにおける可変性を保証するリファクタリングを提案している [73]．さらに，Neves らは，変換前後のフィーチャモデルから導出されるプロダクト群の構成だけでなく，それらを実現するプロダクトの部分実装の同一性も保証する理論体系 [13] に基づくリファクタリングを提案している [58]．

これらのリファクタリングは主に，進化プロセス E_2 と E_3 を支援する．ただし，進化プロセス E_1 においても，既存の SPL 資産に新たな SPL 資産を統合する際には利用できる．

3.9.2 変更影響分析

ソフトウェア進化においては，ソフトウェアの変更は避けられない．通常，ソフトウェアの構成要素に対する変更は直接書き換えた部分だけにとどまらず，他の多くの構成要素にも影響を及ぼす．このような影響に関して，その範囲を正確に把握することを目的とした技術に，変更影響分析（change impact analysis）[4, 12, 50] がある．

変更影響分析は，大きく依存解析（dependency analysis）とトレーサビリティ解析（traceability analysis）に分けられる [12]．前者は，プログラムの構成要素間に現れる依存関係（データ依存関係，制御依存関係，関数の呼び出し関係など）を解析する作業を指す．後者は，ソフトウェア開発工程（要求，設計，実装，テスト，保守）において作成されたあらゆる成果物（要求文書，設計文書，実装コード，テストケース，バグレポート，変更要求チケット）を対象とする [11, 17]．これらの成果物の間に現れる論理的なリンク（logical link）

をあらかじめ識別しておき，変更時にそのリンクをたどることで変更を追跡し，その影響範囲を特定する．

SPL 進化では，ドメインエンジニアリング活動における可変モデルや SPL 資産に対する変更と，アプリケーションエンジニアリング活動におけるプロダクトに対する変更を適切に管理することが要求され，その作業は複雑である．ここでは，SPL を対象としたトレーサビリティ解析を紹介する．Jirapanthong と Zisman は，SPL 開発に現れる 8 種類の文書（フィーチャモデル，サブシステムモデル，プロセスモデル，モジュールモデル，ユースケース，クラス図，ステートチャート，シーケンス図）に対して，9 個のトレーサビリティリンク（Satisfies, Depends-On, Overlaps, Evolves, Implements, Refines, Similar, Different）を定義している [36]．これらのリンクは，あらかじめ用意しておいた規則に基づき自動的に検出する．これに対して，Anquetil らは，より洗練された 4 つの直交するトレーサビリティリンクを定義している [3]．これらのリンクを以下で説明する．

（1） Refinement リンクは，ドメインエンジニアリング活動やアプリケーションエンジニアリング活動に現れる異なる抽象度（要求と設計，設計と実装など）の成果物を関連付ける．

（2） Similar リンクは，ドメインエンジニアリング活動やアプリケーションエンジニアリング活動に現れる同じ抽象度（要求内部や設計内部など）の成果物を関連付ける．

（3） Variability リンクは， Realization と Use に分かれる．Variability /Realization リンクは，ドメインエンジニアリング活動における可変モデルの構成要素（フィーチャなど）とその実装（実現）を関連付ける．一方，Variability/Use は， アプリケーションエンジニアリング活動に現れる要素とドメインエンジニアリング活動に現れる要素を関連付ける．

（4） Versioning リンクは，ドメインエンジニアリング活動やアプリケーションエンジニアリング活動において，連続する 2 つの版間に現れる要素を関連付ける．

SPL における成果物を静的に解析するのでなく，回帰テスト（regression testing）を用いて変更影響分析を行う手法が Heider らによって提案されている [31]．この手法では，SPL 資産が変更された際，その変更前にプロダクトを導出した際のプロダクト構成定義を利用して変更後の SPL 資産からプロダクトを導出する．変更前のプロダクトに対するテストを変更後のプロダクトに適用し，その結果からプロダクト実装に対する SPL 資産の変更影響範囲を特定する．

変更影響範囲の対象をフィーチャモデル（とその実装）に限定した手法もいくつか提案されている．Acher らは，フィーチャモデルのフィーチャとそれらの関係を形式的に表現することで，2 つのフィーチャモデル間の構文的な差分と意味的な差分を定義している [1]．このような差分を定義することで，フィーチャモデルの合成や分割が形式的に適用できる．

Thüm らは，SPL 進化において編集前後のフィーチャモデルから導出可能なプロダクトの集合の関係に基づき，フィーチャモデルの変更を 4 つに分類している [80]．編集前後で導出されるプロダクト集合が変化しない場合を Refactoring, 編集後のプロダクト集合が編集前のプロダクト集合を包含する（プロダクトが追加されているのみの）場合を Generalization, 編集前のプロダクト集合が編集後のプロダクト集合を包含する（プロダクトが削減されて

いるのみ）場合を Specialization と呼ぶ．プロダクト集合の包含関係がこれら 3 つで当てはまらない場合は，任意の編集となる．さらに，Seidl らは，フィーチャモデルとその実装（UML モデル）との対応関係（問題領域と解領域を結ぶリンク）に着目し，フィーチャモデルと SPL 資産の共進化を分類している [74]．フィーチャモデルや SPL 資産の構成要素に対する変更が，それぞれフィーチャモデルや SPL 資産に閉じている場合を Intra-spatial と呼ぶ．一方，構成要素の変更が対応関係にまで影響する場合と，対応関係だけでなく対応先の構成要素にまで影響する場合を Inter-spatial と呼ぶ．

ここで紹介した手法を導入することにより，進化プロセス E_2 と E_3 における変更やその影響範囲が明確になる．これにより，誤った SPL 進化を避けることや適切な SPL 進化を計画することが期待できる．

3.9.3 ソフトウェアリポジトリマイニング

ソフトウェアリポジトリマイニング（以下，MSR と呼ぶ）とは，ソフトウェアリポジトリを分析し，ソフトウェアに関するさまざまな知見を得ようとする研究分野であり，近年非常に活発に取り組まれている．リポジトリとは，Subversion などのソフトウェア構成管理ツール，Bugzilla などのバグ管理システム，メーリングリストアーカイブの 3 つを主に指す [57]．

MSR のアプローチを SPL に適用し，SPL の進化に関する分析を行っている研究は多くないが，いくつかの研究では取り組まれている．Yoshimura らは，SPL の変更履歴に基づいて因子分析を行うことにより，SPL の可変部分の特定を行う方法を提案している [84]．この方法により可変部分を特定することにより，リファクタリングを適用して SPL の再構成を行うことができる．

Krishnan らは，統合開発環境である Eclipse を SPL とみなし，SPL が進化していっても欠陥が発生しにくいモジュールを，リビジョン数やリファクタリングの回数に基づき特定できることを示している [46]．さらに，Krishnan らは，Eclipse の進化にともない，各コンポーネントの信頼性がどのように変化するかを分析している [45]．コンポーネントは，いくつかの観点，たとえば多くのプロダクトで利用されているかどうか，頻繁に変更されているかどうかで分類され，深刻なバグの割合が時系列でどう変化するかなどが分析されている．これらの分析手法はどちらも SPL 再構成時の品質管理に適用可能である．

MSR のアプローチを SPL ではなくソフトウェア進化に適用した研究は多く存在しており，これらの研究の手法を SPL に適用することにより，SPL の進化を補助的にではあるが支援することは可能である．Kagdi らは，ソフトウェア進化に関する MSR の既存研究を進化の分析手法ごとに紹介している [37]．ここでのソフトウェア進化とは，時系列やバージョン間でのソフトウェアの変化を指す．以下，紹介されている分析方法とそれを用いている既存研究の一部を説明する．

メタデータ分析とは，構成管理ツールのコミットログやバグ管理システムなどから得られるデータ（誰が，いつ，なぜその変更を加えたかなど）を分析することを指す．たとえば，Gall らは，通信のスイッチングシステムのリリース履歴より，モジュール間のロジカルカップリング（logical coupling）を発見する方法を提案している [26]．ロジカルカップリングとは，一方のモジュールを変更した場合，他方のモジュールも変更する必要が生じるような関係が存在することを指す．この手法は進化プロセス E_3 を支援できる．また，静

的コード分析とは，ここでは単一のバージョン内でソースコードの分析を行うことを指す．たとえば，Görg と Weißgerber は，不完全なリファクタリングを検出する方法を提案している [27]．不完全なリファクタリングとは，メソッド名などのリネームが，サブクラスなどで行われていないことを指す（正しくは，サブクラスでもメソッド名をリネームする必要がある）．この手法は進化プロセス E_1，E_2，E_3 においてリファクタリングが発生する場合に有効である．

ソフトウェアメトリクスはソフトウェアの規模や複雑度などを定量的に計測したものである．Bieman らは，クラスの継承の深さやクラスの属性の数などのソフトウェアメトリクスに基づき，頻繁に変更される可能性の高いクラスを特定する手法を提案している [9]．この手法によりプロダクトに含まれる特定のクラスが頻繁に変更されやすいと判断された場合，SPL 資産の再構成，すなわち進化プロセス E_3 の実行を検討するとよい．

4　おわりに

ソフトウェアプロダクトライン（SPL）の導入において，その進化は避けられない．このような状況において，SPL 進化を実践するためには，理解しやすい進化モデルの構築が必須である．このような観点から，本研究調査報告書では，SPL 進化を概説し，そのプロセスモデルを示した．さらに，SPL 進化を支援するリファクタリング，変更影響分析，ソフトウェアリポジトリマイニングを取り上げ，これらの技術を SPL 進化において活用している文献を紹介した．

本成果報告書の内容が，実際のソフトウェア開発現場および保守現場における SPL 進化の実践に貢献することを望む．

謝辞

本成果報告書を執筆するにあたり,「プロダクトライン進化に関する調査研究」の機会を頂きました産学戦略研究フォーラム (SSR: Joint Forum for Strategic Software Research) および SSR 会員企業の皆様に深く感謝いたします.

参考文献

[1] Acher, M., Heymans, P., Collet, P., Quinton, C., Lahire, P., and Merle, P.: Feature model differences, *Proc. Advanced Information Systems Engineering (CAiSE'12)*, 2012, pp. 629–645.

[2] Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., and Lucena, C.: Refactoring Product Lines, *Proc. International Conference on Generative Programming and Component Engineering (GPCE'06)*, 2006, pp. 201–210.

[3] Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J.-C., Rummler, A., and Sousa, A.: A Model-driven Traceability Framework for Software Product Lines, *Software Systems Modeling*, Vol. 9, No. 4(2010), pp. 427–451.

[4] Arnold, R. S. and Bohner, S. A.: Impact Analysis - Towards a Framework for Comparison, *Proc. International Conference on Software Maintenance (ICSM'93)*, 1993, pp. 292–301.

[5] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J.-M.: PuLSE: A Methodology to Develop Software Product Lines, *Proc. Symposium on Software Reusability (SSR'99)*, 1999, pp. 122–131.

[6] Bayer, J., Girard, J.-F., Würthner, M., DeBaud, J.-M., and Apel, M.: Transitioning Legacy Assets to a Product Line Architecture, *Proc. European Software Engineering Conference / International Symposium on Foundations of Software Engineering (ESEC/FSE'99)*, 1999, pp. 446–463.

[7] Benavides, D., Segura, S., and Ruiz-Cortés, A.: Automated Analysis of Feature Models 20 Years Later: A Literature Review, *Information Systems*, Vol. 35, No. 6(2010), pp. 615–636.

[8] Bergey, J., O'Brien, L., and Smith, D.: Options analysis for reengineering (OAR): A method for mining legacy assets, Technical Report CMU/SEI-2001-TN-013, Software Engineering Institute, Carnegie Mellon University, 2001.

[9] Bieman, J. M., Andrews, A. A., and Yang, H. J.: Understanding Change-Proneness in OO Software Through Visualization, *Proc. International Workshop on Program Comprehension (IWPC'03)*, 2003, pp. 44–53.

[10] Biggerstaff, T. J., Mitbander, B. G., and Webster, D.: The Concept Assignment Problem in Program Understanding, *Proc. International Conference on Software Engineering (ICSE'93)*, 1993, pp. 482–498.

[11] Bohner, S. A.: Software Change Impacts - An Evolving Perspective, *Proc. International Conference on Software Maintenance (ICSM'02)*, 2002, pp. 263–271.

[12] Bohner, S. A. and Arnold, R. S.: An Introduction to Software Change Impact Analysis, *Software Change Impact Analysis*, Arnold, R. S. and Bohner, S. A.(eds.), IEEE Computer Society Press, 1996, pp. 1–26.

[13] Borba, P., Teixeira, L., and Gheyi, R.: A Theory of Software Product Line Refinement, *Theoretical Computer Science*, Vol. 455(2012), pp. 2–30.

[14] Botterweck, G., Pleuss, A., Dhungana, D., Polzer, A., and Kowalewski, S.: EvoFM: Feature-driven Planning of Product-line Evolution, *Proc. Workshop on Product Line Approaches in Software Engineering (PLEASE'10)*, 2010, pp. 24–31.

[15] Botterweck, G., Pleuss, A., Polzer, A., and Kowalewski, S.: Towards Feature-driven Planning of Product-line Evolution, *Proc. International Workshop on Feature-Oriented Software Development (FOSD'09)*, 2009, pp. 109–116.

[16] Botterweck, G. and Pleuss, A.: Evolution of Software Product Lines, *Evolving Software Systems*, Mens, T., Serebrenik, A., and Cleve, A.(eds.), Springer-Verlag, 2014, chapter 9, pp. 265–295.

[17] Chen, C.-Y. and Chen, P.-C.: A Holistic Approach to Managing Software Change Impact, *Journal of Systems and Software*, Vol. 82, No. 12(2009), pp. 2051–2067.

[18] Clements, P. and Northrop, L.: *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[19] Cordy, M., Classen, A., Schobbens, P.-Y., Heymans, P., and Legay, A.: Managing Evolution in Software Product Lines: A Model-checking Perspective, *Proc. International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12)*, 2012, pp. 183–191.

[20] Critchlow, M., Dodd, K., Chou, J., and van der Hoek, A.: Refactoring Product Line Architectures, *Proc. International Workshop on Refactoring: Achievements, Challenges, and Effects (REFACE'03)*, 2003.

[21] DeBaud, J.-M. and Girard, J.-F.: The Relation Between the Product Line Development Entry Points and Reengineering, *Proc. International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families (ARES'98)*, 1998, pp. 132–139.

[22] DeBaud, J.-M. and Schmid, K.: A Systematic Approach to Derive the Scope of Software Product Lines, *Proc. International Conference on Software Engineering (ICSE'99)*, 1999, pp. 34–43.

[23] Deelstra, S., Sinnema, M., and Bosch, J.: Product Derivation in Software Product Families: A Case Study, *Journal of Systemes and Software*, Vol. 74, No. 2(2005), pp. 173–194.

[24] Dit, B., Revelle, M., Gethers, M., and Poshyvanyk, D.: Feature location in source code: a taxonomy and survey, *Journal of Software: Evolution and Process*, Vol. 25, No. 1(2013), pp. 53–95.

[25] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, 1999. (訳) 児玉公信, 平澤章, 友野晶夫, 梅沢真史, "リファクタリング - 既存のコードを安全に改善する", オーム社 (2014).

[26] Gall, H., Hajek, K., and Jazayeri, M.: Detection of Logical Coupling Based on Product Release History, *Proc. International Conference on Software Maintenance (ICSM'98)*, 1998, pp. 190–199.

[27] Görg, C. and Weißgerber, P.: Error Detection by Refactoring Reconstruction, *Proc. International Workshop on Mining Software Repositories (MSR'05)*, 2005, pp. 1–5.

[28] Guo, J., Wang, Y., Trinidad, P., and Benavides, D.: Consistency Maintenance for Evolving Feature Models, *Expert Systems with Applications*, Vol. 39, No. 5(2012), pp. 4987–4998.

[29] Heider, W. and Rabiser, R.: Tool Support for Evolution of Product Lines through Rapid Feedback from Application Engineering, *Proc. International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12)*, 2010, pp. 167–170.

[30] Heider, W., Rabiser, R., and Grünbacher, P.: Facilitating the evolution of products in product line engineering by capturing and replaying configuration decisions, *International Journal on Software Tools for Technology Transfer*, Vol. 14, No. 5(2012), pp. 613–630.

[31] Heider, W., Rabiser, R., Grünbacher, P., and Lettner, D.: Using Regression Testing to Analyze the Impact of Changes to Variability Models on Products, *Proc. International Software Product Line Conference (SPLC'12)*, 2012, pp. 196–205.

[32] IIBA: International Institute of Business Analysis: A Guide to the Business Analysis Body of Knowledge (BABOK Guide) Version 2.0, Technical report, 2009.

[33] Inoki, M., Kitagawa, T., and Honiden, S.: Application of requirements prioritization decision rules in software product line evolution, *Proc. International Workshop on Requirements Prioritization and Communication (RePriCo'14)*, 2014, pp. 1–10.

[34] International Standards Organization (ISO): Software Engineering - Software Life Cycle Processes - Maintenance, 2006. ISO/IEC 14764.

[35] Jazayeri, M.: Species Evolve, Individuals Age, *Proc. International Workshop on Principles of Software Evolution (IWPSE'05)*, 2005, pp. 3–12.

[36] Jirapanthong, W. and Zisman, A.: XTraQue: traceability for product line systems, *Software & Systems Modeling*, Vol. 8, No. 1(2009), pp. 117–144.

[37] Kagdi, H., Collard, M. L., and Maletic, J. I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 19, No. 2(2007), pp. 77–131.

[38] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, 1990.

[39] Kang, K. C., Lee, J., and Donohoe, P.: Feature-oriented product line engineering, *IEEE Software*, Vol. 19, No. 4(2002), pp. 58–65.

[40] Kang, K. C., Kim, M., Lee, J., and Kim, B.: Feature-oriented Re-engineering of Legacy Systems into Product Line Assets: A Case Study, *Proc. International Conference on Software Product Lines (SPLC'05)*, 2005, pp. 45–56.

[41] Karlsson, J. and Ryan, K.: A Cost-Value Approach for Prioritizing Requirements, *IEEE Software*, Vol. 14, No. 5(1997), pp. 67–74.

[42] Kato, T., Kawakami, M., Myojin, T., Ogawa, H., Hirono, K., and Hasegawa, T.: Case Study of Applying SPLE to Development of Network Switch Products, *Proc. International Software Product Line Conference (SPLC'13)*, 2013, pp. 198–207.

[43] 岸知二: プロダクトライン開発, CQ 出版, 2011, pp. 189–225.

[44] Kolb, R., Muthig, D., Patzke, T., and Yamauchi, K.: Refactoring a Legacy Component for Reuse in a Software Product Line: A Case Study: Practice Articles, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 18, No. 2(2006), pp. 109–132.

[45] Krishnan, S., Lutz, R. R., and Goševa-Popstojanova, K.: Empirical Evaluation of Reliability Improvement in an Evolving Software Product Line, *Proc. Working Conference on Mining Software Repositories (MSR'11)*, 2011, pp. 103–112.

[46] Krishnan, S., Strasburg, C., Lutz, R. R., and Goševa-Popstojanova, K.: Are Change Metrics Good Predictors for an Evolving Software Product Line?, *Proc. International Conference on Predictive Models in Software Engineering (Promise'11)*, 2011, pp. 7:1–7:10.

[47] Krueger, C.: Eliminating the Adoption Barrier, *IEEE Software*, Vol. 19, No. 4(2002), pp. 29–31.

[48] Lee, K. and Kang, K. C.: Feature dependency analysis for product line component design, *Lecture Notes in Computer Science*, Vol. 3107(2004), pp. 69–85.

[49] Leffingwell, D. and Widrig, D.: *Managing Software Requirements: A Use Case Approach*, Addison-Wesley, 2003.

[50] Lehnert, S.: A Taxonomy for Software Change Impact Analysis, *Proc. International Workshop on Principles of Software Evolution / Annual ERCIM Workshop on Software Evolution (IWPSE-EVOL'11)*, 2011, pp. 41–50.

[51] Liu, J., Batory, D., and Lengauer, C.: Feature Oriented Refactoring of Legacy Applications, *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, 2006, pp. 112–121.

[52] Maccari, A.: Experiences in Assessing Product Family Software Architecture for Evolution, *Proc. International Conference on Software Engineering (ICSE'02)*, 2002, pp. 585–592.

[53] Madhavji, N. H., Fernández-Ramil, J., and Perry, D. E.: *Software Evolution and Feedback: Theory and Practice*, John Wiley & Sons, 2006.

[54] Mead, N.: Requirements Prioritization Introduction, Technical report, Carnegie Mellon University, 2006.

[55] Mende, T., Beckwermert, F., Koschke, R., and Meier, G.: Supporting the Grow-and-Prune Model in Software Product Lines Evolution Using Clone Detection, *Proc. European Conference on Software Maintenance and Reengineering (CSMR'08)*, 2008, pp. 163–172.

[56] Mens, T. and Demeyer, S.: *Software Evolution*, Springer, 2008.

[57] 門田暁人, 伊原彰紀, 松本健一: ソフトウェアリポジトリマイニング, コンピュータソフトウェア, Vol. 30, No. 2(2013), pp. 52–65.

[58] Neves, L., Teixeira, L., Sena, D., Alves, V., Kulezsa, U., and Borba, P.: Investigating the Safe Evolution of Software Product Lines, *Proc. International Conference on Generative Programming and Component Engineering (GPCE'11)*, 2011, pp. 33–42.

[59] Ngo-The, A. and Ruhe, G.: A Systematic Approach for Solving the Wicked Problem of Software Release Planning, *Soft Computing*, Vol. 12, No. 1(2008), pp. 95–108.

[60] 野田夏子, 岸知二: プロダクトライン開発における検証, コンピュータソフトウェア, Vol. 30, No. 3(2013), pp. 3–17.

[61] Northrop, L. M.: SEI's software product line tenets, *IEEE Software*, Vol. 19, No. 4(2002), pp. 32–40.

[62] O'Brien, L. and Smith, D.: MAP and OAR Methods: Techniques for Developing Core Assets for Software Product Lines from Existing Assets, Technical Report

CMU/SEI-2002-TN-007, Software Engineering Institute, Carnegie Mellon University, 2002.

[63] 大森隆行, 丸山勝久, 林晋平, 沢田篤史: ソフトウェア進化研究の分類と動向, コンピュータソフトウェア, Vol. 29, No. 3(2012), pp. 3–28.

[64] Pleuss, A., Botterweck, G., Dhungana, D., Polzer, A., and Kowalewski, S.: Model-driven Support for Product Line Evolution on Feature Level, *Journal of Systems and Software*, Vol. 85(2012), pp. 2261–2274.

[65] Pohl, K., Böckle, G., and van derLinden, F. J.: *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer-Verlag, 2005. (訳) 林好一, 吉村健太郎, 今関剛, "ソフトウェアプロダクトラインエンジニアリング - ソフトウェア製品系列開発の基礎と概念から技法まで", エスアイビーアクセス (2009).

[66] REBOK 企画 WG, 情.: 要求工学知識体系, 近代科学社, 2011.

[67] Rubin, J. and Chechik, M.: Combining Related Products into Product Lines, *Proc. International Conference on Fundamental Approaches to Software Engineering (FASE'12)*, 2012, pp. 285–300.

[68] Ruhe, G.: *Product Release Planning Methods, Tools and Applications*, Auerbach Publications, 2010.

[69] Schmid, K. and Eichelberger, H.: A Requirements-Based Taxonomy of Software Product Line Evolution, *Electronic Communications of the EASST*, Vol. 8(2007).

[70] Schmid, K. and Verlage, M.: The Economic Impact of Product Line Adoption and Evolution, *IEEE Software*, Vol. 19, No. 4(2002), pp. 50–57.

[71] Schobbens, P.-Y., Heymans, P., and Trigaux, J.-C.: Feature Diagrams: A Survey and a Formal Semantics, *Proc. IEEE International Requirements Engineering Conference (RE'06)*, 2006, pp. 136–145.

[72] Schubanz, M., Pleuss, A., Pradhan, L., Botterweck, G., and Thurimella, A. K.: Model-driven Planning and Monitoring of Long-term Software Product Line Evolution, *Proc. International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'13)*, 2013, pp. 18:1–18:5.

[73] Schulze, S., Thüm, T., Kuhlemann, M., and Saake, G.: Variant-preserving Refactoring in Feature-oriented Software Product Lines, *Proc. International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12)*, 2012, pp. 73–81.

[74] Seidl, C., Heidenreich, F., and Aßmann, U.: Co-evolution of Models and Feature Mapping in Software Product Lines, *Proc. International Software Product Line Conference (SPLC'12)*, 2012, pp. 76–85.

[75] She, S., Lotufo, R., Berger, T., Wąsowski, A., and Czarnecki, K.: Reverse Engineering Feature Models, *Proc. International Conference on Software Engineering (ICSE'11)*, 2011, pp. 461–470.

[76] Simon, D. and Eisenbarth, T.: Evolutionary Introduction of Software Product Lines, *Proc. International Conference on Software Product Lines (SPLC'02)*, 2002, pp. 272–282.

[77] Stoermer, C. and O'Brien, L.: MAP - Mining Architectures for Product Line Evaluations, *Proc. Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, 2001, pp. 35–44.

[78] Svahnberg, M. and Bosch, J.: Evolution in Software Product Lines: Two Cases, *Journal of Software Maintenance*, Vol. 11, No. 6(1999), pp. 391–422.

[79] Tang, A., Babar, M. A., Gorton, I., and Han, J.: A Survey of the Use and Documentation of Architecture Design Rationale, *Proc. Working Conference on Software Architecture (WICSA'05)*, 2005, pp. 89–98.

[80] Thüm, T., Batory, D., and Kästner, C.: Reasoning About Edits to Feature Models, *Proc. International Conference on Software Engineering (ICSE'09)*, 2009, pp. 254–264.

[81] Thurimella, A. K. and Bruegge, B.: Issue-based Variability Management, *Information Software Technology*, Vol. 54, No. 9(2012), pp. 933–950.

[82] Wiegers, K.: *Software Requirements*, Microsoft Press, 2003.

[83] Wilde, N., Buckellew, M., Page, H., Rajlich, V., and Pounds, L.: A Comparison of Methods for Locating Features in Legacy Software, *Journal of Systems and Software*, Vol. 65, No. 2(2003), pp. 105–114.

[84] Yoshimura, K., Narisawa, F., Hashimoto, K., and Kikuno, T.: FAVE: Factor Analysis Based Approach for Detecting Product Line Variability from Change History, *Proc. International Working Conference on Mining Software Repositories (MSR'08)*, 2008, pp. 11–18.