# 博 士 論 文

# エンタープライズアジャイル開発に適した ソフトウェアアーキテクチャの研究

D2021SE002 田中 優之

指導教員 野呂 昌満

2022 年 1 月

南山大学大学院 理工学研究科 ソフトウェア工学専攻

---

# A Software Architecture for Large-Scale Agile Development

D2021SE002 Masayuki Tanaka

Supervisor Masami Noro

January 2022

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

# 要約

アジャイル開発は一つのチームでの開発から複数チームで開発を行うエンタープライズアジャイル開発へ進化を進めている．実践時の課題としてチーム間の連携（Inter-Team Coordination）は注目を集める研究課題の一つである．チーム間の連携が注目を集める理由は，エンタープライズアジャイル開発を行う時の人員，作業，知識の依存関係の発生が，開発効率や開発スケジュールへ影響することによる．すなわち，エンタープライズアジャイル開発が円滑に実施できるかはこれら依存関係への対処が鍵となる．

本研究では，1 年間のエンタープライズアジャイル開発の事例検証からチーム間の連携に関する課題を確認し，開発スケジュールへ影響を与えることを確認した．事例検証の結果を踏まえ，チーム間の連携が必要となる作業間の依存関係への対処に注目した．ソフトウェアアーキテクチャを中心に課題へ取り組むことで，作業間の依存関係の分析方法とその対処方法としてソフトウェア開発法を提案した．これまで，エンタープライズアジャイル開発に携わるアーキテクト向けに組織とソフトウェアアーキテクチャに関する研究報告はあるが，作業間の依存関係の分析方法と，その依存関係への対処方法に関する提案はない．組織とアーキテクチャの関係を踏まえた依存関係の分析方法とその対処法としてのソフトウェア開発法は依存関係の問題へ対処することへつながる．

これまで，組織とアーキテクチャの関係を整理し，エンタープライズアジャイル開発に携わるアーキテクト向けのガイドラインが提示され，Scaled-Agile Framework（SAFe）ではアーキテクチャの重要性に関し議論する．他方，チーム間の連携が必要となる作業間の依存関係の分析やその対処方法に関する研究はない．本研究では，作業間の依存関係の分析方法とその対処方法をソフトウェア開発法として提示し，事例検証を用いてその妥当性を示した．提案したソフトウェア開発法を用いることで，作業間の依存関係の発生が抑えられ，チーム間の連携の発生が減少する．すなわち，提案したソフトウェア開発法を用いることでチーム間の連携の発生が軽減され，組織がその構造を大幅に変更することなくアジャイル開発を可能となる．

# Abstract

Agile development was originally targeted at small, co-located development teams. Complex requirements lead to evolving to large-scale agile development, which involves more than two development teams. There are several frameworks for large-scale agile development. Inter-team coordination problem, which is the dependencies, e.g., tasks, resources, knowledge, and technical dependencies, is one of the topics attracted by researchers.

This research reports a one-year case study of large-scale development with LeSS huge, and we identified that inter-team coordination problem affects the development schedule. From our case study, we refer to task dependencies. From the point of view of software architecture, we define an analysis method to analyze task dependencies and a software development method as a solution for the dependencies. A report about architectural tactics for large-scale agile development supports architects, but analysis methods to analyze task dependencies and a concrete method for handling them are needed. Our methods make it possible to analyze the dependencies with the software architecture evaluation method and handle them with the software design process.

The architectural tactics indicates the relationship between an organization and architecture. Scaled-Agile Framework (SAFe) also stresses the importance of the architecture in its framework. However, few research projects discuss analyzing task dependencies and a solution in development. This research provides an analysis method and a concrete software design method for handling the dependencies with a case study. Our methods improve inter-team coordination, leading to evolving to agile development.

# 目次

# 1 はじめに

## 1.1 研究背景

アジャイル開発は小さなチームから複数のチームで多くの機能を並行開発する大規模アジャイル開発（エンタープライズアジャイル開発）へと進化を進めている[12]．1 つのチームでアジャイル開発を行うためのフレームワームとしてスクラム[27][32]は多くの実績を挙げ，それはユニットテスト，継続的なデプロイを支える技術，テスト駆動開発[5]など，短い期間にリリースを繰り返すための技術の進歩に支えられている．ソフトウェア開発はより大規模で複雑な要求[45]に対応する必要がある．デジタライゼーションへの注目もあり，組織がビジネスの変化に対応しながらソフトウェアを提供し続ける開発方法，すなわちエンタープライズアジャイル開発への注目が高まっている[12]．エンタープライズアジャイル開発を支援するためのフレームワークは数多く提案されている[19][20][30]．その実践事例は多数報告されている[10]が，その実践時の課題としてフレームワークの複雑さ，多様な組織へ導入する困難さなど課題は多い[12][24][39][40]．実践時の課題として，チーム間の連携（Inter-Team Coordination）がある[42]．例えば，複数チームでのアジャイル開発実践時に発生する依存関係の問題はチーム間の連携に関する課題の一つである[31]．依存関係の例として，開発リソースや作業の前後関係，ドメイン知識，利用している技術によるものがある[31]．Bick らはエンタープライズアジャイル開発を行うときの作業の依存関係に着目し，作業間の依存関係に気づけないことがエンタープライズアジャイル開発を実践するときの非効率さへ繋がることを事例検証で示した[31]．Bick らはチームが作業の依存関係に気づけない原因として，作業計画や見積もり誤りなどを挙げた[31]．作業計画や見積もりの誤りなどがエンタープライズアジャイル開発の失敗へ繋がる，というプロジェクトマネジメント観点の事例報告は有用である．他方，チーム間の作業の依存関係はソフトウェアアーキテクチャによるものも考えられる．すなわち，ソフトウェアアーキテクチャは組織の構造の影響を受ける[14][28]ので，エンタープライズアジャイル開発フレームワークが予め定義する組織構造に変更したことが原因となり，チーム間の作業の依存関係が発生する．ソフトウェアアーキテクチャの観点からこの課題に対する取り組みとして，Nord らはエンタープライズアジャイル開発に携わるアーキテクト向けに組織とアーキテクチャの関係をまとめた[28]．

本研究では，エンタープライズアジャイル開発を実践する際の課題，すなわち複数チームでのアジャイル開発を阻害する要因であるチーム間の連携に関する課題に着目する．これまで，フレームワークの実践事例[24]やプロジェクトマネジメントの改善による課題への取り組み[31]は存在するが，ソフトウェアアーキテクチャ観点からのアプローチはない．特にチーム間の連携に関する課題の原因の一つである作業間の依存関係の発生に着目した取り組みはなく，その分析方法と開発時の対処方法が必要である[28]．本研究では作業間の依存関係の発生に着目し，エンタープライズアジャイル開発実践時のチーム間の連携に関する課題へ対処する．エンタープライズアジャイル開発フレームワークを用いた 1 年間の事例検証を通してその課題を確認し[39][41]，ソフトウェアアーキテクチャ観点からこの課題を整理する．エンタープライズアジャ

イル開発に適するアーキテクチャおよびそのソフトウェア開発法を提示することでエンタープライズアジャイル開発の進化へ貢献する.

## 1.2 研究の目的

本論文の研究目的を以下に示す.

(1) エンタープライズアジャイル開発に適したソフトウェアアーキテクチャはどうあるべきか
(2) エンタープライズアジャイル開発に適するソフトウェア開発法はどうあるべきか

研究の目的を踏まえ，以下の研究課題に取り組む.

(1) 変化に対応可能な複数プロダクトを扱うエンタープライズアジャイル開発方法はどうあるべきか

複数チームでプロダクト群を一括して開発することは組織が効率的に開発を行う一つの方法である[11][15]．他方，既存のエンタープライズアジャイル開発フレームワークではプロダクト群を一括して開発する方法を示すことはない. エンタープライズアジャイル開発では，チーム間の連携が課題となる[13][33][41]. プロダクト群を扱うエンタープライズアジャイル開発方法に必要な開発プロセス，作業管理方法はどうあるべきかを研究課題とする.

(2) エンタープライズアジャイル開発に適したソフトウェアアーキテクチャ評価方法の持つべき特性は何か

エンタープライズアジャイル開発では，チーム間の連携の課題への対処，すなわち依存関係への対処が開発を円滑に行うことの鍵である[31][42]．Nord らは組織とアーキテクチャの関係を整理した[28]．これらを踏まえ，作業間の依存関係に着目し課題に取り組む．作業間の依存関係に着目する理由は，アジャイル開発に影響を与える依存関係である人員，知識，作業[9]の中で作業間の依存関係が最も組織とアーキテクチャの評価に影響を与えると考えたことによる.

(3) エンタープライズアジャイル開発に適したソフトウェア設計プロセスはどうあるべきか

組織とアーキテクチャの関係[28]，さらに人員や知識，作業間の依存関係[9]がエンタープライズアジャイル開発を行う時に課題となる. 本論文では特に作業間の依存関係へ着目する. その理由は, 作業間の依存関係の存在が組織とアーキテクチャに最も影響を与えるという考えによる．組織とアーキテクチャの関係，作業間の依存関係へ着目しエンタープライズアジャイル開発に適するソフトウェア設計プロセスに関して検討する.

## 1.3 本論文の構成

本論文の構成は次の通りである.

第 1 章で研究課題を示し, 第 2 章ではエンタープライズアジャイル開発向けフレームワークの実践事例を示す. 第 3 章ではエンタープライズアジャイル開発に適するソフトウェアアーキテクチャ評価方法を述べ, 第 4 章ではエンタープライズアジャイル開発に適するソフトウェア開発法を示す. 第 5 章でこれまでの取り組みをまとめる.

# 2 複数プロダクトのエンタープライズアジャイル開発方法の提案と実践

複数のチームでアジャイル開発を行う時（エンタープライズアジャイル開発），プロダクト群を一括して開発することは組織を拡大しつつ，効率的に開発を行う方法の一つである[11][15]．既存のエンタープライズアジャイル開発フレームワークでは，プロダクト群を一括して開発する方法はない．他方，組織の要求は多様であるので，その要求への対応が必要となる．

本章では複数のチームでプロダクト群を一括してアジャイル開発する方法を提案する．提案方法は既存フレームワークを踏襲し，プロダクト群の作業管理方法，組織構造を新たに定義した．1 年間の事例検証の結果から提案方法の妥当性を示し，プロダクト群を一括開発する新たな方法を示した．他方，チーム間の連携（Inter-Team Coordination）の課題[12][31][42]が発生し，その対処が開発スケジュールへ影響を与えることを事例検証から確認した．

## 2.1 研究の背景

複数の開発チームでアジャイル開発を行うエンタープライズアジャイル開発を実践する開発組織は様々な課題に向き合う．その実践事例は数多く報告されており[10]，エンタープライズアジャイル開発を実現するエンタープライズアジャイル開発フレームワークは進化を続けている．エンタープライズアジャイル開発実践時のチーム間の連携（Inter-Team Coordination）は注目を集める研究課題の一つである[12][31][42]．小さな開発組織やベンチャー企業では 1 つの製品（プロダクト）を 1 つの開発チームで担当していることが多い．大企業では 1 つのプロダクトを複数の開発チームで担当することがあり開発チーム間の作業調整が課題となり，アジャイル開発が本来実現する要求変化への柔軟な対応や短期間に設計，開発，リリースを繰り返すインクリメンタル開発を実践することが困難になる．スクラムなどアジャイル開発のフレームワークを用いアジャイル開発を実践するベンチャー企業は市場や利用者の要求に柔軟に対応したソフトウェアを提供し続けるので，多くの分野で大企業の競合となる．エンタープライズアジャイル開発を実践することは大企業が市場や利用者の要求に柔軟に対応したプロダクトを提供し続けるための方法の一つである．Large-Scale Scrum（LeSS）, LeSS Huge，Scaled-Agile Framework（SAFe）など様々なフレームワークが提案されている．それぞれのフレームワークの実践事例はその組織においてフレームワークが機能したことを示すが，事情の異なる他の組織でも同様に機能するかは不明である．現在の主要なエンタープライズアジャイル開発フレームワークはチーム間の連携の困難さなどを原因とした実践時の課題が存在するが，その対処法として開発組織が組織の事情や要求を考慮しフレームワークを選択することがある[12].

本章では LeSS Huge を拡張した複数プロダクトのエンタープライズアジャイル開発方法の提案と 1 年間の実践結果を報告する．エンタープライズアジャイル開発フレームワークは複数チームでのアジャイル開発を可能とする方法であるが，フレームワーク導入の難しさ[24]やチーム間の連携の困難さ[12][31][42]など課題が存在する．Dingsøyr らはフレームワークを過信することなく，小さい規模で導入を始めることの重要性を述べた[12]．そこで，組織の事情を考慮し新たに開発プロセスフレームワークを設計することはこの課題に対処する方法の一つと考えた．本章では提案方法を用いた 1 年の実践と提案方法の妥当性を示す．さらに，エンタープライズアジャイル開発実践時，すなわち複数チームでのアジャイル開発実践時における課題を事例検証から示す.

## 2.2 研究課題

本章の研究課題を以下に示す.

(1) 変化に対応可能な複数プロダクトを扱うエンタープライズアジャイル開発方法はどうあるべきか

複数チームでプロダクト群を一括して開発することは組織が効率的に開発を行う一つの方法である[11][15]. 他方, 既存のエンタープライズアジャイル開発フレームワークではプロダクト群を一括して開発する方法を示すことはない. エンタープライズアジャイル開発では, チーム間の連携が課題となる[12][31][42]. この課題を踏まえ, プロダクト群を扱うエンタープライズアジャイル開発方法に必要な開発プロセス, 作業管理方法はどうあるべきかを研究課題とする.

(2) 提案方法は実システムに対して有効か

本章では, 既存のフレームワークを拡張し, 複数チームでプロダクト群を一括して開発する開発方法を提案する. エンタープライズアジャイル開発フレームワークは複雑であるので, 導入が困難であることが多い[12][24]. 事例検証を通し, 提案方法が有効であるか検証を行う.

## 2.3 関連研究

### 2.3.1 エンタープライズアジャイル開発の開発プロセスフレームワーク

アジャイル開発は小さな 1 チームが地理的に同じ場所で開発を行う方法から複数のチームで行うアジャイル開発を行うエンタープライズアジャイル開発へ進化をしてきた[12]．1 つの開発チームで行うアジャイル開発を支援するフレームワークとしてスクラムがある．多くの実践事例が報告され，特に短期間でリリースを繰り返す開発プロセスを支援する継続的インテグレーション（Continuous Integration）および継続的デリバリ（Continuous Delivery）を行う環境（CI/CD 環境）が整備されている組織においてその実践は容易となる．スクラムはその開発プロセスを実現するにあたり開発プロセスの理解とその開発プロセスを支える技術の導入も重要である．継続的インテグレーションおよび継続的デリバリは重要な技術の一つである．スクラムの開発プロセスを導入し，開発プロセスを支える技術を利用することでスクラムのの効果は発揮される．

多くのエンタープライズアジャイル開発向けフレームワークはスクラムを参考に複数チームで並行開発を行う方法を定義する．SAFe (Scaled-Agile Framework) [20], DAD (Disciplined Agile Delivery)[2], SoS (Scrum of Scrums)[1] and LeSS (Large-Scale Scrum) and LeSS Huge[19]は主なフレームワークである．毎年多くの事例が報告されている[10]が，共通の課題としてチーム間の連携（Inter-Team Coordination）がある[31][42]．例えば，作業の依存関係の存在，すなわち作業の実施順序が存在する場合が挙げられる．Bick らはこの課題に着目し，エンタープライズアジャイル開発が失敗する要因として作業の依存関係にチームが気づけないことがあることを事例検証で示した[31]．Bick らは作業の依存関係に気づけない原因として作業計画や見積もりの誤りなどを挙げ，依存関係の存在に気づけないことが開発効率に影響を与えることを組織が把握していることで課題を防げるとした[31]．エンタープライズアジャイル開発フレームワークではチーム間の連携を支援する方法を定義するが，それぞれの開発組織の文化や事情，扱うプロダクトは様々であるので，多くの開発組織で再現性高く実践可能な方法を定義するには至っていない．Bick らが提案する課題への対処法は妥当である．エンタープライズアジャイル開発を実践する必要がある開発組織は各フレームワークそれぞれのメリットとデメリットを把握し導入をすることが対処法となる[12]．各フレームワークは複数の開発チームでアジャイル開発を実現するために複雑な定義を持つので，開発組織が適切な開発プロセスフレームワークを選択することは困難なことが多い[12]．さらに，フレームワークの導入も複雑な定義によって困難であることが事例報告からも明らかとなっている[24]．

### 2.3.2 開発組織とエンタープライズアジャイル開発における役割

多くのアジャイル開発フレームワークは最大で 9 人を 1 チームとするスクラムから派生した[27]ので，エンタープライズアジャイル開発の課題は複数チームでのアジャイル開発実践である．スクラムを用いてアジャイル開発を行う開発組織にはプロダクトオーナ，スクラムマスタ，そしてスクラムチームが存在する．開発組織を拡大するためにはそのスクラムチームを複数にする方法がある[33]．

SAFe は小規模の開発組織から大規模な組織まで対応することを可能である．各開発チームはスクラムと同じようにプロダクトオーナ，スクラムマスタ，そして最大 9 人の開発者で構成されたチームとなる．

LeSS と LeSS Huge はスクラムを拡張したフレームワークである．LeSS は最大で 8 チームで構成され，LeSS Huge は 8 チーム以上で構成される．LeSS Huge は LeSS を拡張し大規模な開発を行うことを可能とする．LeSS Huge はエリアという概念を導入することで LeSS よりも大規模な開発を行うことを実現した．LeSS Huge においてプロダクトオーナは複数のエリアの開発状況を考慮し開発全体を管理する．各エリアはエリアプロダクトオーナが管理を行い担当する機能の開発管理を行う．プロダクトオーナは LeSS Huge において重要な役割を担当する[4]．主な役割は開発スケジュールの調整，ステークホルダとの交渉，リスク管理などである．1 人でこれら全ての役割を担うことは難しい．

### 2.3.3 アジャイル開発の大規模化

より大規模なアジャイル開発を行うためには課題がある[42]．大規模なアジャイル開発を行う際，その方法はいくつかある．注目を集めている方法として 3 つ以下に示す．

(1) 開発チームの地理的分散：GSD (Global Software Development)など地理的分散開発は重要な研究テーマであり実際の開発時の課題である．複数のチームでアジャイル開発を行うとき，地理的に離れたチームで協業することは発生し，多くの事例が報告されている[36][37][44].

(2) 複数機能の並行開発 : アジャイル並行開発は新しいテーマである[29]が複数のチームでアジャイル開発をするので，特にチーム間の連携に関する課題が発生し，これらは注目を集める研究テーマである[31][42]．アジャイル開発は対話をすることを重要視するので開発チームやステークホルダが増える並行開発においてその調整は課題となる．

(3) 複数プロダクトの並行開発：ソフトウェアプロダクトラインエンジニアリング（Software Product Line Engineering：SPLE）とアジャイル開発の統合である APLE (Agile Product Line Engineering)は複数のプロダクトを並行開発するアプローチとして組込ソフトウェア開発の分野において注目を集めてきた．組込ソフトウェア開発の分野においては SPLE の事例がある[11][15]ので APLE の事例も報告されているが，エンタープライズソフトウェア開発においては APLE の報告はない．

### 2.3.4 アジャイル開発におけるメトリクス

アジャイル開発の計測メトリクスは重要な研究課題である．

(1) ストーリポイントを用いた作業量計測

ストーリポイントはある作業量を基準とした相対的な計測単位である．ストーリポイントログアイテムはチームリファインメントにおいてストーリポイントが決定される．ストーリポイントストーリーポイントを用いることは実際の作業量と比較して正確であ

ることが報告されている[8]. その理由はある作業量を基準とした相対的な見積もり方法であることによる.

(2) ベロシティ （開発速度）の計測

ベロシティ はチームの開発速度を測るメトリクスである. ベロシティ は 1 スプリントにチームが完了した作業のストーリポイントを合計し算出する[16][22]. ベロシティ はチームの開発速度を計測する方法としては最適であるが, その値を他のチームと比較することは難しい. その理由はベロシティ はチームの成熟度やチームの人数によって増減することによる.

## 2.4 アプローチ

本章では，LeSS Huge を拡張し複数のプロダクトを並行開発する開発プロセスフレームワークを提案する．エンタープライズアジャイル開発フレームワークはそのフレームワークの複雑さ[24]やチーム間連携の困難さ[12][31][42]による再現性の課題があるので，開発組織が自身の組織に合う開発プロセスを設計することは対処法の一つと考え，提案方法は LeSS Huge を拡張することで構築した．LeSS Huge を拡張した理由は，LeSS を経験した開発者が組織内におり，その経験を生かした開発プロセスの設計が容易となり，組織全体への新たな開発プロセス導入を円滑に進めることが可能，という考えによる．本章では提案方法の 1 年間の実システムへの適用を通してその妥当性を評価する．

## 2.5 マルチプロダクトアジャイル開発方法

### 2.5.1 提案方法概要

本論文で提案するマルチプロダクトアジャイル開発方法を図 1 に示す．マルチプロダクトアジャイル開発方法は LeSS Huge をベースとし，プロダクトバックログの管理方法，複数プロダクトの優先度を管理するマルチプロダクトオーナの役割を追加した．

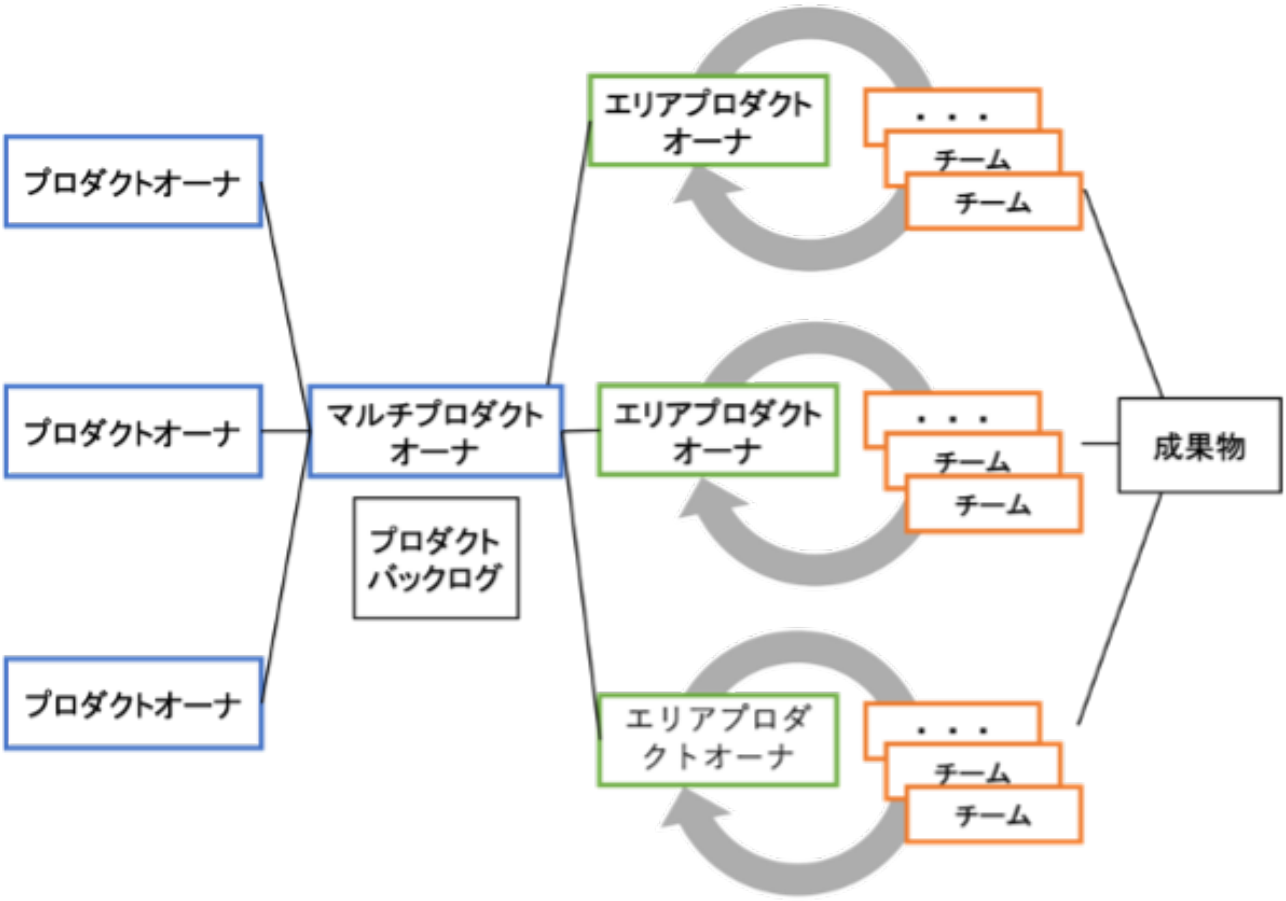


**図 1 マルチプロダクトアジャイル開発方法モデル**

### 2.5.2 プロダクトバックログアイテムの統合管理

マルチプロダクトアジャイル開発方法は 1 つのプロダクトバックログで複数プロダクトのプロダクトバックログアイテム (作業詳細を示したチケット) を管理する. LeSS, LeSS Huge では，プロダクトバックログは 1 つのプロダクトのプロダクトバックログアイテムが管理される．これに対してマルチプロダクトアジャイル開発方法では，扱うプロダクトすべてのプロダクトバックログアイテムを統合して管理するので複数プロダクトの開発優先度を柔軟に調整することが可能である．プロダクトバックログの整理は LeSS，LeSS Huge と同様にリファインメントの時間に実施する．複数プロダクトのプロダクトバックログアイテムが 1 つのプロダクトバックログで管理されるので，管理方法を工夫する必要がある．この点についてはアジャイル開発を行

う組織用に開発された JIRA Software[3]を利用しプロダクトバックログを作成することで継続的に整理することを可能とした．図 2 にプロダクトバックログ，プロダクトバックログアイテム例を示す．プロダクト，チーム，案件，技術領域などさまざまな視点で並び替えと検索を可能とすることで，登録されているプロダクトバックログアイテムを効率よく管理できる．これらは JIRA Software の機能であるフィルタやラベル付けの機能を用いることで実現した．


**図 2 複数プロダクトを管理するプロダクトバックログ**

### 2.5.3 マルチプロダクトオーナの役割

マルチプロダクトオーナは個別のプロダクトの責任は持たない．プロダクトオーナ，エリアプロダクトオーナ，ステークホルダとプロダクトバックログアイテムの優先度をすり合わせることで組織とプロダクトの成長を全体最適する．

### 2.5.4 プロダクトオーナの役割

プロダクトオーナは担当するプロダクトの成長に責任を持つ．プロダクトの新たな機能の開発，不具合の改修は開発チームが実施するが，プロダクトに関する最終責任はプロダクトオーナにある．プロダクトオーナはプロダクトが目指すビジョンを開発チームに伝える．マルチプロダクトアジャイル開発方法ではプロダクトオーナとマルチプロダクトオーナが議論することで組織とプロダクトの状況に応じて全体最適化を図る．

### 2.5.5 エリアプロダクトオーナの役割

エリアプロダクトオーナは担当する機能の開発に責任を持つ．エリアは複数の開発チームから構成され，エリアが担当する機能の開発をエリアプロダクトオーナが管理する．マルチプロダクトアジャイル開発方法では，エリアプロダクトオーナは開発を担当する機能についてプロダクトオーナ，マルチプロダクトオーナと共有する．エンタープライズアジャイル開発フレームワークではチーム間の連携が課題になる[31][42]ので，プロダクトオーナ，マルチプロダクトオーナと開発状況を共有し他のエリアの開発状況を把握することもエリアプロダクトオーナの重要

な役割である．

### 2.5.6 チーム内およびチーム間のコミュニケーション設計

提案方法では LeSS Huge が定義する方法を利用する．図 3 に木曜日に始まり，水曜日に終わる 1 週間のスプリントを実施する例を示す．月曜日，金曜日は打ち合わせ（セレモニ）を少なくすることで開発作業に集中する時間を確保した．各リファインメントの参加者を表 1 に示す．提案方法で新たに定義したマルチプロダクトオーナが参加するリファインメントを示した．

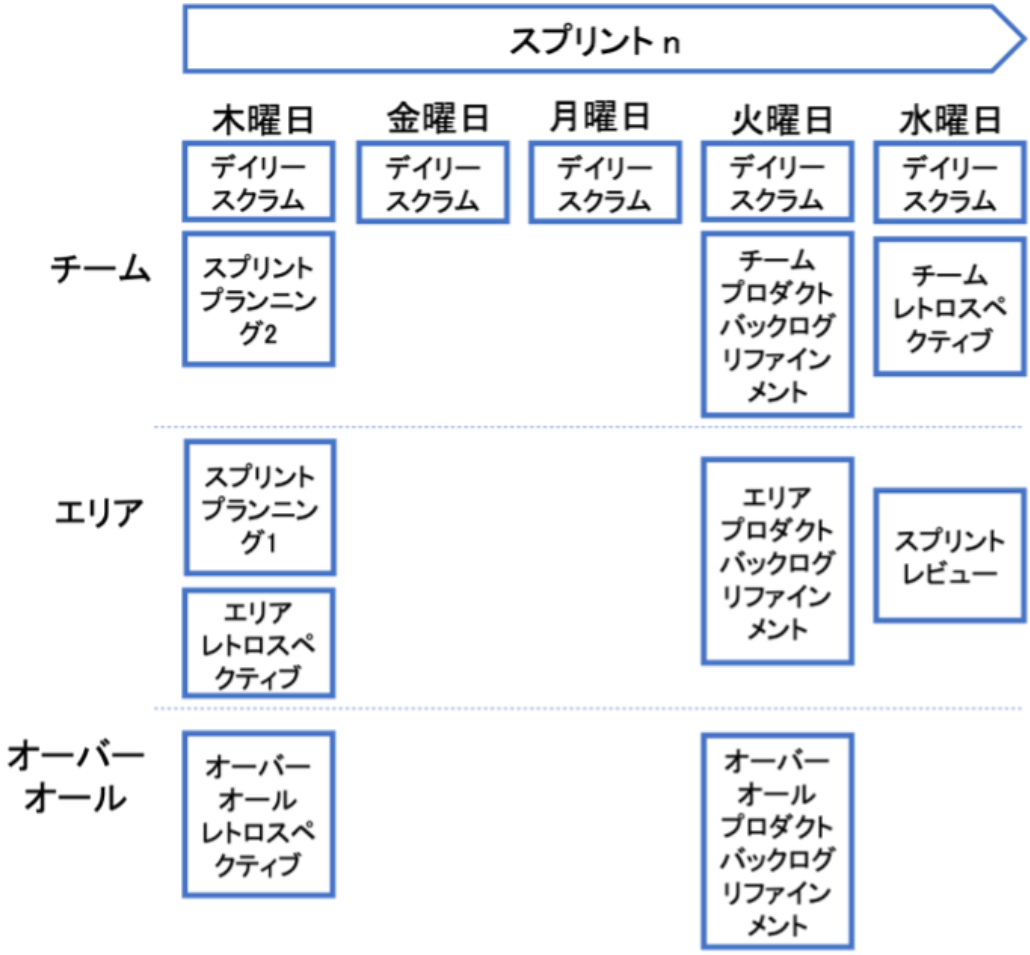


**図 3 コミュニケーション設計**

**表 1 提案方法におけるリファインメント参加者**

| セレモニ | 参加者 |
|---|---|
| ・オーバーオールプロダクトバックログリファインメント | ・マルチプロダクトオーナ<br>・エリアプロダクトオーナ |
| ・エリアプロダクトバックログリファインメント | ・エリアプロダクトオーナ<br>・チーム代表 |
| ・チームプロダクトバックログリファインメント | ・スクラムチームメンバ |

### 2.5.7 チーム間のコミュニケーション支援

エンタープライズアジャイル開発ではチーム間の開発状況の共有が課題となる[12][31]．マルチプロダクトアジャイル開発方法ではマルチプロダクトオーナ，プロダクトオーナ，エリアプロダクトオーナのコミュニケーションを円滑に進めることがこの課題解決につながる．マルチプロダクトオーナはプロダクトの開発責任を負ってはいないが，プロダクトオーナと連携し，組織および外部環境の状況や変化に対応して優先度を把握し，調整する．その優先度に基づき，マルチプロダクトオーナはプロダクトバックログの開発を調整する．詳細な開発計画はエリアプロダクトオーナと連携して調整する．このとき，エリアプロダクトオーナが担当するフィーチャ（機能）は優先度に応じて変化する．そのためプロダクトオーナともコミュニケーションをする必要がある．当面の予定が決まっているエリアにおいてはプロダクトオーナと連携することも必要となる．マルチプロダクトオーナ，プロダクトオマルチプロダクトオーナは相互に密に連携することでプロダクトバックログを継続的に整理する．これらの取り組みがチーム間の作業の依存関係の存在に気づくことへ繋がり，エンタープライズアジャイル開発を効率的に進めることへ繋がる[31].

### 2.5.8 担当するプロダクトとエリアのパターン分類による柔軟な開発の実現

マルチプロダクトアジャイル開発方法では複数のプロダクトを 1 つのプロダクトバックログで管理している．そのため各スプリントにおいてはプロダクトバックログの優先度に応じて各エリアが担当するプロダクト，フィーチャに変更が発生する．このとき，各スプリントにおいて各エリアと担当するプロダクトとの関係を次の 3 つのパターンに分類して開発を進めた.

1) パターン 1: 単一プロダクトの優先開発
   図 4 にパターン 1 の構造を示す．図に示すように，任意のスプリントにおいてすべてのチームがプロダクト A のフィーチャの開発を担当する．図中プロダクト B，C では今後予定されているフィーチャの要件定義，見積もりを進める.

2) パターン 2: 一部プロダクトの優先開発
   図 5 にパターン 2 の構造を示す．プロダクト A，プロダクト B の開発を行い，プロダクト C では今後開発が予定されているフィーチャの要件定義，見積もりを進める．開発組織として一部のプロダクトの開発を優先する場合に利用するパターンである.

3) パターン 3: 全プロダクトの開発並行
   図 6 にパターン 3 の構造を示す．各プロダクトにそれぞれ開発を担当するエリアが割り当てられ，並行開発を行う.

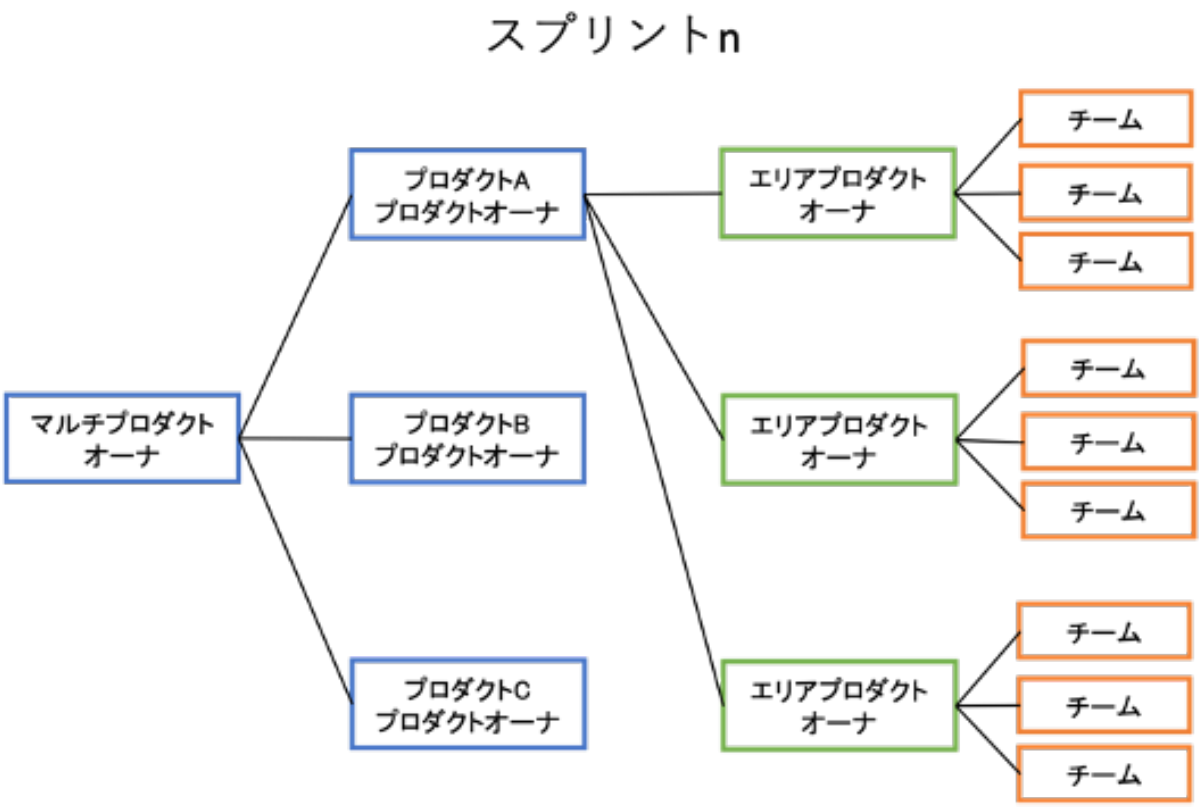


図 4 パターン1

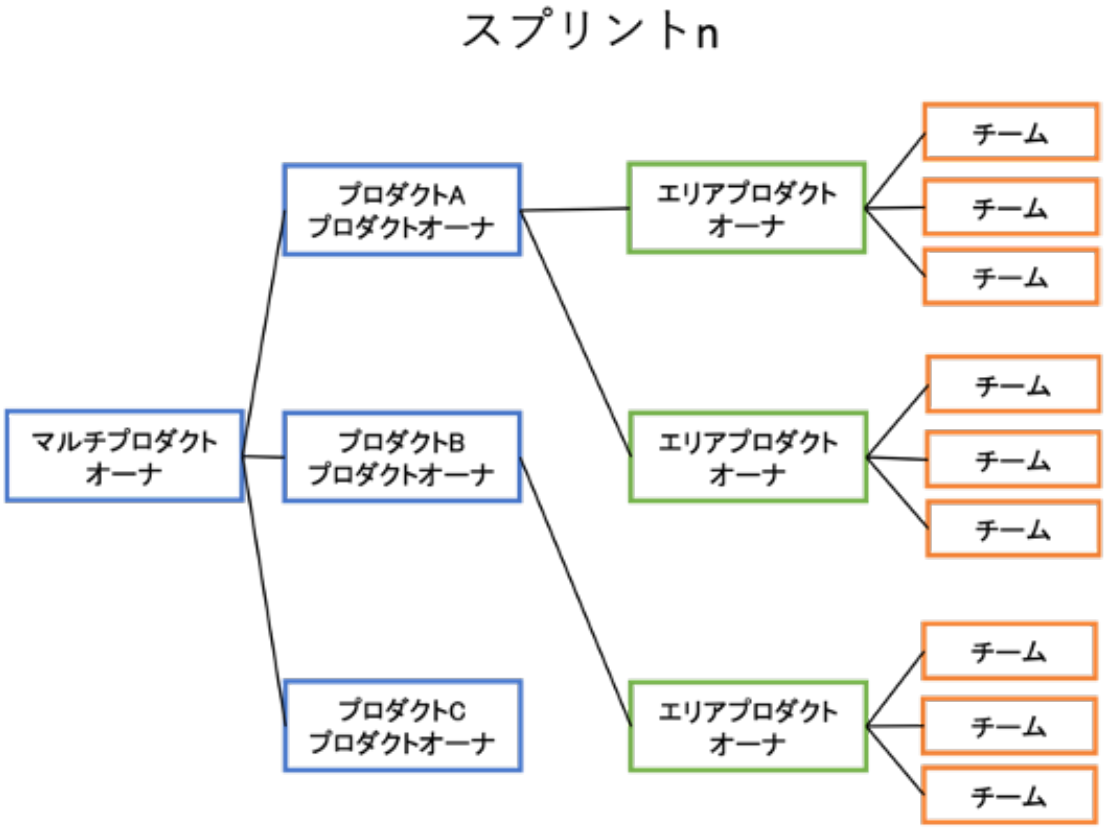


図 5 パターン2

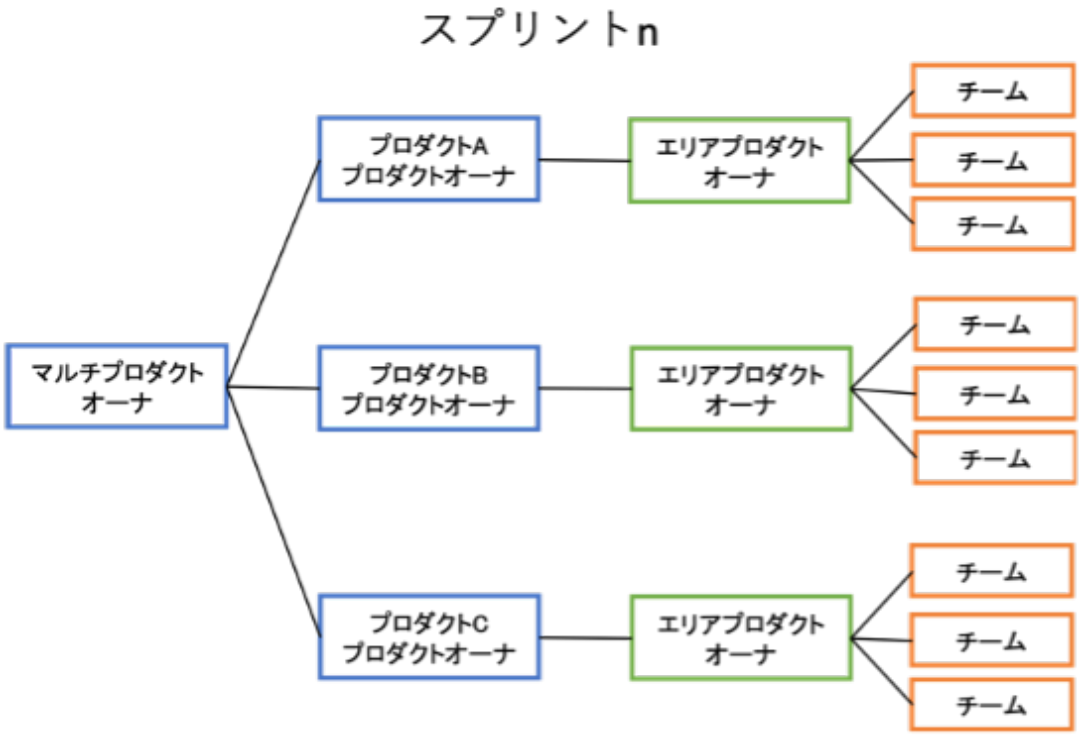
スプリントn
マルチプロダクト
オーナ
プロダクトA
プロダクトオーナ
プロダクトB
プロダクトオーナ
プロダクトC
プロダクトオーナ
エリアプロダクト
オーナ
エリアプロダクト
オーナ
エリアプロダクト
オーナ
チーム
チーム
チーム
チーム
チーム
チーム
チーム
チーム
チーム

**図 6 パターン3**

## 2.6 提案方法の実践

2017 年 11 月 30 日（木）から 2018 年 10 月 10 日（水）の期間における事例を報告する.

### 2.6.1 移行前の開発方法

提案開発方法を実践する以前は各プロダクト内で案件ごとにプロジェクト管理を行う開発方法を採用しており，スクラムや LeSS を利用していなかった．移行前の開発組織を図 7 に示す．プロダクトごとにプロダクトの責任者（プロダクトオーナ）がおり，担当する技術領域ごとに開発チームが構成されていた．開発プロセスを図 8 に示す．初期開発フェーズとして最小限の機能開発とリリースを実施し，初回の機能リリース後にユーザの利用状況をデータで確認する．データを確認し機能改修を繰り返すことで，利用者の要求に柔軟に対応していた.

開発期間は仮設定したリリース日から逆算し，現在の開発リソースと要求を満たす最小限の機能開発を優先度付けすることで設定する．スクラムのような一定のイテレーション期間を設けてはいないが，優先度を設定し短期間でリリースを繰り返すことを実践していた．この方法を用い各アプリケーションで平均して毎月 2〜3 回のリリースを行っていた.

各開発チームの進捗管理はチームに一任されていた．GitHub が提供する Issue 機能，Jira Software，ホワイトボードに付箋紙を用いて作業管理をするなど方法は様々であった．チームの自主性を尊重した開発環境であったが，全体の開発進捗の把握が難しいという課題があった.

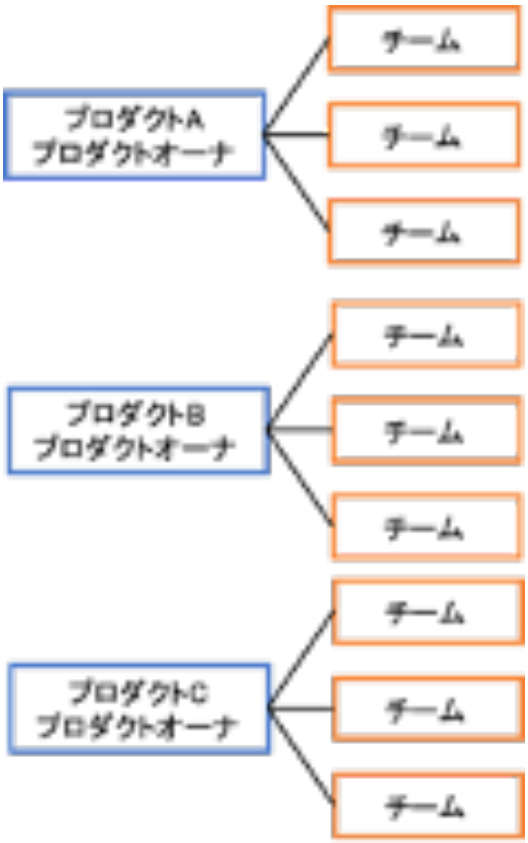


図 7 移行前の開発体制

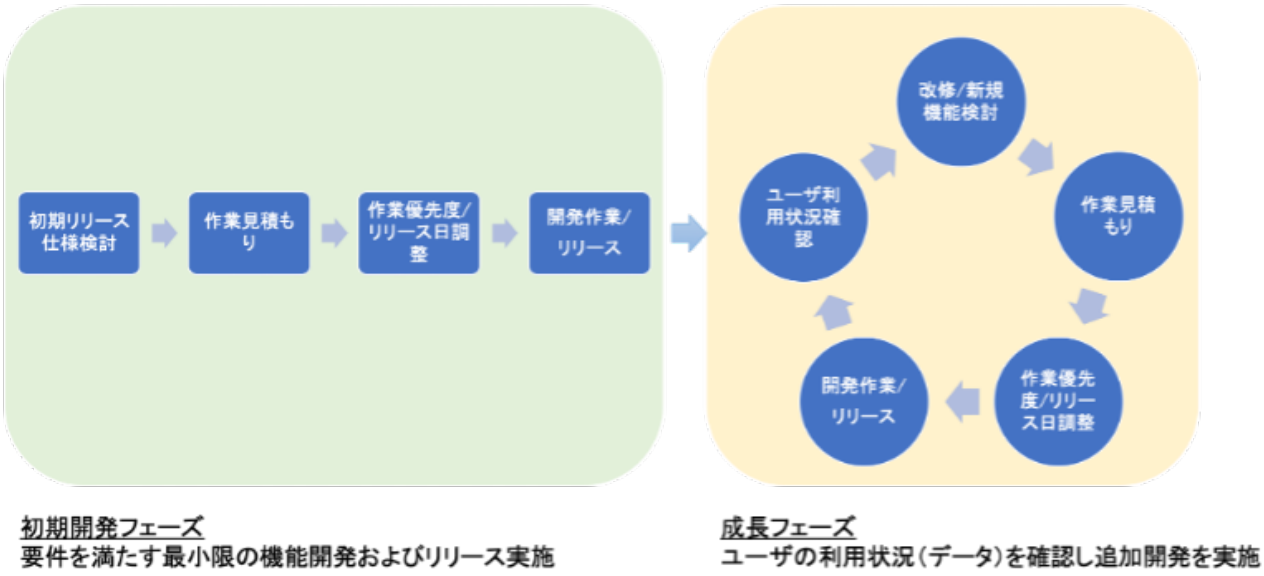


**図 8 移行前の開発プロセス**

### 2.6.2 提案方法への移行期間

2017 年 10 月から 2 カ月を開発体制移行期間とし，プロダクトバックログの準備や組織全体への説明を進めた．開発体制の移行は LeSS による開発経験があるメンバを中心に進めた．体制の移行には大きな問題はなかった．それは 2.6.1 で示したように移行前にアジャイル開発を取り入れていたことによる．短期間で小さくリリースを積み重ねるというアジャイル開発方法への理解が浸透していたことは開発体制の移行を円滑に進めた．

### 2.6.3 開発プロダクト群

開発組織は 2 つのスマートフォン向けアプリとブラウザ向けサービスを担当する． 表 2 にそれぞれのプロダクトが提供する機能を示す．各プロダクトのソースコードは個別に管理され，独立したプロダクトである．

**表 2 プロダクト詳細**

| プロダクト | 提供機能 |
|---|---|
| 地図サービス（スマートフォン向けアプリ） | 地図機能，近隣の店舗検索機能，おすすめ店舗情報，任意の地点への徒歩案内機能 |
| 自動車向け地図ナビゲーションサービス | カーナビゲーション機能，駐車場情報，渋滞情報の地図表示，道路規制情報の地図面表示 |
| 地図サービス（ブラウザ向け） | PC ブラウザでの地図利用，スマートフォンブラウザでの地図利用 |

### 2.6.4 適用結果

組織変更により変動もあったがチーム数は常時10チーム前後，各チームは5名前後であった．各チームでスクラムマスタの役割を担うメンバは 1 名であった．提案開発方法の有効性をベロ

シティを用いて評価する．図 9 に 2017 年 11 月 30 日（木）から 2018 年 10 月 10 日（水）の期間に提案開発方法を実践した直近 3 スプリントの移動平均ベロシティを示す．スプリントは木曜日に始まり水曜日に終了する 1 週間である．

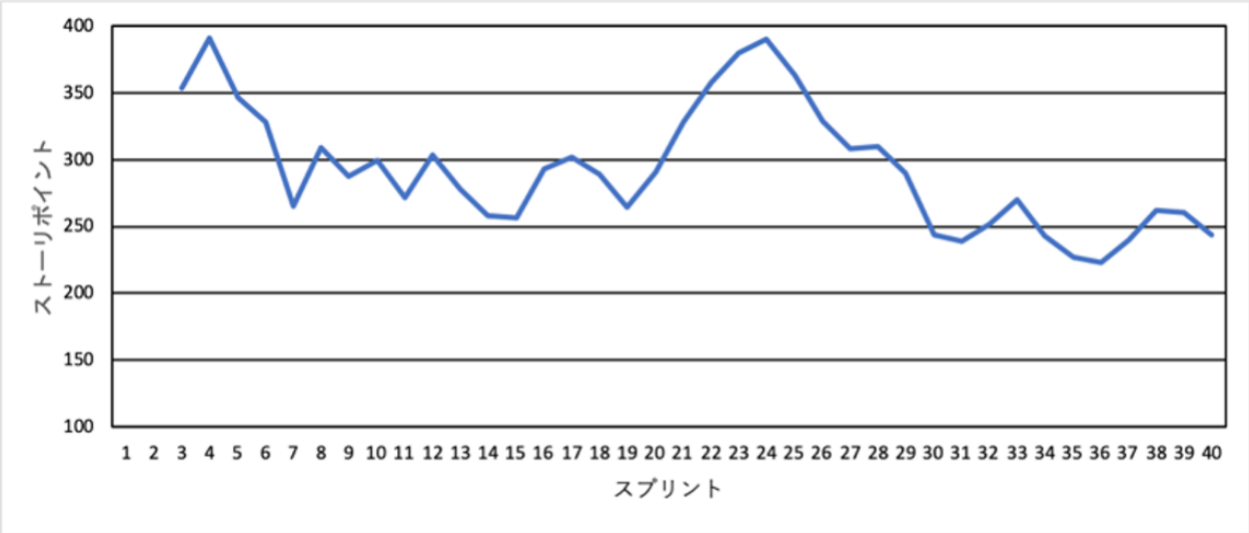


**図 9 直近 3 スプリントの移動平均ベロシティ**

各スプリントのベロシティはチームメンバの勤怠状況や稼働日数による影響を受けるので，直近 3 スプリントの移動平均ベロシティを評価尺度とすることでその影響を軽減する．ベロシティによって組織におけるスプリントごとの開発速度，すなわち，開発生産性を可視化する．さらに，ベロシティとその変動，すなわち開発の加減速はスプリント単位を超えた中期的な開発とそのリリース配分の計画やリスク管理に有用である．

組織全体の開発の達成度の尺度として，コミット率を式（1）で定義する．2017 年 11 月 30 日（木）から 2018 年 10 月 10 日（水）の間のコミット率の推移を図 10 に示す．図 10 中には期間中のコミット率の回帰直線を破線で示す．コミット率はスプリントプランニング時に各チームがそのスプリントにおいて合意した達成可能な目標であるので，移動平均とする必要はない．

$$\text{コミット率} = \frac{\text{完了したストーリ(ポイント)}}{\text{プランニング時に決定したストーリ(ポイント)}} \qquad (1)$$

図 10 よりコミット率は 70%から 100%の間で変動しているが，体制の立ち上げ初期を除くと組織全体としておおむね 80%のコミット率を維持できていることがわかる．最大値で 100%となり，最小値のスプリント 6，14，25 では前後のスプリントと比較してコミット率が低下している傾向がある．これらのスプリントでは優先度が低く先送りとなっていた，いわゆる Undone ワーク[3]を解消することに開発チームが取り組んだ結果と考えられる．このことから，Undone ワークを解消する周期は，開発規模やスプリント単位を超えた組織活動の変化などに応じて調整が必要であると言える．

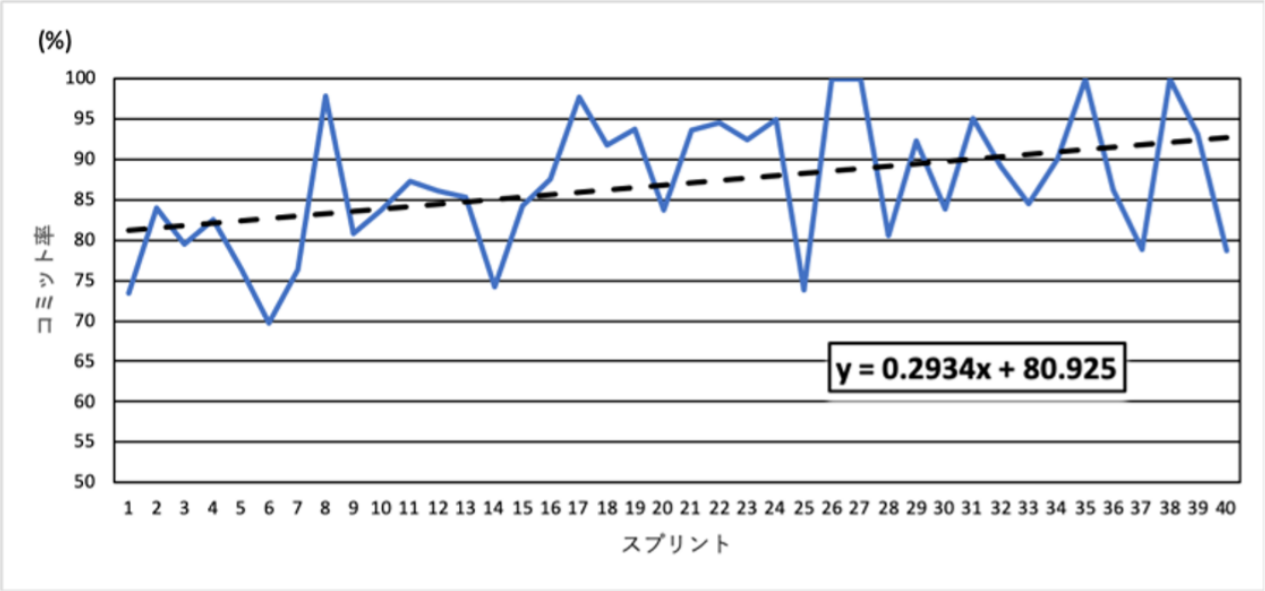
(%)
100
95
90
85
80
75
70
65
60
55
50
コミット率
y = 0.2934x + 80.925
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
スプリント

**図 10** コミット率

## 2.7 考察

### 2.7.1 研究課題(1)に対する考察

提案方法は 1 年間の実践において組織変更という変化に実際に対応できたことがベロシティ，コミット率の推移より明らかになった．2.3.1 で示したように様々な開発プロセスフレームワークが提案されているが，開発組織が対応する要求は多様である．Zoran らはアジャイル開発を行う組織が担当するソフトウェアのアーキテクチャに関する傾向をまとめ，これらはエンタープライズアジャイル開発を行う開発組織が対応する要求と見ることができる[45]．例えば，ビジネス要求と技術負債への対応を考慮した作業計画，地理的に分散したチーム間の連携はソフトウェアアーキテクチャが対応する要求であり，エンタープライズアジャイル開発を行う開発組織が対応する要求である[45]．このように開発組織が対応する要求は多様であるので，すべての組織に適する開発プロセスフレームワーク構築は困難である．よって，開発組織の要件に適する最小の開発プロセスフレームワークを導入し，プロダクトの増加や開発組織の拡大など要求が変化したときにより大規模な開発プロセスフレームワークを導入することが適する．すなわち，変化に対応可能な複数プロダクトを扱うエンタープライズアジャイル開発方法は拡大する開発組織に合わせ段階的にそのフレームワークも選択することが可能である必要がある．提案方法は LeSS Huge を拡張して設計したフレームワークであるので，LeSS または LeSS Huge を実践する開発組織の要求が変化した場合に導入候補として有用である．実際に，多くの事例報告がある SAFe[10]も拡大する開発組織の要求に合わせ開発組織の規模別に複数のフレームワークを定義する．

### 2.7.2 研究課題(2)に対する考察

複数のプロダクトの開発を組織全体の優先度に応じて変更しつつ開発を進めた．コミット率は 70%から 100%の間で変動が見られたが，組織変更時の開発組織の再編中にも 80%前後のコミット率を維持できた．さらに，期間中のコミット率は上昇傾向にあった．これらより，提案方法はこの開発組織において有用であったと評価できる．他方，他の開発組織においても同様に有用か事例検証が必要である．特にチーム間の連携についてより多様な開発組織での事例検証が必要である．すなわち，提案方法でもチーム間の連携[42]は課題となるので，Bick らが示すように作業の依存関係の存在への気づき[31]が提案方法を他の組織において実施する際の鍵となる．

### 2.7.3 チーム間の連携

2.3.1 で述べたようにエンタープライズアジャイル開発を行う時，チーム間の連携が課題となる．提案方法では LeSS Huge におけるプロダクトオーナの役割の一部をマルチプロダクトオーナへ権限委譲することで複数のプロダクトを並行開発する．その結果，プロダクトオーナから開発チームの状況把握が難しいという意見があり，エンタープライズアジャイル開発で課題となるチーム間の連携の困難さを確認した．この課題に対処するため新たなセレモ二の定義やセレ

モ二の参加者の変更を検討する必要がある．

### 2.7.4 依存関係を持つ作業と開発スケジュールへの影響

依存関係を持つ作業を複数のエリアで並行開発する場合，開発スケジュールへの影響が発生することがある．依存関係を持つ作業例として，作業間のリリース順序や同一ソフトウェアコンポーネントを修正する作業などがあった．これはエンタープライズアジャイル開発実践時に発生するチーム間の連携に関する課題[42]であり，事例検証時にも発生した．提案方法では，マルチプロダクトオーナ，プロダクトオーナ，エリアプロダクトオーナが密に連携をとり開発状況を整理することで作業間の依存関係による遅延発生を未然に防ぐよう取り組んだ．これは依存関係の存在に気づくための取り組みであり，Bick らが提案するこの課題へ対処する方法と一致する[31]．依存関係の存在は開発スケジュールへ影響を与えるので．依存関係が存在する作業の発生を軽減する取り組みが必要だが，それらの対処は開発プロセスを改善することでは対処が難しいと考える．

### 2.7.5 複数プロダクトを並行開発する時に必要なドメイン知識と学習コスト

本事例では 3 つのプロダクトを並行開発した．その際，図 7 に示したように 1 つのプロダクトに集中して開発を進めたことで，一部チームは開発経験のないプロダクトの機能開発を担当した．すなわち，プロダクトに対するドメイン知識がない状態で開発を担当したチームが存在した．図 10 で示すコミット率は右肩上がりであることから，長期的には各チームは開発に必要なドメイン知識を獲得していたと考える．これは，経験したことのないプロダクトであっても地図関連の技術を知る技術者が多く，それが学習コストを抑え，早い段階でドメイン知識を獲得できたことによると考える．提案方法では，チームは様々なプロダクトを担当することになる．ドメイン知識の獲得に時間がかかる場合もあるので，それらを考慮したスケジュールを組み立てる必要がある．

## 2.8 まとめ

複数のプロダクトを並行開発するエンタープライズアジャイル開発方法を提案した．エンタープライズアジャイル開発フレームワークの一つである LeSS Huge では一つのプロダクトを複数のチームで開発する方法を定義するが，複数のプロダクトを一括で開発することはできない．開発組織への要求へ対応するため，複数プロダクトの機能開発を一括で管理するマルチプロダクトアジャイル開発方法を提案した．提案方法は LeSS Huge[19]を拡張した開発プロセスフレームワークであり，複数プロダクトを一括管理するため新たな役割の追加，開発組織の構造を新たに定義した．エンタープライズアジャイ開発フレームワークを組織へ導入することは容易ではなく[12][24]，組織の要求，特に組織の規模に応じて段階的にフレームワークを適用していくことが妥当であり，実際に多くの実践事例が報告される SAFe[20][30]や LeSS[19]も組織の規模に応じたフレームワークを提示する．提案方法は LeSS または LeSS Huge を利用した組織がさらに組織の規模を拡大する時の選択肢を新たに提示した．

本章では，提案方法を 3 つの市販プロダクトの開発に適用した事例検証を実施した．その結果，実践した開発組織において提案方法が有用であったことを開発組織全体のベロシティの推移，すなわち開発組織が 1 年間に完了した開発作業量の推移が増加傾向であったことから確認した．エンタープライズアジャイル開発を行う組織への要求は多様[45]であり，それに対応するため，フレームワークは組織の規模に応じて段階的にフレームワークを切り替える方法を提示する[19][20][30]．これまで，LeSS または LeSS Huge では複数のプロダクトを一括して扱う方法を提示できていない[19]．本章では，複数のプロダクトを一括して扱う必要がある組織において提案方法を事例検証として実施した．特に LeSS または LeSS Huge を利用した経験のある組織がさらに組織の規模を拡大する時に選択可能なフレームワークを新たに提示した．

SAFe，LeSS Huge 共に開発組織への要求に対し組織の規模に応じ，段階的に対応可能なフレームワークを提供するが，複数のプロダクトを一括管理し，並行開発する方法は提示していない[19][20][30]．事例検証を行った組織では複数のプロダクトを一括して開発を行う必要があり，それは既存フレームワークでは対応が難しかったので，提案方法は複数のプロダクトを一括管理し，並行開発する方法を提案し，1 年間の実践事例を示した．開発組織の要求は多様[45]であるので，それら全てに対応する開発プロセスフレームワークの構築は困難であるが，提案方法によって複数のプロダクトを一括管理し，並行開発したいという要求へ対応が可能である．

# 3 エンタープライズアジャイル並行開発に適したソフトウェアアーキテクチャ評価方法に関する研究

複数チームでアジャイル開発を行う時，チーム間の連携が課題となる[42]．本論文でも 2.7.3 でその課題を確認した．チーム間の連携が必要となる理由は人員，作業，知識の依存関係が存在することによる[9]．Nord らは組織とアーキテクチャの関係を整理し，依存関係への対処の重要性を述べた[28]．エンタープライズアジャイル開発フレームワークでは，依存関係の分析方法やその対処方法に関する議論はない．

本章では，作業間の依存関係に着目することで，エンタープライズアジャイル開発に適するソフトウェアアーキテクチャを評価する方法を提案する．提案方法を用いることで，依存関係の存在する作業を少なくすることが可能である．すなわち，提案方法を用い，組織に適するソフトウェアアーキテクチャを選択することで，複数チームでのアジャイル開発を円滑に進めることにつながる．

## 3.1 研究の背景

複数機能の並行開発を可能とするエンタープライズアジャイル開発(Enterprise Agile 開発，以下 EA 開発と略記)の実践事例が数多く報告されている．それぞれの実践事例は導入した EA 開発フレームワークがその組織において機能したことを示す．他方，他の組織において同様にその EA 開発が機能するかを判断することは難しい．2.5 で複数のプロダクトを一括して管理する EA 開発フレームワークを提案した．2.6 で示したように実プロダクトを用いた 1 年間の事例検証を行ったが提案方法が他の開発組織においても同様に有用であるか，特にチーム間の連携に関する問題[31][42]の観点でさらなる事例検証は必要である．EA 開発を実践する組織への調査結果からも，導入時の課題として組織の文化，EA 開発フレームワークの複雑さ，ソフトウェアアーキテクチャ(以下 SA と略記)の不適合など様々な観点の課題が挙げられている[24].

本章では，EA 開発と SA の関係に着目する．全体が一つのコンポーネントで設計されているモノリシックアーキテクチャは EA 開発に適していないことが報告されている[24]．Nord らはエンタープライズアジャイル開発における組織とアーキテクチャの関係，さらにエンタープライズアジャイル開発に適するソフトウェアアーキテクチャとして複数の層に分かれたレイヤードアーキテクチャを提案した[28]．SAFe ではアーキテクチャ滑走路（Architectural Runway）としてアーキテクチャを継続的に改善することを定義する[20][30]が，具体的な技術やアーキテクチャについて言及はなく，Nord らの提案がその参考になる．他方，Nord らはアーキテクチャ上の依存関係の存在の分析や，依存関係が存在する場合の対処については今後の課題とした[28]．本章では依存関係のある作業，特に同一コンポーネントの更新に着目する．同一コンポーネント更新作業が発生する時，その整合性を取る作業，すなわちコンフリクトの解消作業が発生する．この課題に対処する方法として開発組織に合う SA の選択がある．SA のみでこの課題へ対処することは困難であるので，開発プロセスフレームワークを組み合わせて利用することは有効である．Large Scale Scrum (LeSS)，LeSS Huge[19]などの実践事例が報告されている EA 開発フレームワークでは SA に関する詳細な言及はない．Scaled Agile Framework (SAFe)[20][30]ではアーキテクチャの重要性に関して言及されるが，エンタープライズアジャイル開発に適する具体的なアーキテクチャの提示やその実現方法については言及がない．本章では EA の並行開発時に発生するコンフリクト解消作業の発生に着目することで開発組織と SA のコンポーネントが適合するかを評価する方法を提案する．提案方法を用いることで開発組織と SA コンポーネントの適合度合いを評価する．評価結果を用いることで 2.7.4 でも挙げた開発を進める機能間の依存関係に起因する開発スケジュールへの影響を軽減することが期待できる．

## 3.2 研究課題

本章の研究課題を以下に示す.

(1) EA 開発に適した SA 評価方法の持つべき特性は何か

エンタープライズアジャイル開発では，チーム間の連携の課題への対処，すなわち依存関係への対処が開発を円滑に行うことの鍵である[31][42]．Nord らは組織とアーキテクチャの関係を整理した[28]．これらを踏まえ，本章では作業間の依存関係に着目した評価方法を構築する．作業間の依存関係に着目する理由は，アジャイル開発に影響を与える依存関係である人員，知識，作業[9]の中で作業間の依存関係が最も組織とアーキテクチャの評価に影響を与えると考えたことによる．提案方法を用いた事例検証を通し，評価方法の持つべき特性を検討する．

(2) EA 開発のマネジメントを支援する開発プロセスの可視化は可能か

複数チームのアジャイル開発は人員，知識，作業の依存関係を原因とするチーム間の連携が課題となる[42]．全ての依存関係への対処が困難な場合もあり，その時の対処方法としてプロジェクトマネジメント技術がある[31]．すなわち，エンタープライズアジャイル開発における開発プロセスの可視化を行うことは依存関係への対処が困難な場合の補助になる．本章では，作業間の依存関係に着目することでプロジェクトマネジメントを支援する開発プロセスの可視化を研究課題とする．

## 3.3 関連研究

### 3.3.1 EA 開発フレームワークとアーキテクチャ

アジャイル開発は小さな一つのチームで実施することから時間を経て複数のチームで大規模に開発する EA 開発へ進化をしてきた. Scaled Agile Framework（SAFe）, Large Scale Scrum（LeSS）, Scrum of Scrums など様々な EA 開発フレームワークが提案されておりその事例が数多く報告されている[10][12]. EA 開発フレームワークの開発プロセスは一つのチームでアジャイル開発をする時と同様に，短い期間に規模が小さなリリースを繰り返すことで変化に対応する．EA 開発では複数のチームが並行して開発を行う必要があり，各 EA 開発フレームワークではタスクの優先度付けを行う仕組みを提供している．これはプロダクトオーナが担う役割の一つであり，LeSS, LeSS Huge ではこの優先度付けをするツールとしてプロダクトバックログと呼ぶ作業一覧表を用意する．各 EA 開発フレームワークは優先度付けを行うための打ち合わせを実施することを定義している．図 1 に EA のプロダクトオーナの役割を示す[4]．図 11 はプロダクトオーナの役割が要求の把握（図中 Conflicting Business Needs），ステークホルダとのコミュニケーション（図中 Sponsor Intermediary），優先度付け（図中 Groom Prioritizer），そしてリリース管理（図中 Release Master）など多岐に渡ることを示す．各 EA 開発フレームワークは大規模なアジャイル開発を行うため多くのルールを定義するがルールの複雑さは導入時の課題となっている[24].

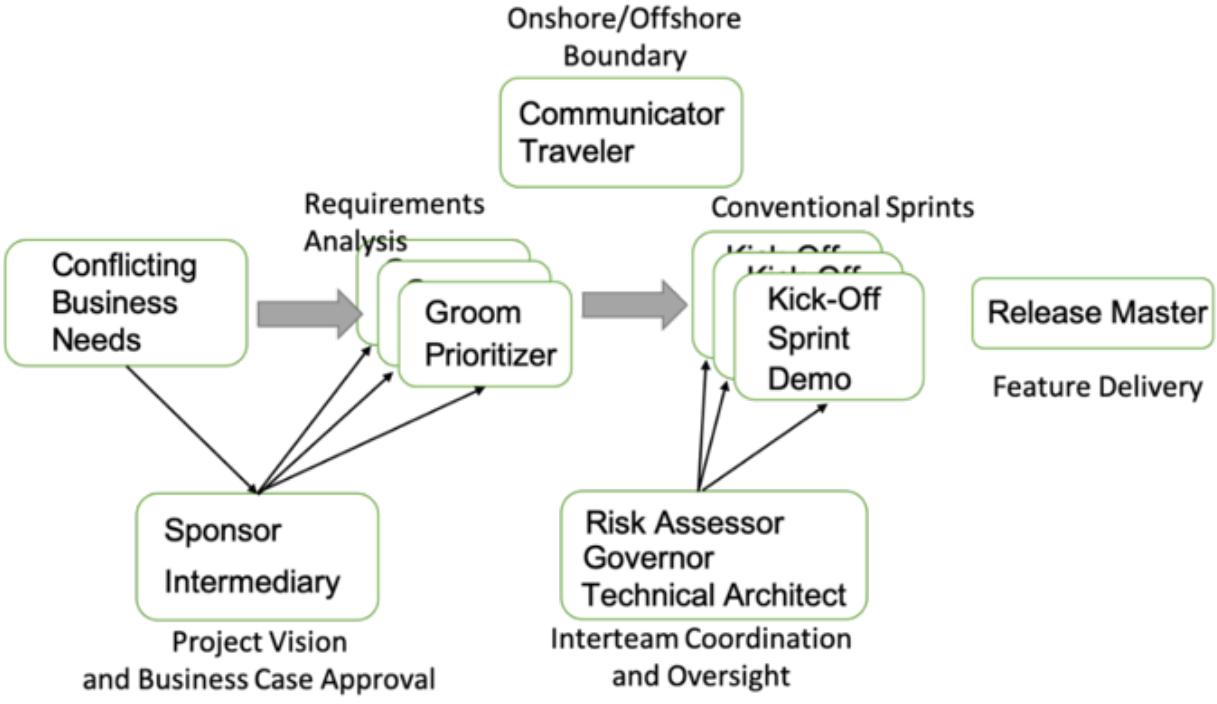


**図 11 EA のプロダクトオーナの役割**

EA 開発ではリリース戦略も課題の一つである[42]．短い期間に複数のチームで並行して開発とリリースを繰り返すので，それらを支援する技術と環境が必要である．具体的な技術としてビルド自動化，TDD(Test Driven Development)[5]，マイクロサービスアーキテクチャ[23]などが挙げ

られる．これらを組み合わせることで短い期間にリリースを繰り返すことが可能となるが，2.7.4 で述べたように依存関係のある機能を並行開発する場合もあり，短期間にリリースを繰り返すことは容易ではない．EA 開発フレームワークは開発プロセスのフレームワークであるので，開発プロセス観点の対処については議論されるが短期間のリリースを支える技術について述べられることはない．SA についても同様に EA 開発フレームワークで述べられることはないが，SAFe ではアーキテクチャ滑走路（Architectural Runway）と呼ぶ概念を定義[20][30]し，アーキテクチャの重要性について言及する．他方，エンタープライズアジャイル開発に適するアーキテクチャや技術について言及されることはない．モノリシックアーキテクチャは EA 開発に適していないことが報告されている[24]ように開発プロセスだけで並行開発を実現することは難しい．Nord らは開発組織とアーキテクチャの関係を整理し，エンタープライズアジャイル開発に適する SA として複数の層に分かれたレイヤードアーキテクチャが適することを示した[28]．他方，アーキテクチャ上の依存関係の存在の分析や，その対処方法については今後の課題とした[28]．さらに，Nord らが今後の課題とした依存関係の問題はチーム間の連携の課題[42]へとつながる．

### 3.3.2 並行開発を支援する SA とソフトウェア設計技術

MVC や MVP，レイヤードアーキテクチャの SA はプレゼンテーション層，ビジネスロジック層などを機能分割のためのコンポーネントとして設計するドメイン独立な SA である．この境界に応じたチーム編成を行い，複数機能を並行開発することを考える．このときドメイン独立な SA はチームを横断して作業調整をする必要がある．ドメインの概念を導入した SA はチームを横断した作業調整は必要ない．それはドメイン内の実装に限定されるという理由による[14]．ドメインの概念を導入した SA として Clean Architecture（図 12）[21]がある．ドメイン固有の実装を図中エンティティへ集約することでドメインの概念を SA へ導入する．Clean Architecture はドメインを定義するモデリング方法 Domain-Driven Development（DDD）[13]を実現する方法の一つである．

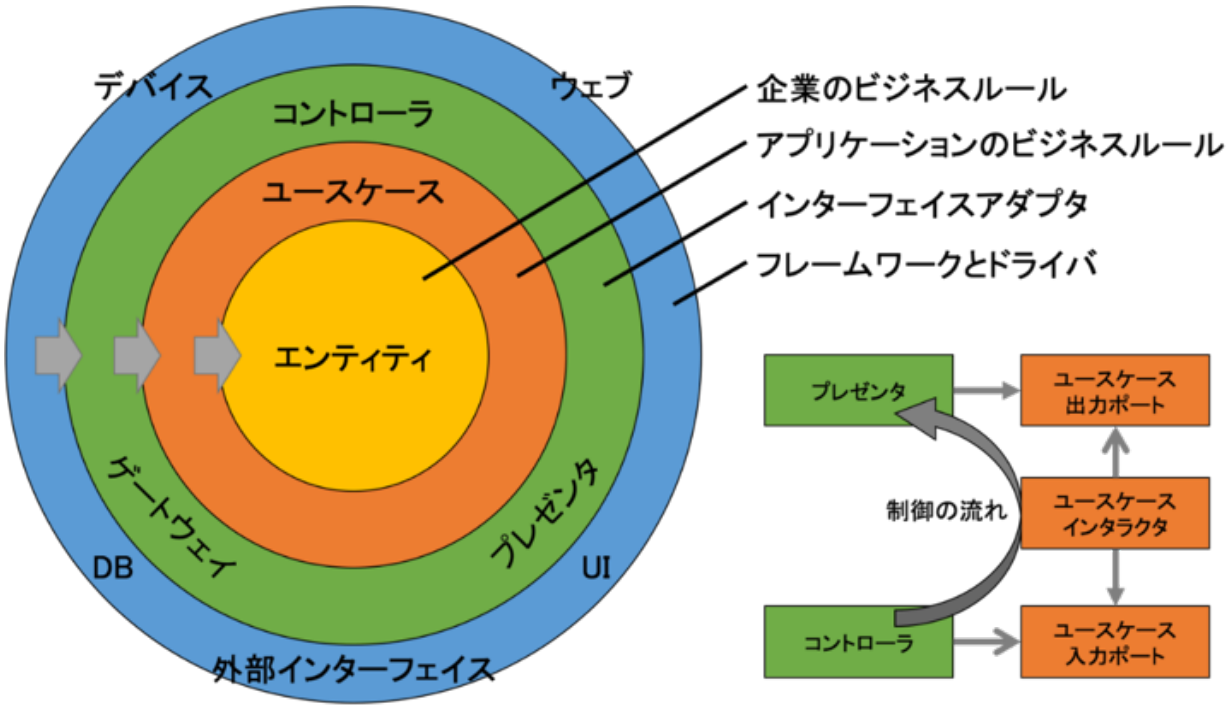


**図 12 Clean Architecture**

Clean Architecture を iOS アプリケーション開発時に利用する方法として VIPER（図 13）が提案されている．VIPER は iOS アプリケーション開発時に利用する標準ライブラリの特性を考慮し Clean Architecture を実現する方法を示しているので, エンティティへドメイン固有の実装を集約することでドメインの概念を導入する．Clean Architecture はドメインの概念を SA に追加することで SA に新しいコンポーネントを持つレイヤを構築する．VIPRE は Clean Architecture を iOS アプリケーションで実現するためのソフトウェア設計を示す．すなわち，Clean Architecture が示す SA を実現する VIPER を用いて設計されたソフトウェアは，SA のコンポーネント粒度が小さく，明確な責務を持つので，並行開発が容易となる．Nord らはエンタープライズアジャイル開発に適する SA としてレイヤードアーキテクチャの利用を提案しており，その理由も同様である [28].

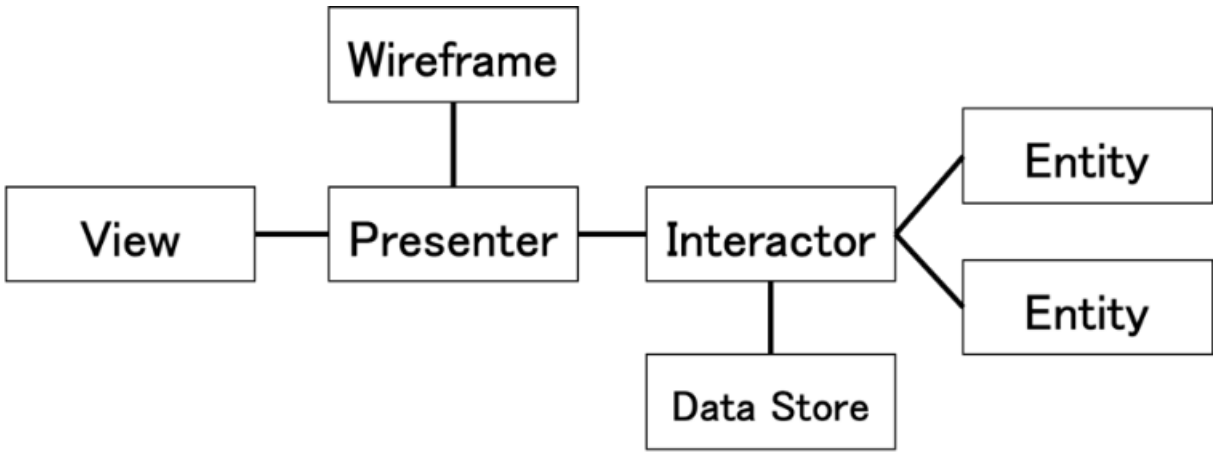


**図 13 VIPER**

サービス指向アーキテクチャ（Service-Oriented Architectures: SOA）はアプリケーションフロントエンド，サービス，サービスリポジトリ，サービスバスから構成される SA である[18]．マイクロサービスアーキテクチャは SOA を実現する方法の一つであり，ドメイン単位でサービスを分割する．ドメイン間は疎結合となり，HTTP 通信などを用いた連携を行うのでドメイン内で利用する技術の選択が柔軟になる[14]．その結果，サービスとチーム構造を一致させることが可能となりドメインごとにリリースが可能となる[23]．チームは担当するドメイン以外を意識することなく開発が可能なので，複数のサービスが並行して開発可能となる．よって，マイクロサービスアーキテクチャは並行開発を支援する技術である．

### 3.3.3 Scenario-Based Architecture Analysis Method（SAAM）

SAAM は用意したシナリオの集合に対してスコアリングを行うことで評価対象の SA の変更容易性や拡張性を評価する方法である[7]．SAAM は簡潔な評価方法であり学習コストが低く，実施しやすいという特徴を持つ．評価結果がシナリオに依存する点が課題である．

## 3.4 アプローチ

本章では，EA 開発に適した SA かを評価するための SA 評価方法を提案する．EA 並行開発時の作業の依存関係に着目する．特に同一コンポーネントの更新が必要となる依存関係に着目し，その整合性を取る作業，すなわちコンフリクト解消作業の発生に注目する．コンフリクト解消作業の発生は開発組織と SA コンポーネントの不適合を原因として発生すると考えた．コンフリクトを解消，すなわち依存関係を解消するまで開発作業は中断する（待ち時間の発生）ので，開発スケジュールに影響する．開発スケジュールに影響を与えるコンフリクトおよび待ち時間の発生が少ない SA は EA 並行開発に適する，という観点から SA 評価方法を構築した．提案方法をスマートフォンアプリケーションに適用し評価方法の有効性，妥当性を示す．

## 3.5 SAAM for EA (Scenario-Based Architecture Analysis Method for Enterprise Agile)

### 3.5.1 評価方法の概要

EA において課題となる並行開発に適した SA の評価を行うため SAAM for EA を提案する（図 14）．SAAM for EA は SAAM と同様にシナリオに基づき SA を評価する．

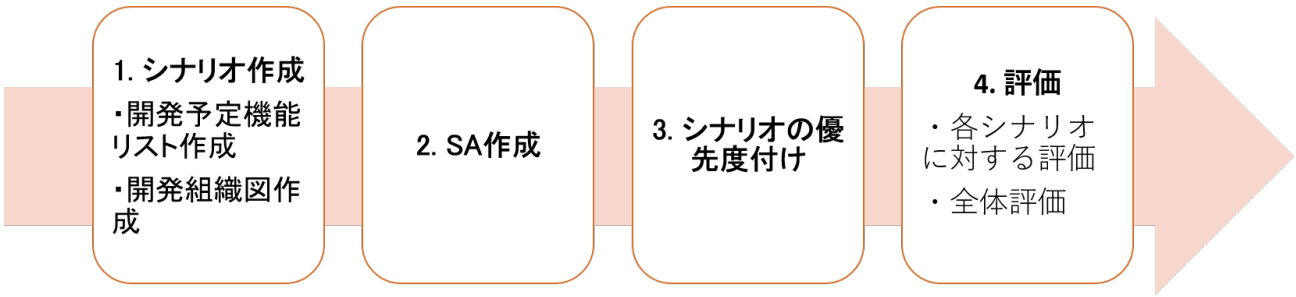


**図 14 SAAM for EA 概要**

### 3.5.2 評価シナリオの設計

評価シナリオは開発予定の機能リストを組み合わせ，開発組織を定義することで作成する（図 15）．作成された評価シナリオは各開発チームが開発を担当する機能を示す．これは複数の機能を並行開発するシナリオとなる．SAAM の評価結果は評価に使用するシナリオの質に影響を受ける課題がある．SAAM for EA は開発予定の機能リストと開発組織を事前に定義することで SAAM の課題であるシナリオ作成を補助する．

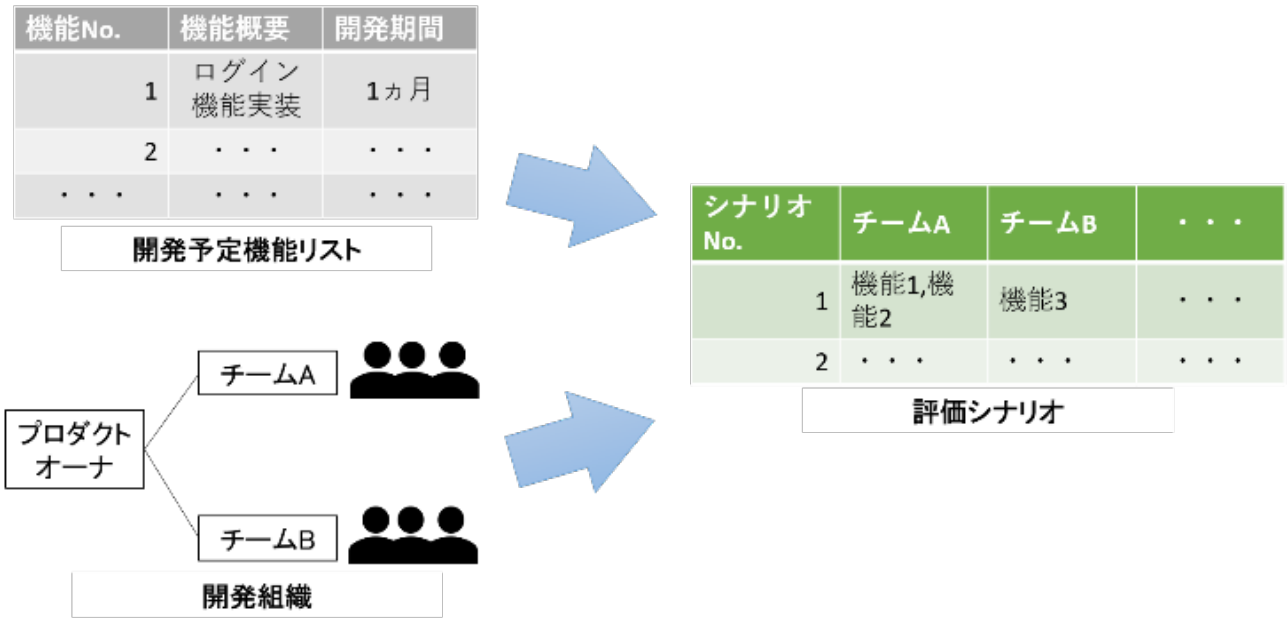


**図 15 評価シナリオの設計**

### 3.5.3 並行開発の可視化によるシミュレーション

SAAM for EA では開発の並行実行の視覚化のために並行開発シミュレーション図（図 16）を提案する．並行開発シミュレーション図はコンフリクトに着目することでコンフリクト解消作業の発生，すなわちコンフリクトを解消するまでの待ち時間の発生を可視化する．図中の横軸は時系列を示し，縦軸は開発予定の機能を示す．図 16 ではスプリント 1 において機能（1）は他の機能開発と依存することなくリリースできることを示す．機能（2），（3），(4)はスプリント 1 において依存関係があり待ち時間が発生することを示す．機能（5）は開発期間が 2 スプリント必要であり，機能（6）に依存していることを示す．機能（6）は開発期間が 2 スプリント必要であり，機能（5）に依存していることを示す．こちらも待ち時間が発生することがわかる．並行開発シミュレーション図を用い機能の開発順序を検討することでスケジュール遅延を最小限にする開発順序をシミュレーションすることができる．様々な開発順序を検討した結果，依存関係を最小化することが難しい場合は開発組織と SA のコンポーネントの粒度が合わない，すなわち同一 SA コンポーネントを更新する事象が頻発することを示す．この場合の対処として，SA のコンポーネントを開発組織に合う粒度に再設計することがある．

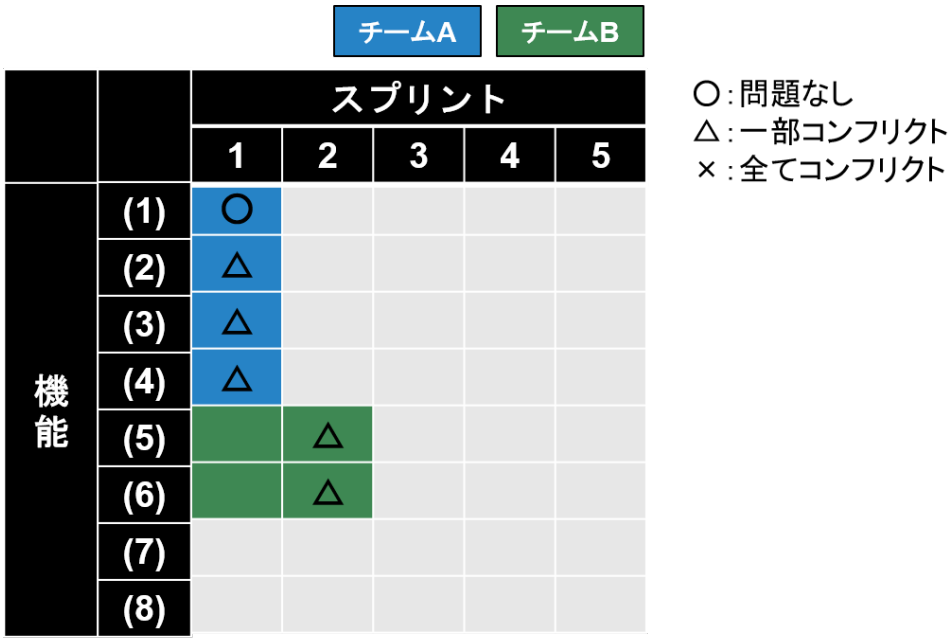


**図 16 並行開発シミュレーション図**

### 3.5.4 評価

評価作業は SAAM と同様に各シナリオに対する評価を実施することに加え，並行開発シミュレーション図を用いた評価を各シナリオに対して実施する．各スプリントでコンフリクトの発生の有無を確認し，待ち時間の発生を時間軸と共に評価する．コンフリクトが多く発生する場合はそのシナリオにおいて開発組織と SA のコンポーネントの粒度が合わないことを示す．

## 3.6 評価対象ソフトウェア

### 3.6.1 機能

評価対象は京都の観光情報を提供するスマートフォンアプリケーションとする[38]．スマートフォンアプリケーションが持つ主な機能を表 3 に示す．

**表 3 評価対象ソフトウェア機能リスト**

| No. | 機　能 |
|---|---|
| 1 | 地図が表示され，地図面の操作が可能 |
| 2 | 任意の地点への徒歩ナビゲーション |
| 3 | 京都市からのお知らせ情報ページ |
| 4 | 観光地の情報を地図上で確認できる |
| 5 | 近くの飲食店を検索することができる |

### 3.6.2 SA

MVC を用いた SA を図 17 に示す．Model を担当するコンポーネントを桃色，Controller を担当するコンポーネントを橙色，View を担当するコンポーネントを緑色で示した．VIPER を用いた SA を図 18，図 19 に示す．VIPER を用いた SA は図 18 に示すようにコンテキストが近いコンポーネントをまとめた．図 18，図 19 では MVC における Model 相当のコンポーネントを桃色，Controller 相当のコンポーネントを橙色，View 相当のコンポーネントを緑色で示した．

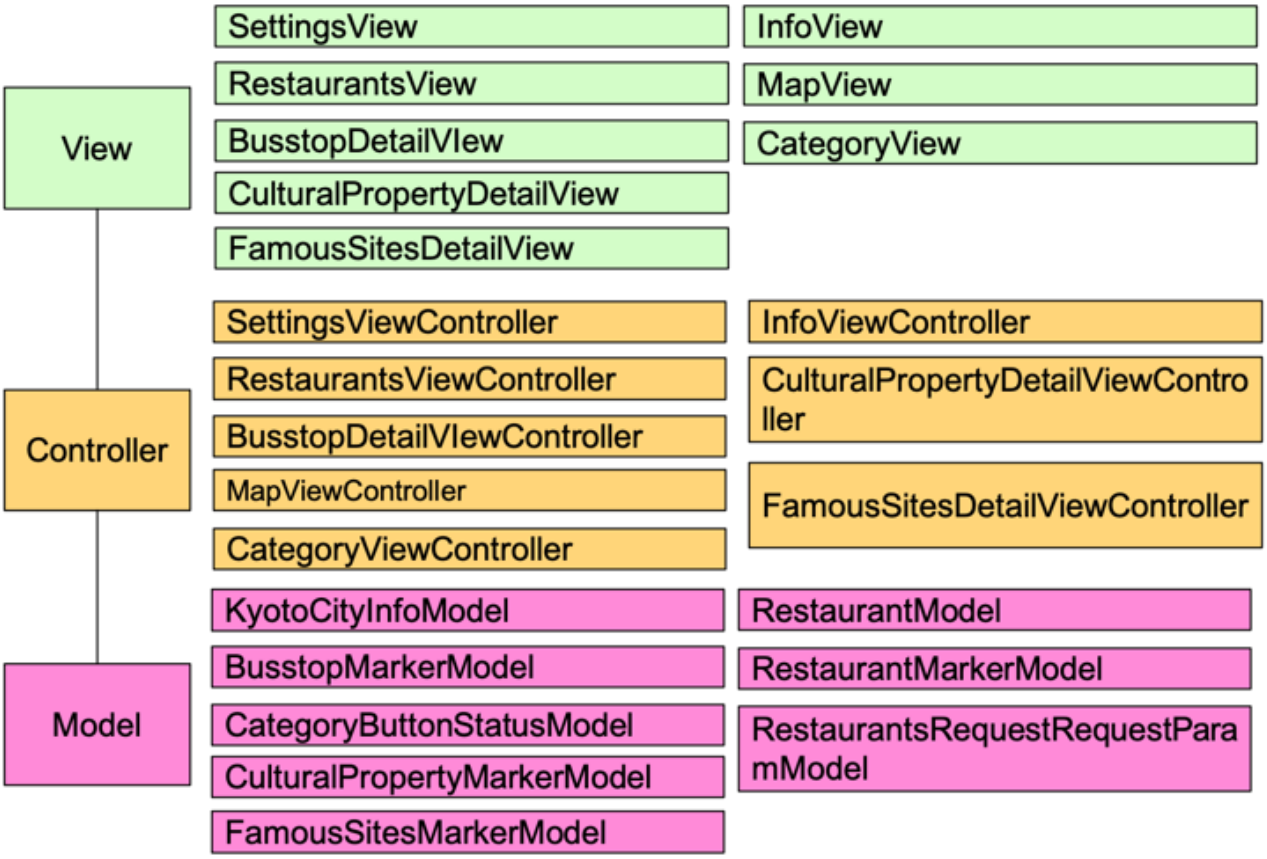


図 17 評価対象ソフトウェア（MVC）

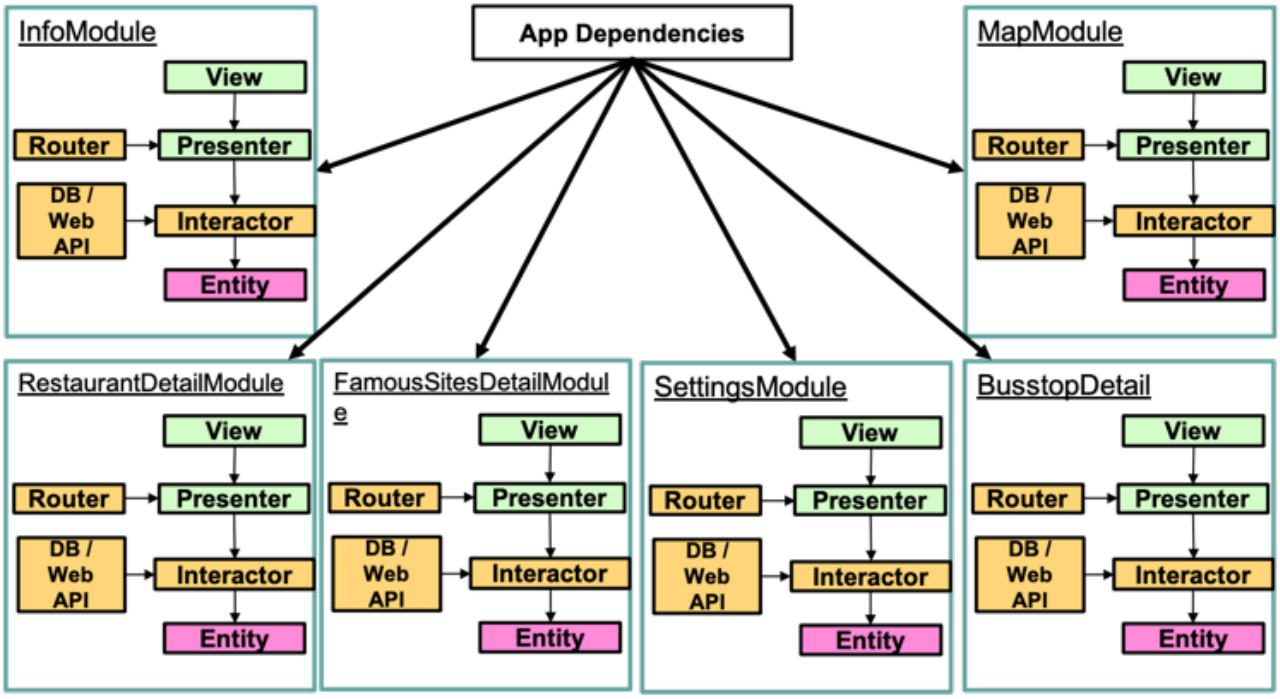


図 18 評価対象ソフトウェア（VIPER）

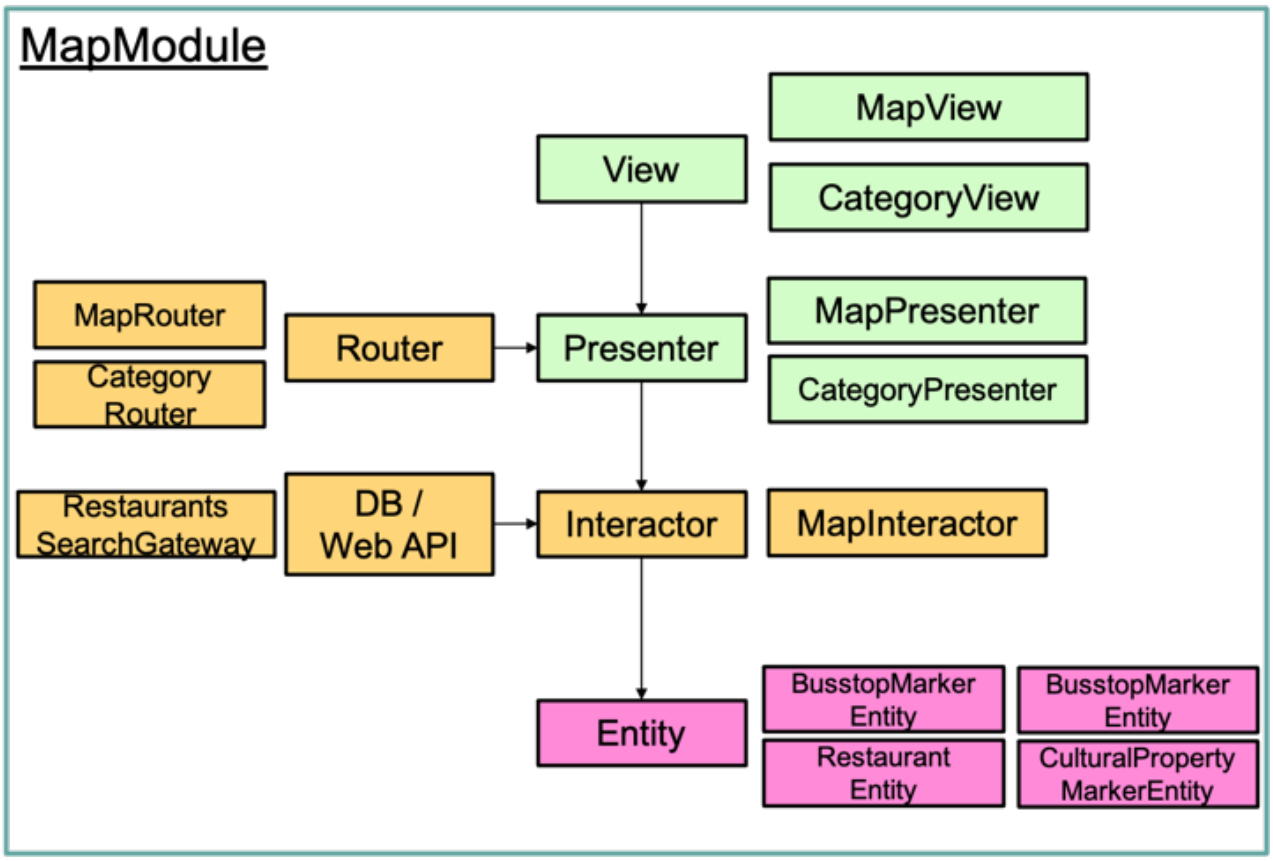


図 19 地図に関連するコンポーネント群（VIPER）

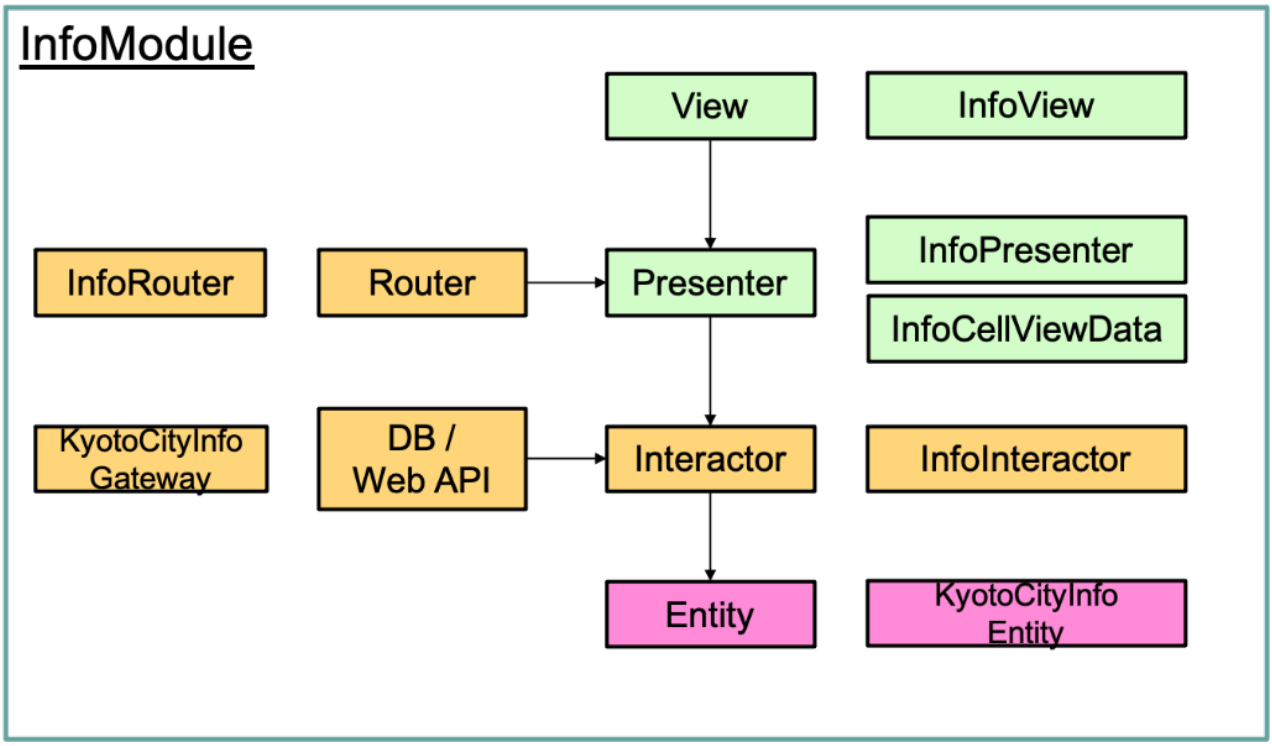


図 20 お知らせに関連するコンポーネント群（VIPER）

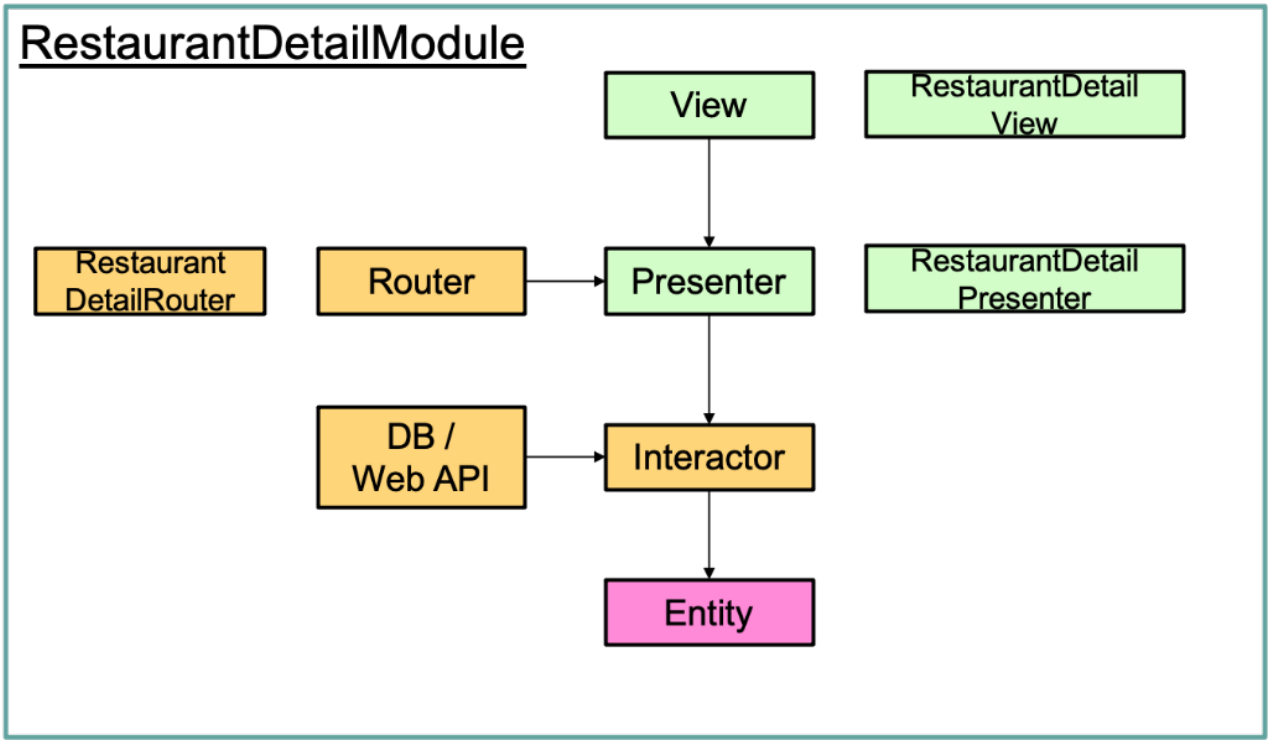


図 21 飲食店情報に関連するコンポーネント群（VIPER）

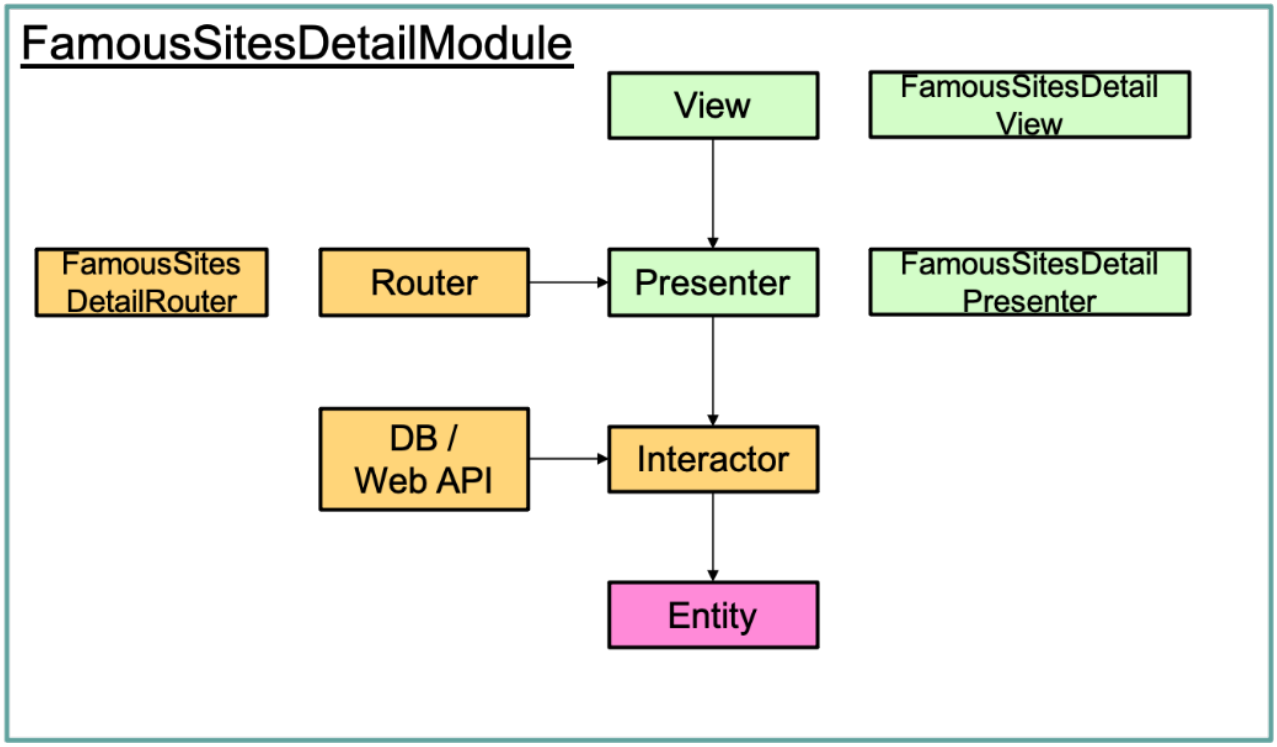


図 22 観光地情報に関連するコンポーネント群（VIPER）

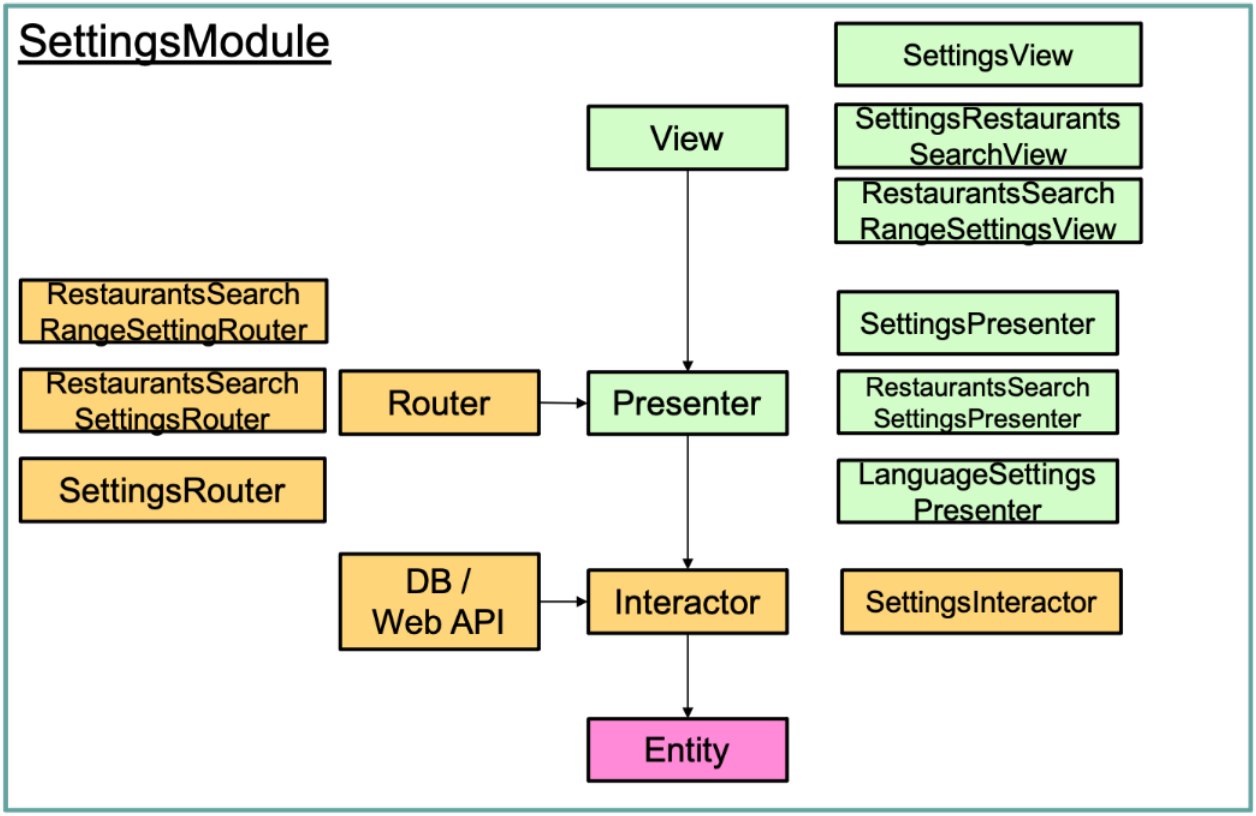


図 23 アプリ設定情報に関連するコンポーネント群（VIPER）

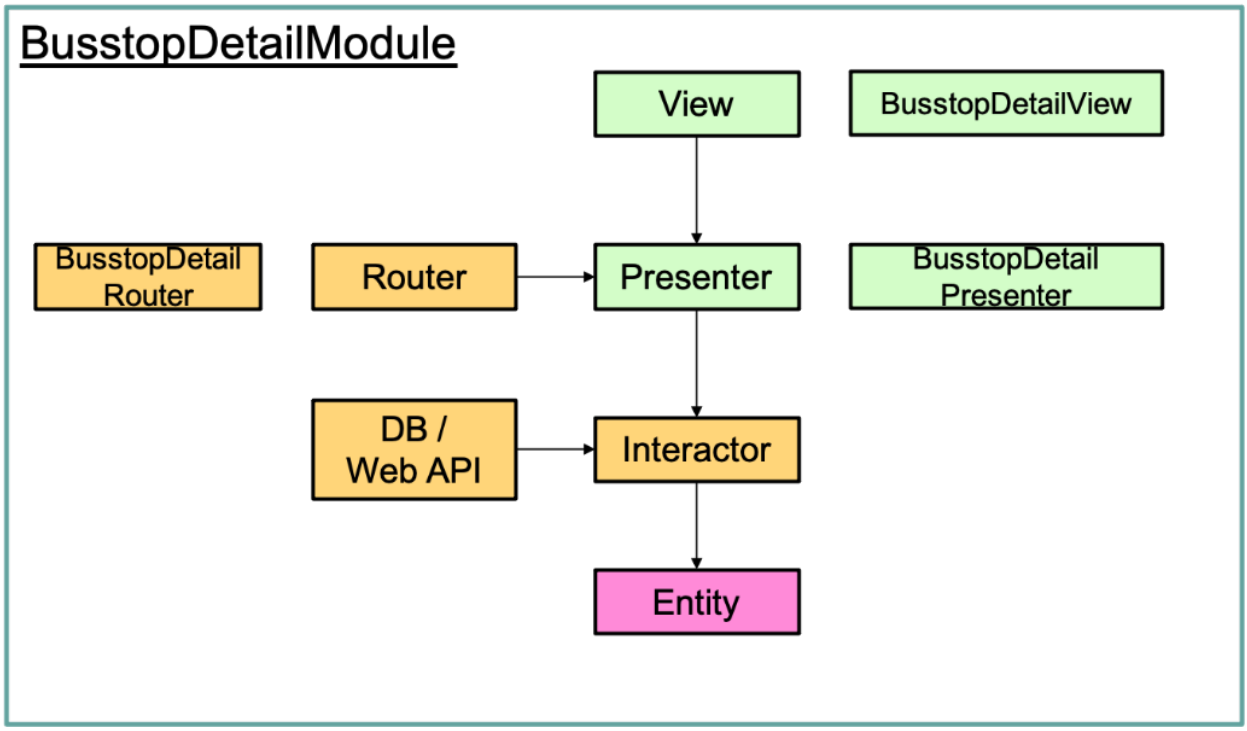


図 24 バス停情報に関連するコンポーネント群（VIPER）

地図機能に関係するコンポーネントに着目することで MVC と VIPER の SA の違いを確認する．MVC における Controller のコンポーネントは MapViewController のみである（図 17）．VIPER における Controller のコンポーネントは MapInteractor，MapRouter など複数存在する（図 19）．このことから VIPER は MVC と比べて各コンポーネントの責務が小さくなることがわかる．

## 3.7 評価

SAAM for EA を用いた評価を評価対象ソフトウェアに対して実施する．図 25 に示す開発組織（LeSS Huge）において表 4 に示す機能リストを組み合わせて作成した評価シナリオ（表 5）を用い評価を行う．エリアは大規模なアジャイル開発を行うために LeSS Huge が定義する概念である．開発チームはエリアに所属し，担当する機能の開発を進める．3.5.2 で述べたように開発組織（図 25）と開発予定の機能リスト（表 4）を組み合わせることで複数の機能を並行開発する評価シナリオとなる．なお，表 4 に示した開発予定期間は 1 スプリントを 2 週間として算出した．

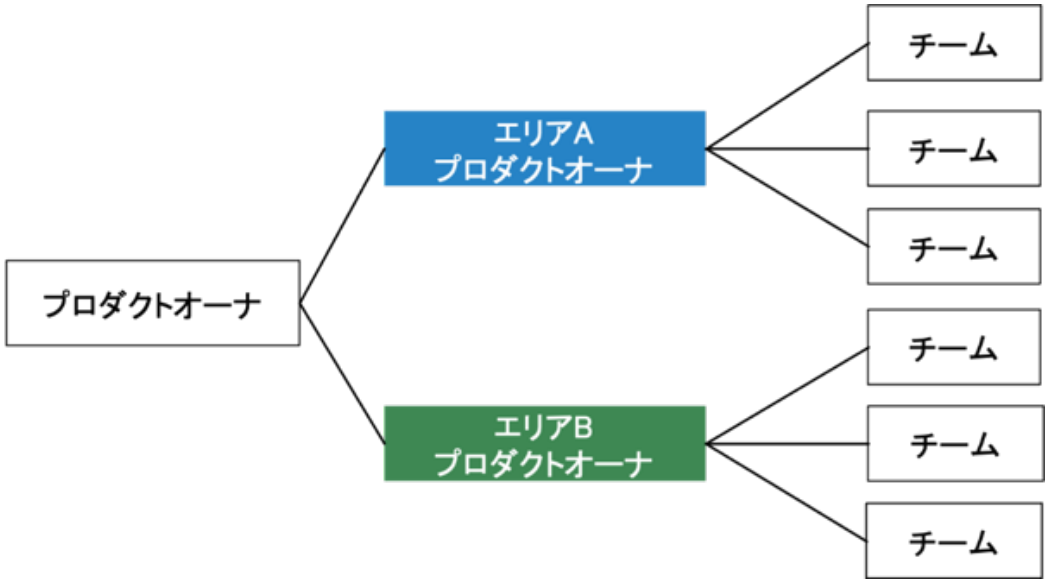


**図 25 評価シナリオ（開発組織）**

**表 4 評価シナリオ（開発予定の機能）**

| 機能番号 | 機能 | 開発予定期間 (スプリント) |
|---|---|---|
| (1) | イベントの情報をプッシュ通知で配信する | 1 |
| (2) | お気に入りの観光地をローカルストレージに保存可能 | 1 |
| (3) | ユーザ登録およびログインができる | 1 |
| (4) | 地図デザインの変更 | 1 |
| (5) | WebAPI（飲食店情報検索 API）の切り替えおよび仕様変更への対応 | 2 |
| (6) | 観光地の検索機能 | 2 |
| (7) | アプリ起動時にお知らせダイアログを表示する | 1 |
| (8) | 地図 SDK の変更 | 3 |

表 5 評価シナリオ

| シナリオNo. | エリアA | エリアB |
|---|---|---|
| 1 | (1), (2) , (3) , (4)を開発 | (5), (6)を開発 |
| 2 | (1), (2), (3), (4), (7)を開発 | (5), (6)を開発 |
| 3 | (1), (3), (5)を開発 | (2), (4), (6)を開発 |
| 4 | (1), (2), (3), (7)を開発 | (4), (8)を開発 |
| 5 | (1), (3), (5)を開発 | (4), (8)を開発 |

評価前の各シナリオの並行開発シミュレーション図を図 26 から図 30 に示す. MVC と VIPER それぞれの場合において評価を実施する.

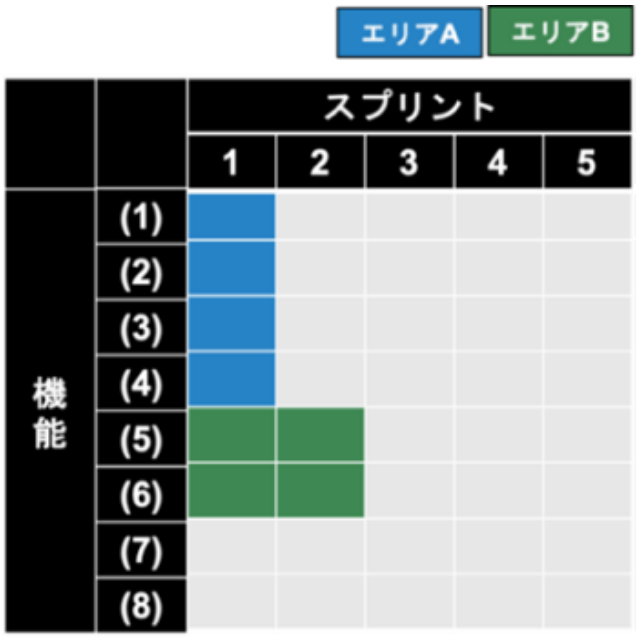


図 26 並行開発シミュレーション図（シナリオ1）

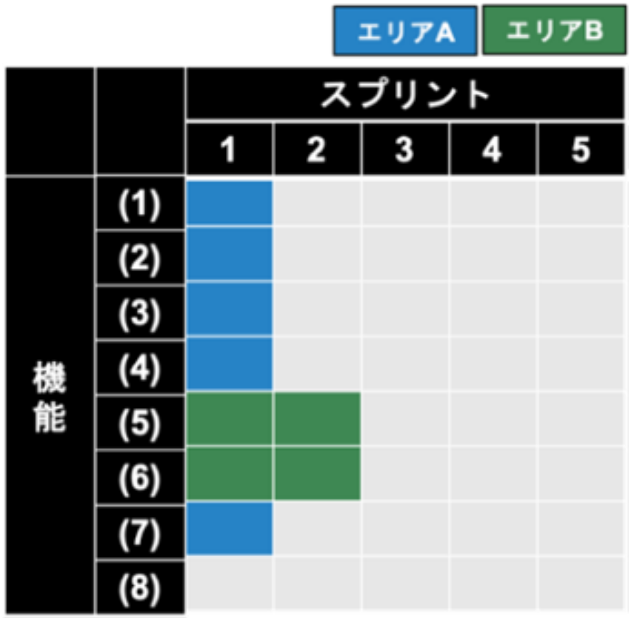


図 27 並行開発シミュレーション図（シナリオ2）


図 28 並行開発シミュレーション図（シナリオ3）

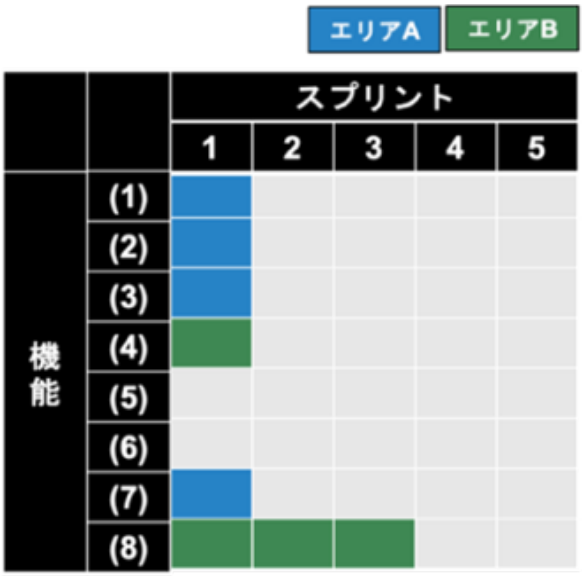


図 29 並行開発シミュレーション図（シナリオ4）

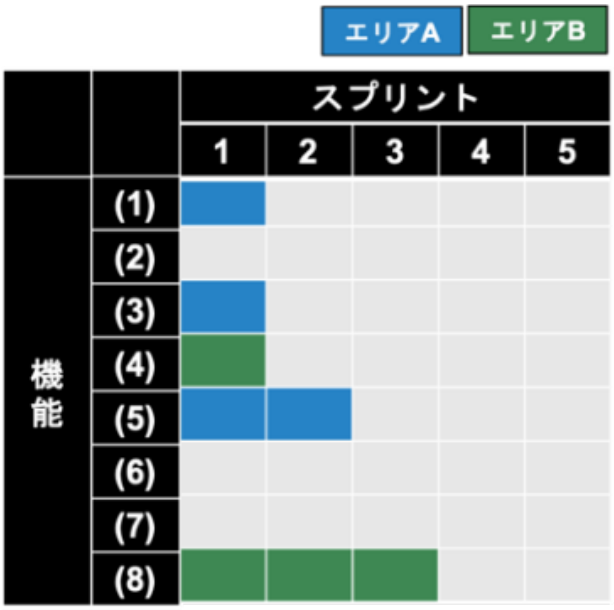


図 30 並行開発シミュレーション図（シナリオ5）

## 3.8 評価結果

### 3.8.1 SA 評価結果

図 31 から図 35 に MVC における各シナリオの評価結果を示す．表中，評価欄〇はコンフリクトが未発生，△は一部クラスファイルのコンフリクト発生，×は全てのクラスファイルのコンフリクト発生を示す．

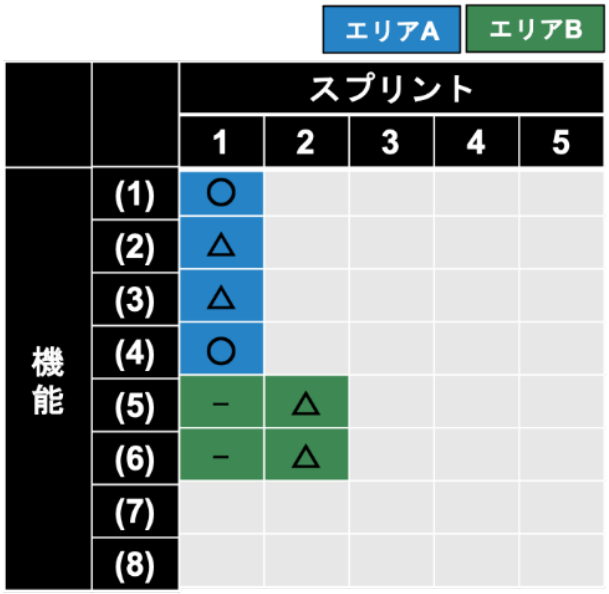


図 31 並行開発シミュレーション図(MVC シナリオ 1)

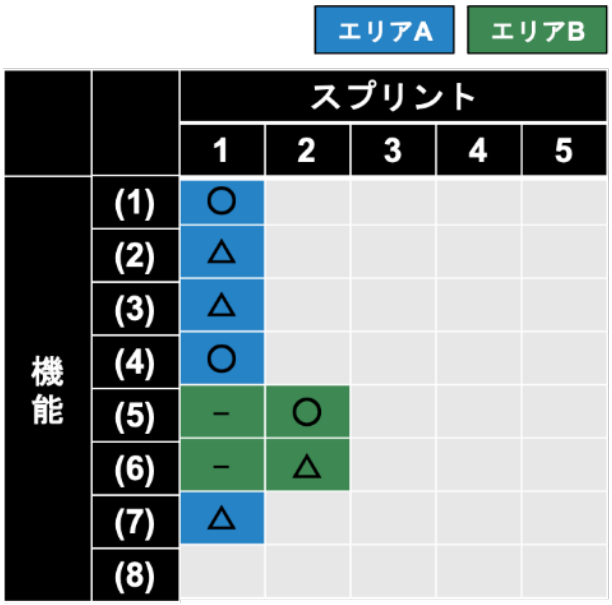


図 32 並行開発シミュレーション図(MVC シナリオ 2)

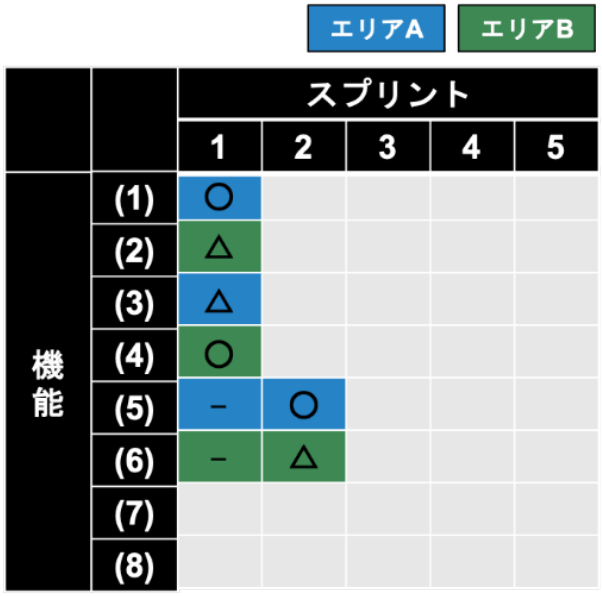


図 33 並行開発シミュレーション図(MVC シナリオ 3)

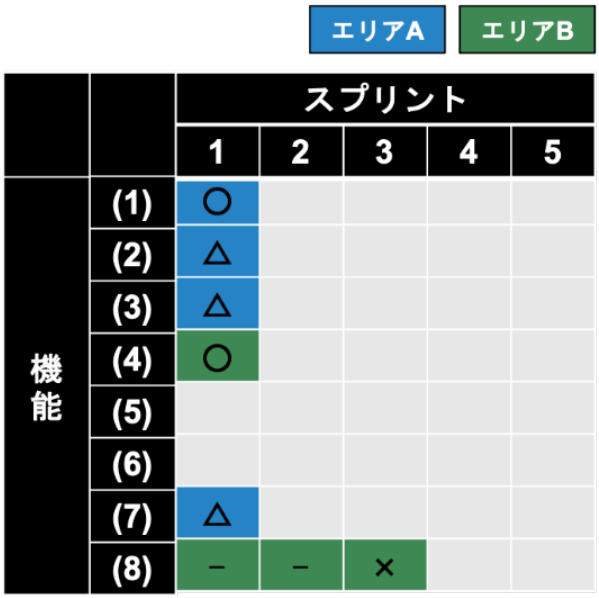


図 34 並行開発シミュレーション図(MVC シナリオ 4)

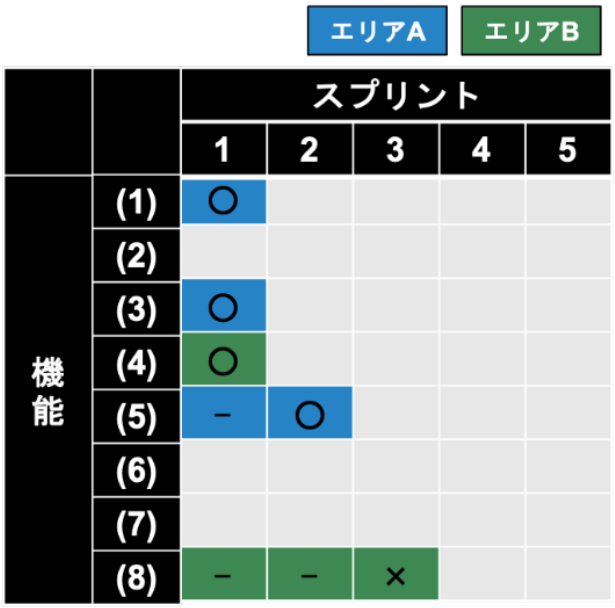


**図 35 並行開発シミュレーション図(MVC シナリオ 5)**

図 36 から図 40 に VIPER における各シナリオの評価結果を示す．表中，評価欄○はコンフリクトが未発生，△は一部クラスファイルのコンフリクト発生，×は全てのクラスファイルのコンフリクト発生を示す．

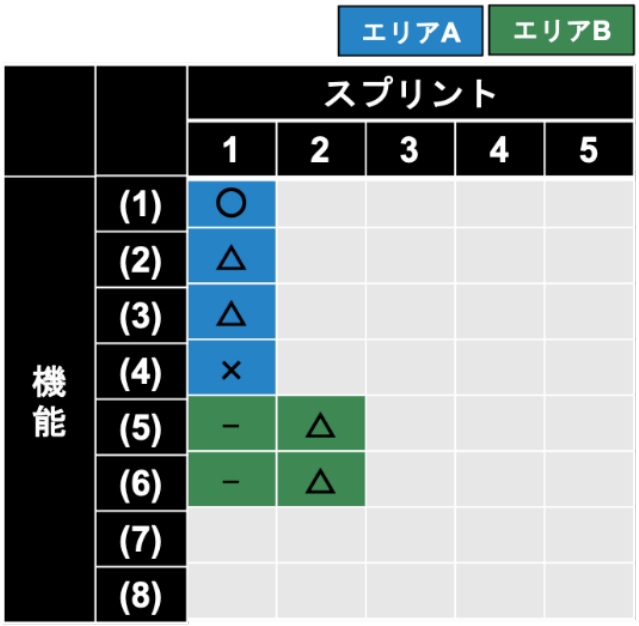


**図 36 並行開発シミュレーション図（VIPER シナリオ 1）**

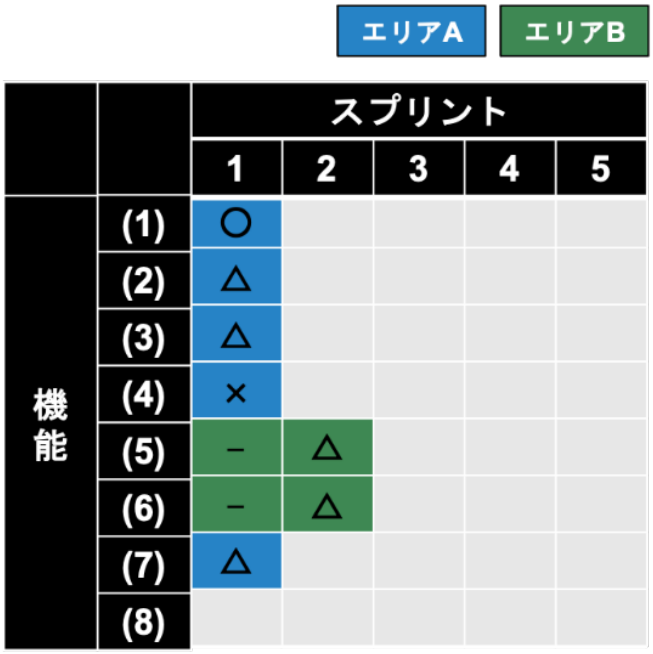


図 37 並行開発シミュレーション図（VIPER シナリオ 2）

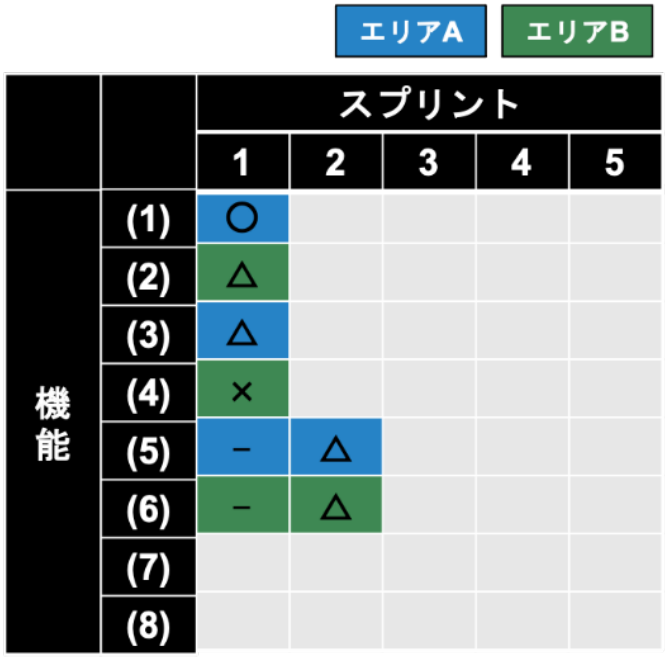


図 38 並行開発シミュレーション図（VIPER シナリオ 3）

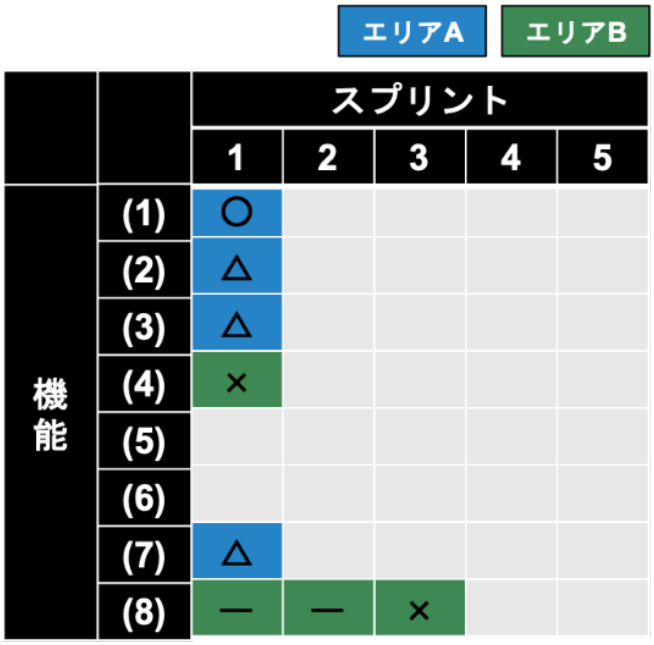


図 39 並行開発シミュレーション図（VIPER シナリオ 4）

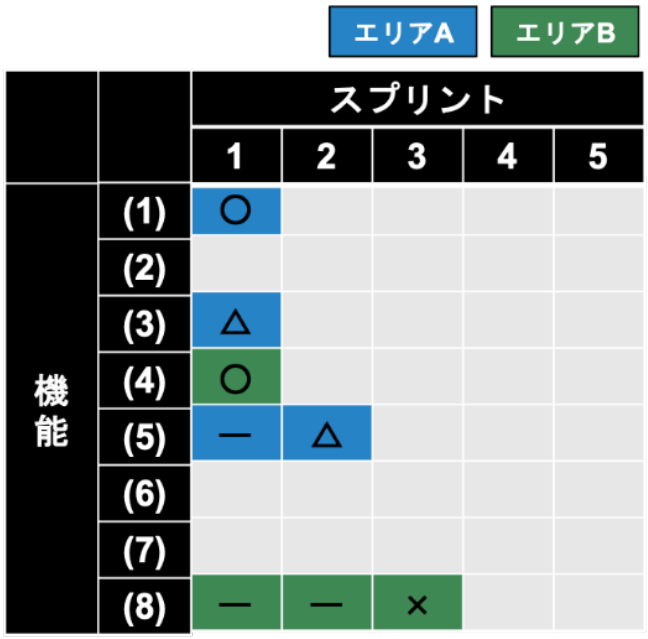


図 40 並行開発シミュレーション図（VIPER シナリオ 5）

表 6 にこれらをまとめた結果として各シナリオの評価結果を示す．表中，評価欄Δは一部クラスファイルのコンフリクト発生を示す．表 6 から全てのシナリオでコンフリクトが発生していることがわかる．

表 6 機能リリース時のコンフリクト比較

| シナリオ No. | MVC | | VIPER | |
|---|---|---|---|---|
| | 評価 | コンフリクトするクラス | 評価 | コンフリクトするクラス |
| 1 | △ | MapViewController<br>MapView | △ | MapPresenter<br>MapInteractor |
| 2 | △ | MapViewController<br>MapView | △ | MapView<br>MapPresenter<br>MapInteractor |
| 3 | △ | MapViewController | △ | MapPresenter<br>MapInteractor |
| 4 | △ | MapViewController | △ | MapView<br>MapPresenter |
| 5 | △ | MapViewController | △ | MapPresenter |

図 41 にシナリオ4における各コンポーネントの変更範囲を示す. MVC ではMapViewControllerが変更対象のクラスになっており，VIPER では MapView と MapPresenter が変更対象となっている．これは VIPER では各コンポーネントの責務が小さくなっていることが原因である．責務の小さいコンポーネントで発生しているコンフリクトは解消が容易である．同様のことがシナリオ 2 とシナリオ 3 においても確認できた.

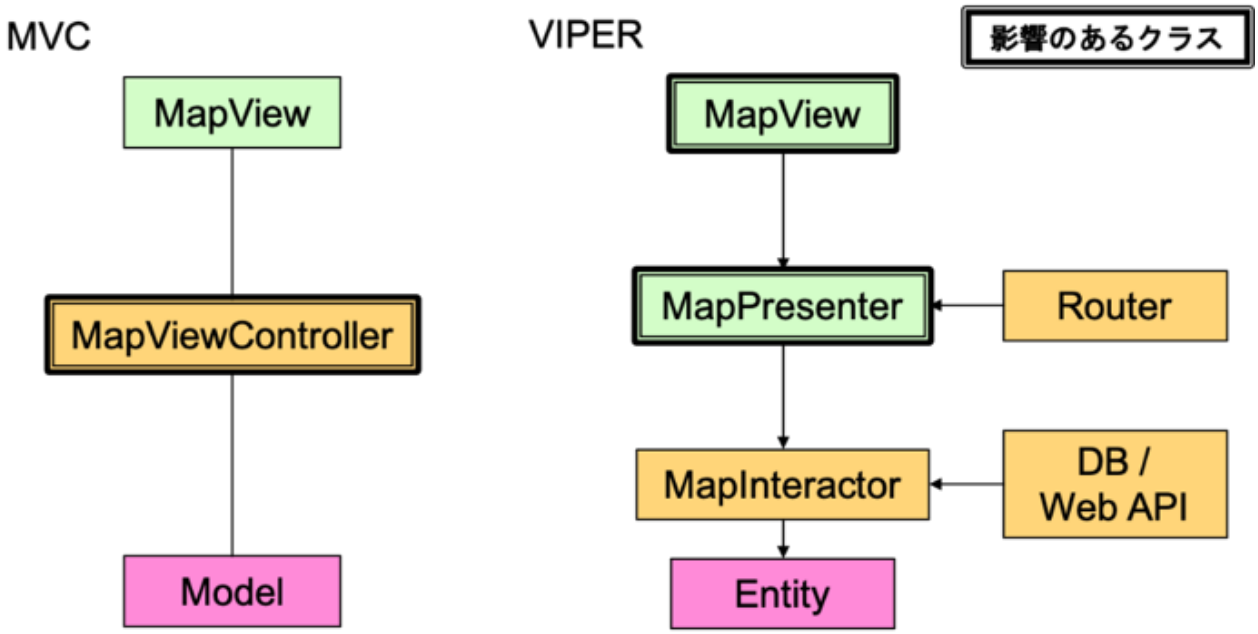


図 41 シナリオ 4 における比較

これらを踏まえ，評価結果を表 7 に示す．表中の+は相対的に変更が容易であることを示し，-は相対的に変更が困難であることを示す．0 は差がないことを示す．VIPER がシナリオ 2，3，4 において MVC よりも良い結果となった．なお，シナリオ 1 とシナリオ 5 については差を確認することはできなかった.

表 7 SA 評価結果

| シナリオNo. | MVC | VIPER |
|---|---|---|
| 1 | 0 | 0 |
| 2 | - | + |
| 3 | - | + |
| 4 | - | + |
| 5 | 0 | 0 |

### 3.8.2 並行開発シミュレーション図による評価結果

図 42 にシナリオ 4 における MVC と VIPER の並行開発シミュレーション図を示す．表中評価欄は○をコンフリクトなし，△を一部クラスファイルのコンフリクト発生，×を全てのクラスファイルのコンフリクト発生を示す．シナリオ 4 のスプリント 1 において MVC は機能(4)のリリースが VIPER よりも容易であった．

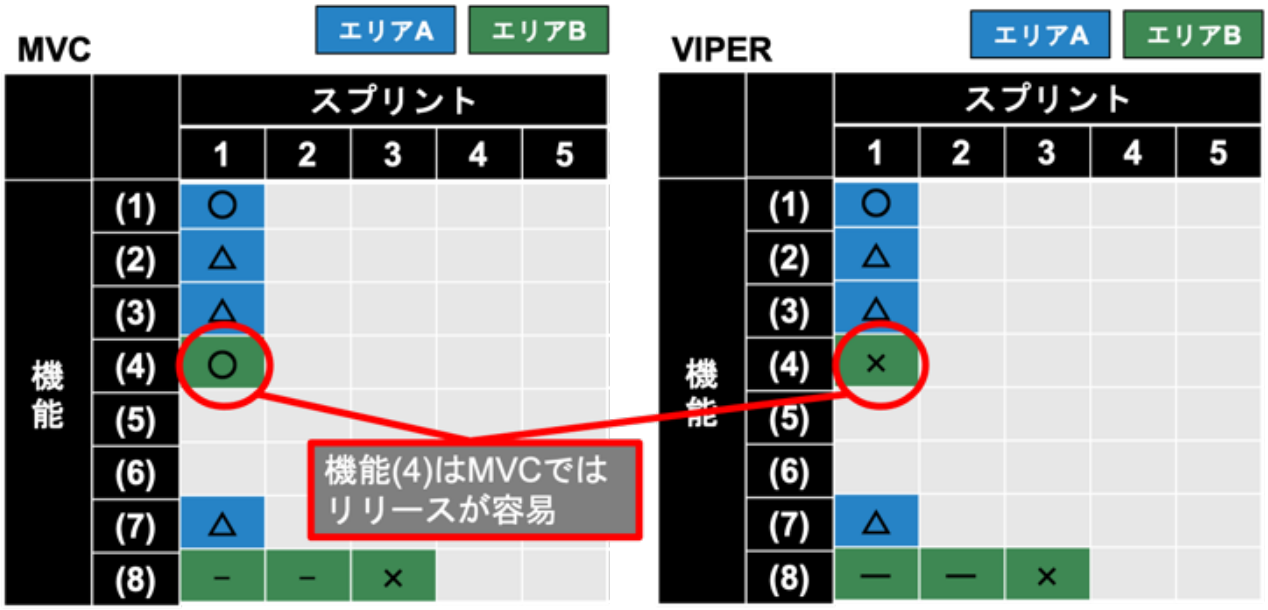


図 42 シナリオ 4 における並行開発シミュレーション図

図 43 は地図機能に関係するコンポーネントを示す．図中左側は MVC，右側は VIPER の SA である．機能(2)に関する変更箇所に着目する．MVC では変更箇所は MapViewController のみであり，VIPER では変更箇所は MapView，MapPresenter，MapInteractor など複数のコンポーネントに存在することがわかる．その結果，機能(2)の変更箇所は機能(4)の変更箇所と重なることとなった．VIPER は各コンポーネントの責務が小さく設計されているので複数のコンポーネントへ修正が発生する．これはコンフリクトが発生する原因となる．他のシナリオにおいても MVC の方がコンフリクトを少なくリリースできるケースを確認した．その理由はシナリオ 4 におけるケースと同様である．

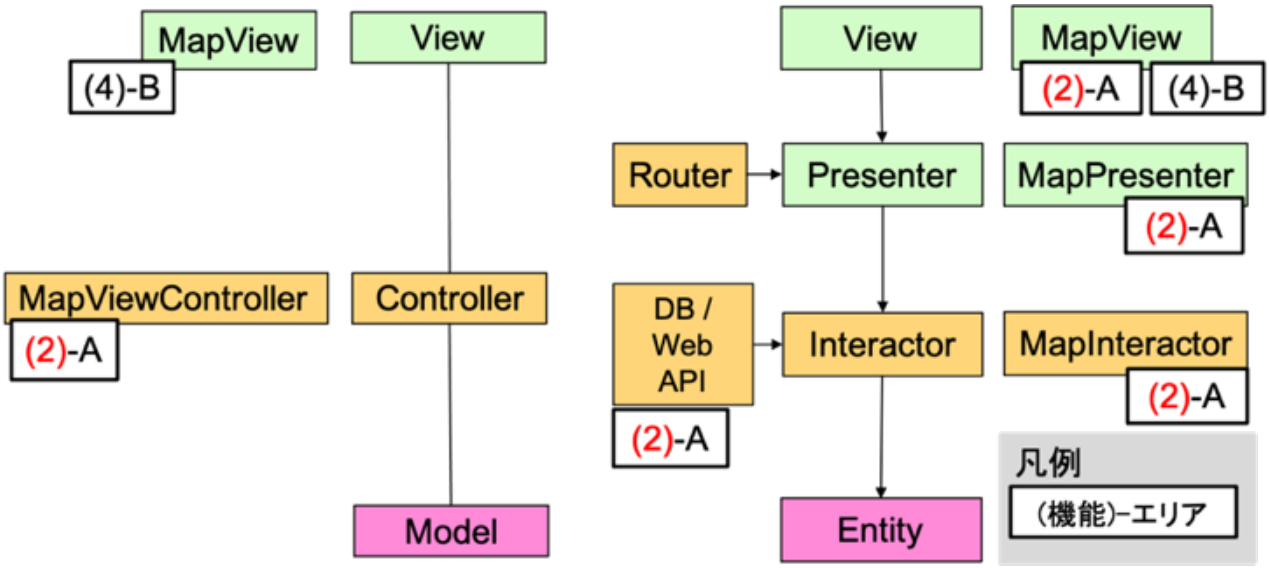


図 43 シナリオ 4 スプリント 1 における変更箇所の比較

## 3.9 考察

### 3.9.1 研究課題(1)に対する考察

本論文は MVC と VIPER を比較する評価を実施した．MVC は VIPER と比較してコンポーネントの粒度が大きいため VIPER よりモノリシックな SA であると捉えることができる．モノリシックな SA は EA 開発に適さないことが EA 開発を実践する組織への調査から報告されている[24]．表 5 に示す評価結果はモノリシックな SA（MVC）はコンポーネントの粒度が小さい SA（VIPER）と比較して EA 開発に適さないことを示した．このことから EA 開発に適した SA 評価方法に必要な特性の一つは，開発組織と SA のコンポーネントの構造が適合するかを評価できることと考えられる．

提案方法を用いることで実際にモノリシックな SA は EA 開発に適さないことを確認できた理由は，並行開発シミュレーション図がコンフリクト，すなわち SA コンポーネント間の依存関係の存在に着目することで待ち時間発生を評価することによる．開発スケジュールを遅延させる待ち時間の発生はコンフリクト発生，すなわち依存関係の存在を原因とし，依存関係の存在は開発組織と SA の不適合を原因とする．提案方法を用いることで，開発組織は依存関係の少ない SA を選択することができるので，開発スケジュールへの影響を軽減できる．SA のみで対処が難しい依存関係も存在するので，EA 開発フレームワークを導入し，Bick らが示すように依存関係の存在を気づけるよう開発プロセスを工夫することが必要である[31]．

図 45 に開発組織と SA コンポーネントの関係を示す．図の仮想組織は開発プロセスフレームワークにおいて予め定義されている．仮想組織レイヤの各コンポーネントはチーム数やチームに所属するメンバ数が潜在的に開発可能なアーキテクチャ上のコンポーネントの数量を表す指標と一対で定義されている．他方，SA レイヤの各コンポーネントもその開発規模を表す指標，例えば，オブジェクトポイントやファンクションポイント（FP）などと併せて定義されている．開発組織（仮想組織）と SA の不適合は両者のトポロジと各コンポーネントの重みとを総合的に考慮し，解消する．例えば，複数チームで構成される複雑な仮想組織のトポロジと単純なトポロジを持つモノリシックな SA の場合を考える．この場合，極論すれば単一コンポーネントをチーム総体で開発することになり，待ち時間が頻繁に発生し，非効率な開発となる．対して，マイクロサービスアーキテクチャの技術を用い SA コンポーネントを分割することで両者のトポロジを適合させ，仮想組織と SA の不適合を解消する．さらに，アーキテクチャレベルでのトポロジ照合が難しい場合は，アーキテクチャに基づくソフトウェア設計を行い，その設計上のコンポーネント構成のトポロジとの一致を図る．

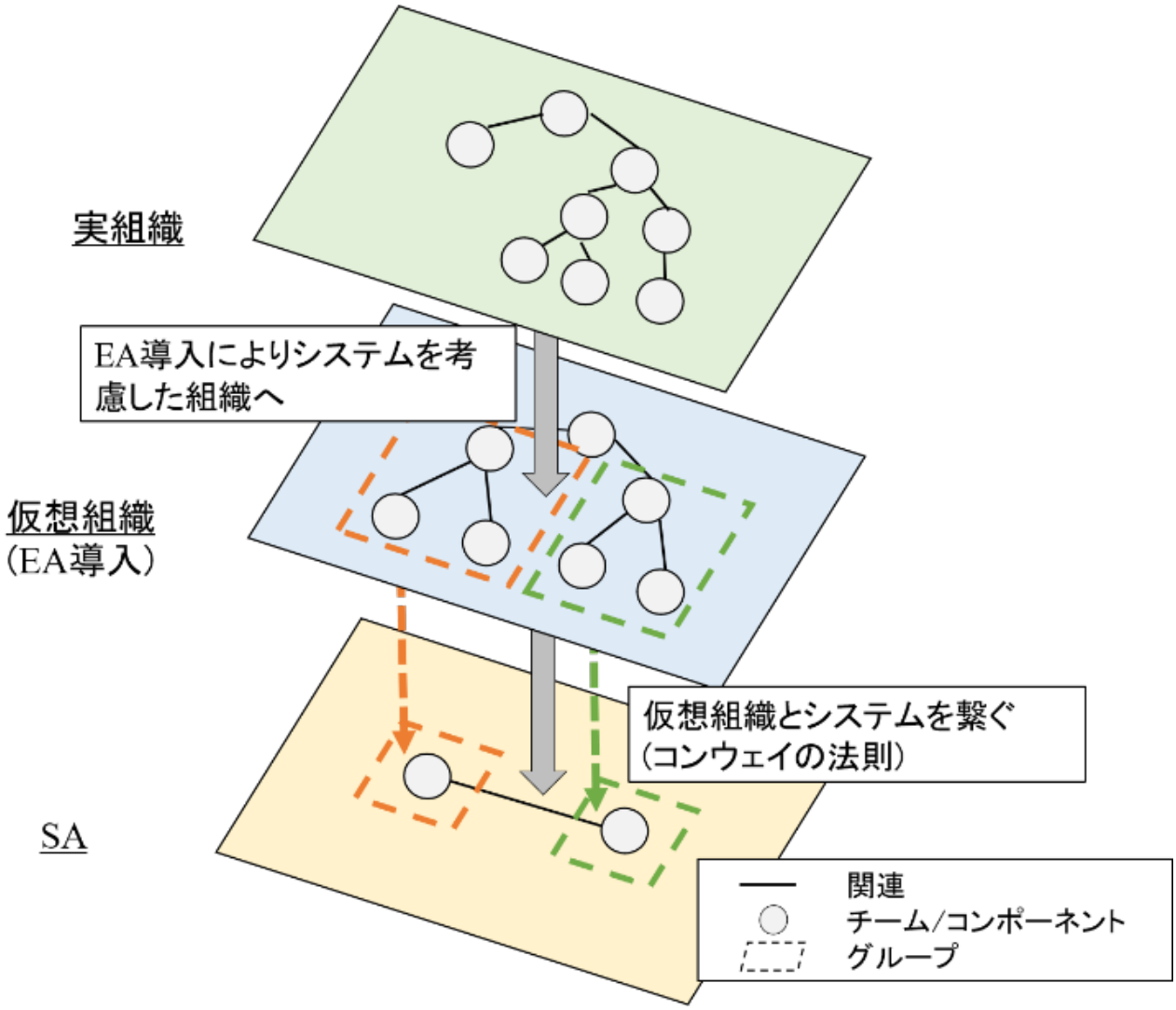


**図 44 EA と SA の関係**

### 3.9.2 研究課題(2)に対する考察

本論文提案の並行開発シミュレーション図はコンフリクトに着目することで待ち時間の発生，すなわち依存関係の存在を可視化する．これは EA 開発においてプロダクトマネージャがリリーススケジュールを検討する時に必要とする情報と同様である．図 11 で示したようにプロダクトマネージャの役割は優先度付けとリリース戦略の策定である．プロダクトにとって最適なリリーススケジュールを導き出すために，プロダクトマネージャは複数のシナリオを比較する．このとき，プロダクトマネージャは開発を進める機能間の依存関係を考慮し最適なシナリオを選択する．これはコンフリクト発生により生じる待ち時間発生を抑え，開発スケジュールの遅延を防ぐ．このことから，並行開発シミュレーション図を用い待ち時間の発生を可視化することで EA 開発においてプロダクトマネージャが担うマネジメント作業を支援可能と考える．

### 3.9.3 EA 開発フレームワークの選択と SA

同じシナリオにおいて MVC と VIPER で結果が異なるケースを確認した．これは開発組織と SA のコンポーネントの粒度の違いによる結果と考えられる．図 45 に開発組織と SA のコンポーネントの繋がりを示した．設計を行う組織の構造がシステムの構造に反映されること[14]を考慮すると，EA 開発フレームワークの導入はシステムの構造に合わせた仮想組織の構築手段と考えられる．SAFe5.0 は実際の組織を第 1 の組織，EA 組織を第 2 の組織（仮想組織）と定義する[30]．EA 開発フレームワーク選択時に重要なことは構築した仮想組織に合う SA を利用することである．仮想組織と SA の適合度合いは提案方法を用いることで評価が可能であるので，提案方法は EA 開発フレームワーク導入時の効果検証において有用と考えられる．

### 3.9.4 組織構造と SA

3.9.3 で述べたように開発組織と SA のトポロジを合わせることが EA の選択時に重要である．このとき, SOA を用いて SA を表現し, 仮想組織との対応関係を整理することを考える（図 45）．SOA を導入できておらず SA を分割できていない場合(モノリシックアーキテクチャ)はそれに合わせ一つのアジャイル開発チームとすることがリリースマネジメントの課題を最小限に抑えると考える（図 46）．マイクロサービスアーキテクチャを導入している場合は，より大規模な EA を導入できる可能性がある（図 47）．アプリケーションフロントエンドはドメインの概念を導入した Clean Architecture などを導入することで，同様のアプローチが可能と考えられる（図 48）．本論文の評価作業はスマートフォンアプリケーション（アプリケーションフロントエンド）に対してドメインを考慮した SA である VIPER を適用していることから図 48 のケースである．

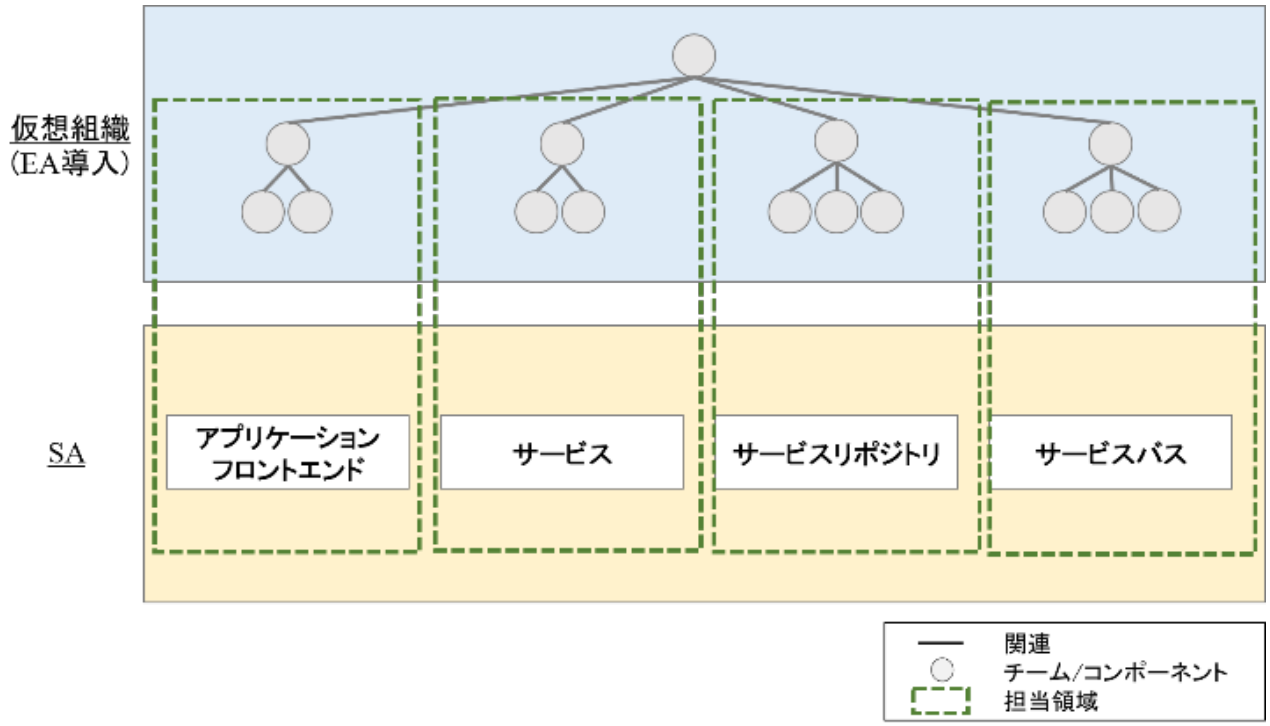


**図 45 組織と SA（SOA）**


図 46 組織と SA（モノリシックアーキテクチャ）

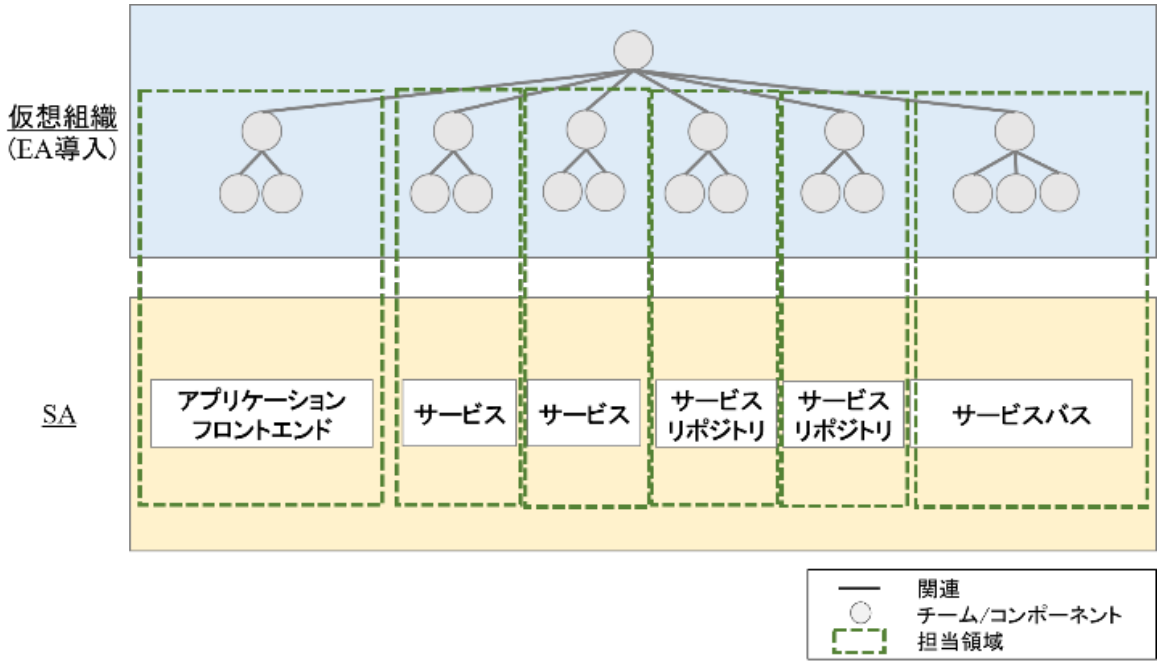


図 47 組織と SA（SOA）

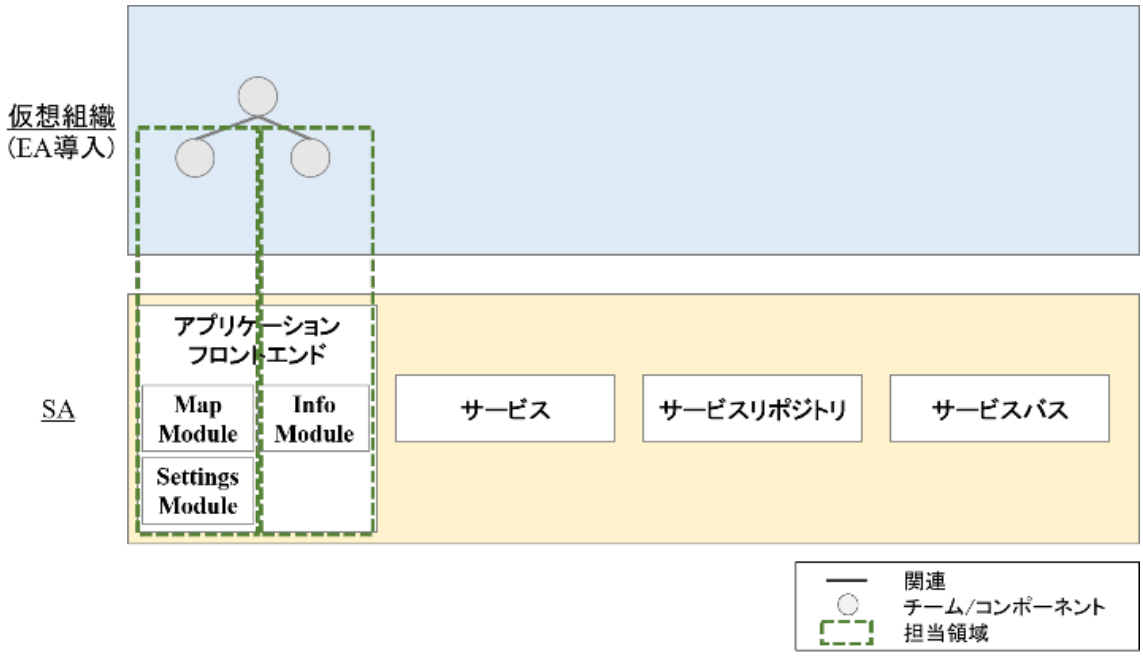


**図 48 組織と SA（Clean Architecture）**

Nord らは開発組織とアーキテクチャの関係を整理し，それを考慮したエンタープライズアジャイル開発を導入する戦略をまとめた[28]．SOA を用いているので Web システムを扱う組織に限られるが，実組織，仮想組織，そしてアーキテクチャを見通すことができる．以上から SOA を EA 導入に適用することで開発組織と SA コンポーネントの関係を適切に繋ぐことが見込める．

## 3.10 検証上の問題点と残された課題

本章では，提案方法に対し評価シナリオを用いた定性的な検証を実施した．評価シナリオは用意した評価対象ソフトウェアを用いた複数チームでのアジャイル開発を行うシナリオであった．十分実用的であるが，提案方法を用いた定量的な事例検証が不足している．特に様々な規模のソフトウェア，アーキテクチャ，組織における事例検証が必要である．

## 3.11 まとめ

エンタープライズアジャイル開発を行う際の課題として，チーム間の連携，すなわち作業の依存関係の問題がある[31][42]．本章では，依存関係のある作業として同一コンポーネントの更新作業に着目した．同一コンポーネントの更新作業が発生する時，その整合性を取る作業，すなわちコンフリクト解消作業の発生に着目し SAAM for EA を提案した．開発組織と SA の不適合を原因とするコンフリクト発生，すなわち依存関係の存在によって開発作業は中断（待ち時間の発生）し，開発スケジュールは遅延する．提案方法は開発組織と SA コンポーネントの適合度合いを評価し，依存関係のある作業が発生することが少ない SA を選択することを支援するので，EA 並行開発時の開発スケジュール遅延を軽減する．

スマートフォン向けアプリケーションへ提案方法を適用しその有効性の評価を行なった．開発スケジュールに影響を与えるコンフリクトおよび待ち時間の発生が少ない SA が EA 並行開発に適するという観点から評価した結果，実践事例の報告[24]および 2.7.4 で述べた本論文の実践事例で確認した通りモノリシックな SA，すなわち依存関係が存在する作業が多く発生する SA またはソフトウェア設計は開発スケジュールへの影響があるのでエンタープライズアジャイル開発に適さないことを示した．これは Nord らが提案するようにレイヤードアーキテクチャの方がエンタープライズアジャイル開発に適する，という提案とも一致する[28]．Nord らは開発組織とアーキテクチャの関係をまとめ，エンタープライズアジャイル開発に携わるアーキテクト向けのガイドラインを示した[28]．Nord らはアーキテクチャ上の依存関係の存在を分析する方法の研究を今後の課題とした．本章で提案した方法は同一コンポーネントの更新作業に着目したことでその依存関係の存在を分析する方法であり，チーム間の連携に問題が発生するかを分析する方法である．

コンフリクトの原因となる開発組織と SA コンポーネントの粒度の不適合は開発スケジュールの遅延という課題につながる．並行開発シミュレーション図はコンフリクトに着目することで作業間の依存関係の存在を示し，さらに待ち時間の発生を可視化する．エンタープライズアジャイル開発を導入する組織は提案方法を用い複数のシナリオを検討することで開発組織と SA コンポーネントの粒度の適合度合いを評価可能となる．評価結果を用い，開発組織に合う SA を選択することは依存関係のある作業の発生を抑え，さらに待ち時間の発生を軽減するのでスケジュールの遅延を防ぐ．

# 4 エンタープライズアジャイル並行開発に適したソフトウェア開発法の提案

組織とアーキテクチャの関係[28]，さらに人員や知識，作業間の依存関係[9]を考慮することが複数チームでのアジャイル開発を行う時に必要である．依存関係への対処はチーム間の連携の発生頻度を抑え，円滑にエンタープライズアジャイル開発を行うことへつながる[42]．本論文では 3 章で作業間の依存関係を分析することでエンタープライズアジャイル開発に適するソフトウェアアーキテクチャ評価する方法を示した．依存関係の分析とその対処方法に関する議論はない[28].

本章では，作業間の依存関係へ対処するソフトウェア開発法を提案する．組織とフレームワークを用いて構築された仮想組織，アーキテクチャの関係を整理した．さらに，仮想組織とアーキテクチャの関係をパターン化することでソフトウェア開発法として作業間の依存関係を対処する方法を示す．提案方法を用いることで，チーム間の連携の発生が抑えられ，複数チームでのアジャイル開発を円滑に進めることへつながる．

## 4.1 研究の背景

3 章で EA 開発の実践とエンタープライズアジャイルに適する SA 評価方法の設計に取り組んできた．提案した SA 評価方法 SAAM for EA は，EA 開発実践時に課題となる開発組織と SA の不適合を同一コンポーネントの変更作業の発生，すなわち依存関係のある作業の発生に着目し構築した．提案方法を用いた事例検証からその妥当性を 3.9 で示し，開発組織の構造と SA またはソフトウェア設計を合わせる必要性を確認した．SAAM for EA は EA 開発において発生する作業の依存関係を明らかにする．これは Nord らが今後の課題として提示した依存関係の分析方法[28]として利用することが可能であり, EA 開発における課題であるチーム間連携の課題[42]への対処へつながる．さらに，サービス指向アーキテクチャ（Service-Oriented Architectures: SOA）[18]を用い，開発組織と SA のコンポーネントの対応関係を考慮するアイデアを示した．EA 開発を実践する時，チーム間の連携，すなわち依存関係のある作業の発生は，開発効率や開発スケジュール，プロジェクトの成否に影響を与える課題である[31][42].

本章では，開発組織と SA のコンポーネントの対応関係をアーキテクチャとして SOA を用いることで整理し，EA 開発に適するソフトウェア開発法を示す．事例への適用を通じてその妥当性を評価する．提案方法を用いることで，EA 開発を実践する時に発生するチーム間の連携に関する課題[42], 特に作業間の依存関係の発生が軽減され, EA 開発を円滑に行うことへつながる．

## 4.2 研究課題

本章の研究課題をまとめると以下の 2 点となる.

(1) EA 開発に適したソフトウェア設計プロセスはどうあるべきか

組織とアーキテクチャの関係[28], さらに人員や知識, 作業間の依存関係[9]がエンタープライズアジャイル開発を行う時に課題となる. 本論文では特に作業間の依存関係へ着目する. その理由は, 作業間の依存関係の存在が組織とアーキテクチャに最も影響を与えるという考えによる. 本章では, 組織とアーキテクチャの関係, 作業間の依存関係へ着目しエンタープライズアジャイル開発に適するソフトウェア設計プロセスに関して検討する.

(2) EA 開発に適した SA はどうあるべきか

エンタープライズアジャイル開発における組織とアーキテクチャに関する議論はない. 組織がアーキテクチャに影響を与える[28]ことを考慮し, エンタープライズアジャイル開発に適する汎用的なアーキテクチャが必要である. 本章ではエンタープライズアジャイル開発に適するソフトウェアアーキテクチャを検討することを研究課題とする.

## 4.3 関連研究

### 4.3.1 開発組織を構成するチームとその分類

アジャイル開発ではフィーチャーチームと呼ばれる最大 9 名で構成されるクロスファンクショナルなチームが推奨される．フィーチャーチームはソフトウェアの設計，実装，テスト，リリースまで全ての開発フェーズをチームとして担当する．複数の領域を担当可能なメンバと特定の領域に特化するメンバで構成されるので，チームとして変化に対応することができアジャイル開発を実現する．さらに，そのチームが地理的に同じ場所で作業に携わるとき，アジャイル開発が最も成果を上げることが知られている[28]．EA 開発ではより大規模で複雑なシステムを複数のチームで担当する．図 49 に Skelton らが提案するチームの分類を示す[34]．Skelton らはソフトウェアの開発からリリースまでを一つの流れとして捉え，アジャイル開発におけるフィーチャーチームをストリーム構成チーム（Stream Aligned Teams）とした．他にストリーム構成チームの自立を支援する実現化チーム（Enabling Teams），ストリーム構成チームを支えるチームとして特定ドメインに特化する複合サブシステムチーム（Complicated Subsystem Teams），インフラを担当するチームをプラットフォームチーム（Platform Teams）を定義した．Skelton らはチーム間の連携方法についても言及した．チームが開発を担当する機能をリリースするとき，リリースまでの作業が他のチームの作業待ち，すなわち他チームの作業に依存することがある．この課題はエンタープライズアジャイル開発においてチーム間の連携の問題（Inter-Team Coordination）の一つとして注目を集めており[42]，Bick らは作業間の依存関係の存在に気づくよう作業を進めることが重要であることをデータと共に示した[31]．Bick らも主張する通り，依存関係の存在により発生する作業はチームの生産性に影響を与えるので，Skelton らはチーム間のコミュニケーション方法について図 49 の通り API を通した連携と共同作業として定義した．EA 開発フレームワークはそのフレームワーク内で開発組織（仮想組織）を定義する．その仮想組織を構成するチームは Skelton らが提案する 4 つのチームとチーム間のコミュニケーション方法を用いることで分類することが可能である．本章では仮想組織を構成するチームが担当する領域とそのソフトウェアアーキテクチャコンポーネントの繋がりをこの方法を用い一般化する．

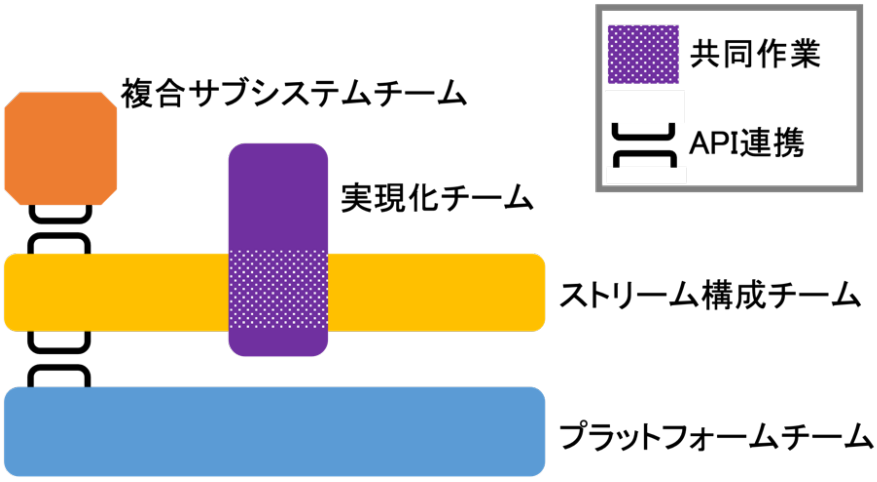


図 49 チーム分類

## 4.4 アプローチ

本章では EA 開発に適するソフトウェア開発法を提案する．開発組織を構成するチームと SA を構成するコンポーネントに着目し，SOA をアイデアとして用いることでトポロジの対応関係を整理する．開発組織と SA の関係を考慮することで EA 開発に適するソフトウェア開発法を示す．さらに，提案方法の事例検証を通じてその妥当性を評価する．

## 4.5 エンタープライズアジャイル開発向けソフトウェア開発法

SOA で提案される SA を利用し開発組織と SA の関係を整理する．図 52 に実際の組織，EA 開発フレームワークなどの開発プロセスフレームワークを導入し構築された仮想組織，SA の各ソフトウェアコンポーネントとの繋がりを示す．開発プロセスフレームワークを導入することで構築された仮想組織はSkelton らが提案するチーム分類の方法[34]を用いることで一般化した．提案する開発法では仮想組織とコンポーネントの関係を 3 つのパターンへまとめた．このパターンを用いて仮想組織とコンポーネントを整理したのちに，提案するソフトウェア設計プロセスを経ることで SOA を用いた EA に適するソフトウェア設計が可能となる．

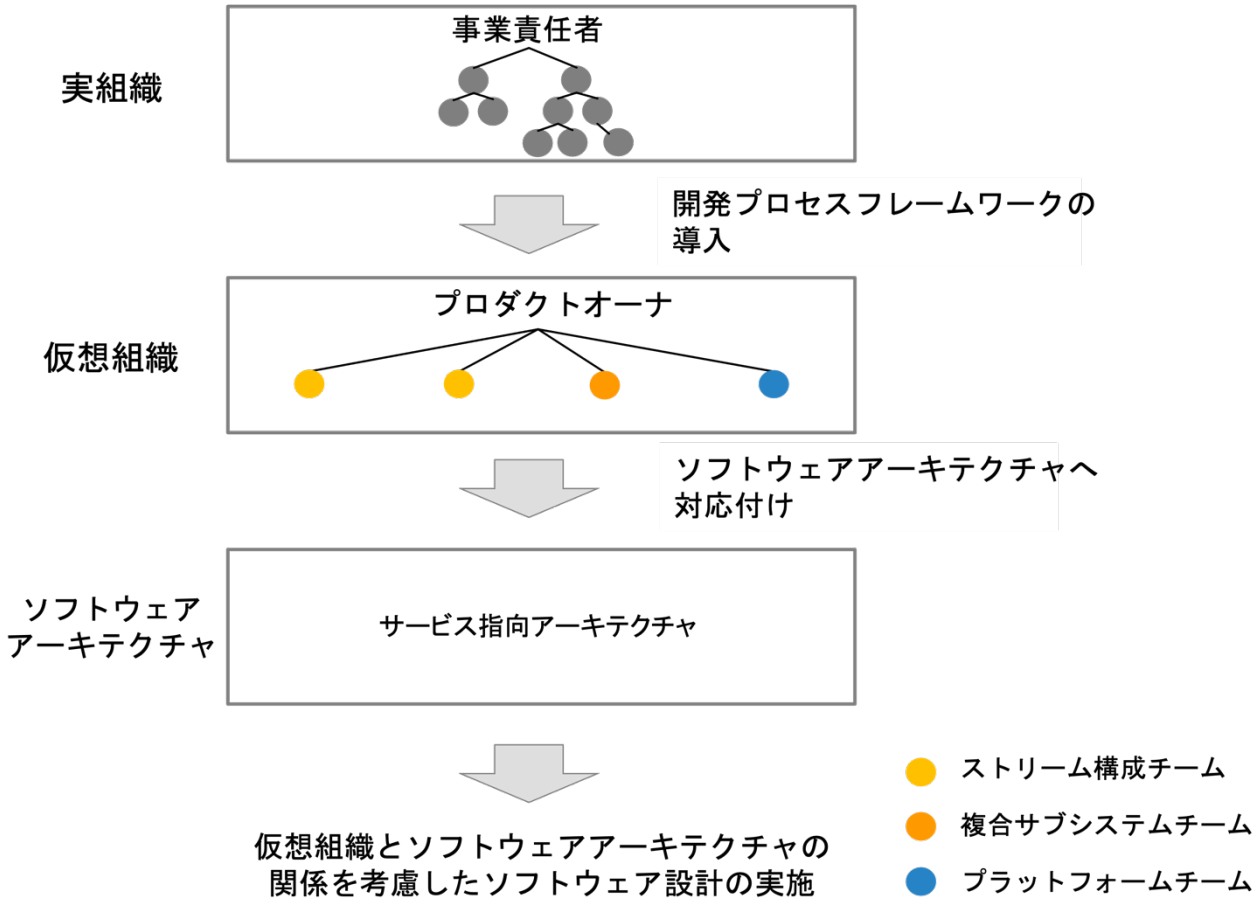


**図 50 EA 開発向けソフトウェア開発法の概要**

### 4.5.1 開発組織と SA の対応関係

1) パターン 1

図 51 に開発組織のチーム数が少ない場合に適する SA を示す．チーム数が少ないので，対応する SA も単純な構成であっても並行開発時に開発スケジュールの遅延は発生することは少ない．このとき，チームはソフトウェアの設計，実装，テスト，リリースなど全ての開発フェーズを担当する必要があるので，ストリーム構成チームである必要が

ある．

2) パターン 2

開発組織を構成するチームのトポロジとソフトウェアを構成するコンポーネントのトポロジの対応関係が 1 対 1（図 52）または 1 対多（図 53）となる場合のパターンを示す．提案方法は SOA が示す SA を利用しているので，ソフトウェアを 4 つのコンポーネントに分割して捉える．各コンポーネント間は疎結合であるので，チームは設計された通信プロトコルに従い他コンポーネントを担当するチームとシステムの詳細を議論する．図 52 においてはチームと SA のコンポーネントは 1 対 1 で紐づく．チームはストリームアラインチーム，複雑系システムチーム，プラットフォームチームで構成されることを想定する. 図 53 はストリームアラインチームがアプリケーションフロントエンドおよびサービスの SA コンポーネントの 2 つを担当する場合を示す.

3) パターン 3

図 54 に任意のコンポーネントを複数のチームで担当する際に有効な SA を示す．SOA では 4 つのソフトウェアコンポーネントが定義されているがコンポーネントによっては規模（ファンクションポイントなど）が大きいケースが発生する．開発組織としては規模の大きなコンポーネントまたは多くの開発計画を予定しているコンポーネントに対しては複数のチームを担当させることになる．このとき，開発組織を構成するチームのトポロジと SA を構成するコンポーネントのトポロジは合わない．この場合の対処方法としてコンポーネントを分割することが考えられる．すなわち，マルチチームパターンは開発組織を構成するチームのトポロジと SA を構成するコンポーネントのトポロジは合わない場合に適用するパターンである．ソフトウェアコンポーネントを分割するための方法として，ソフトウェア設計時にマイクロサービスアーキテクチャの技術を用いドメインの概念を導入することで新たなレイヤを追加することが考えられる．

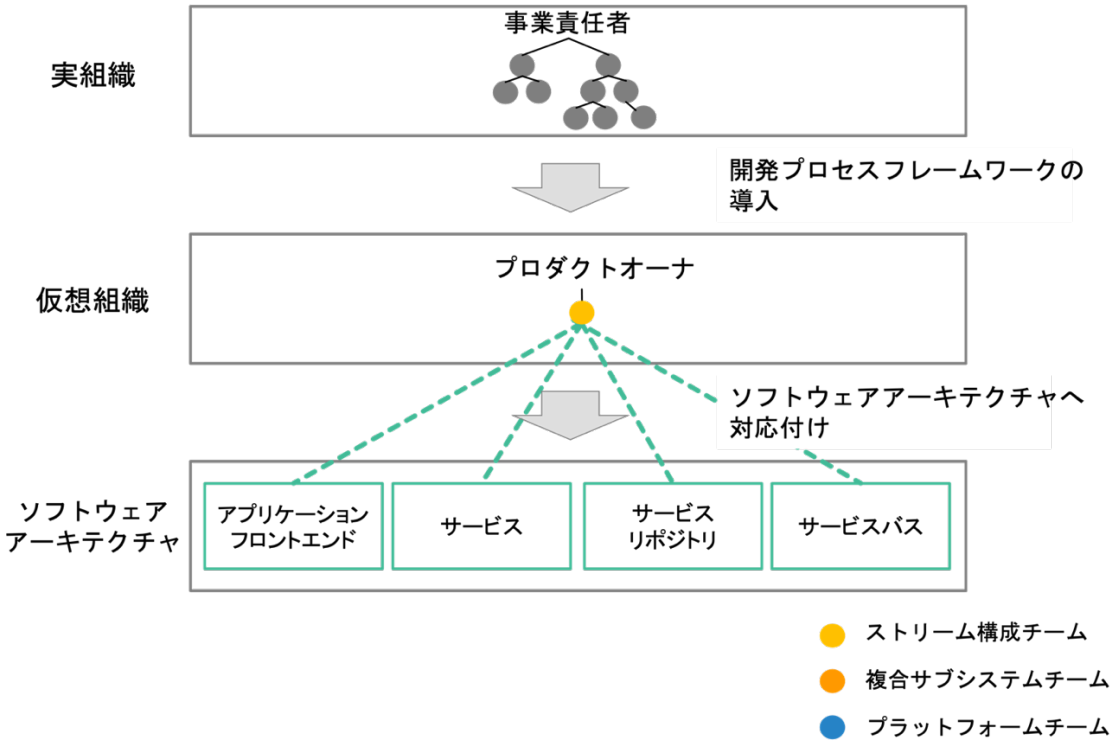


図 51 パターン 1

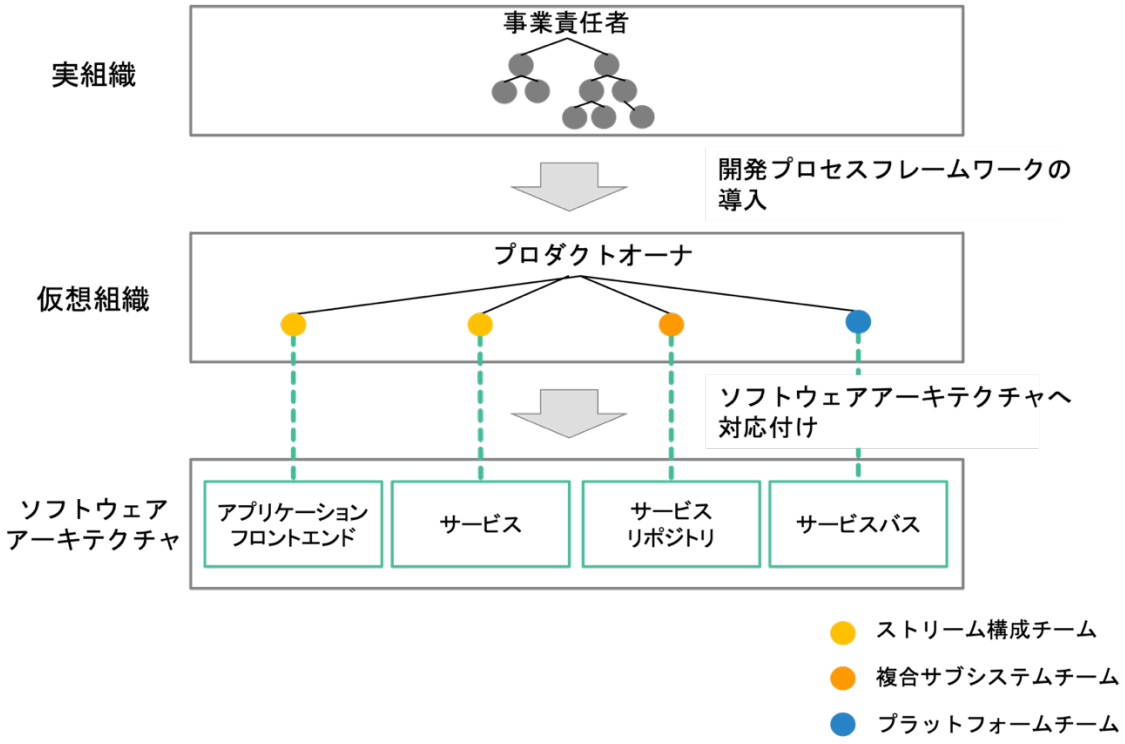


図 52 パターン 2-1

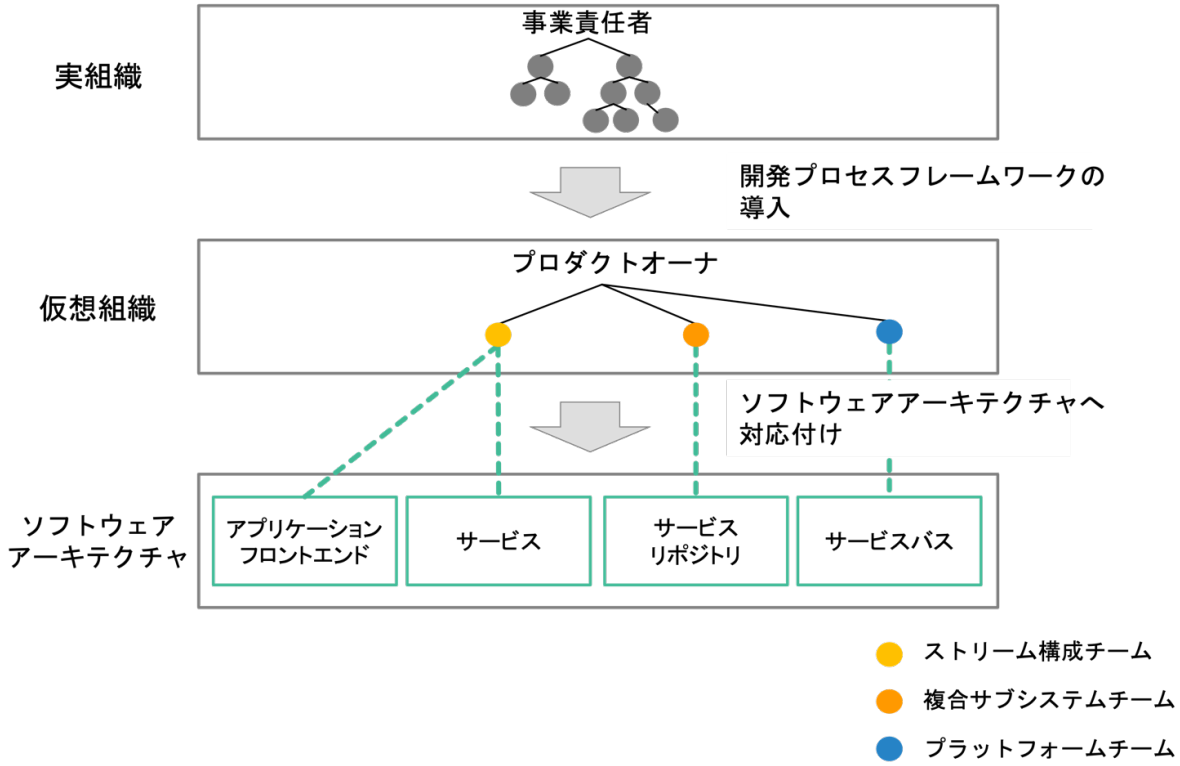


図 53 パターン 2-2

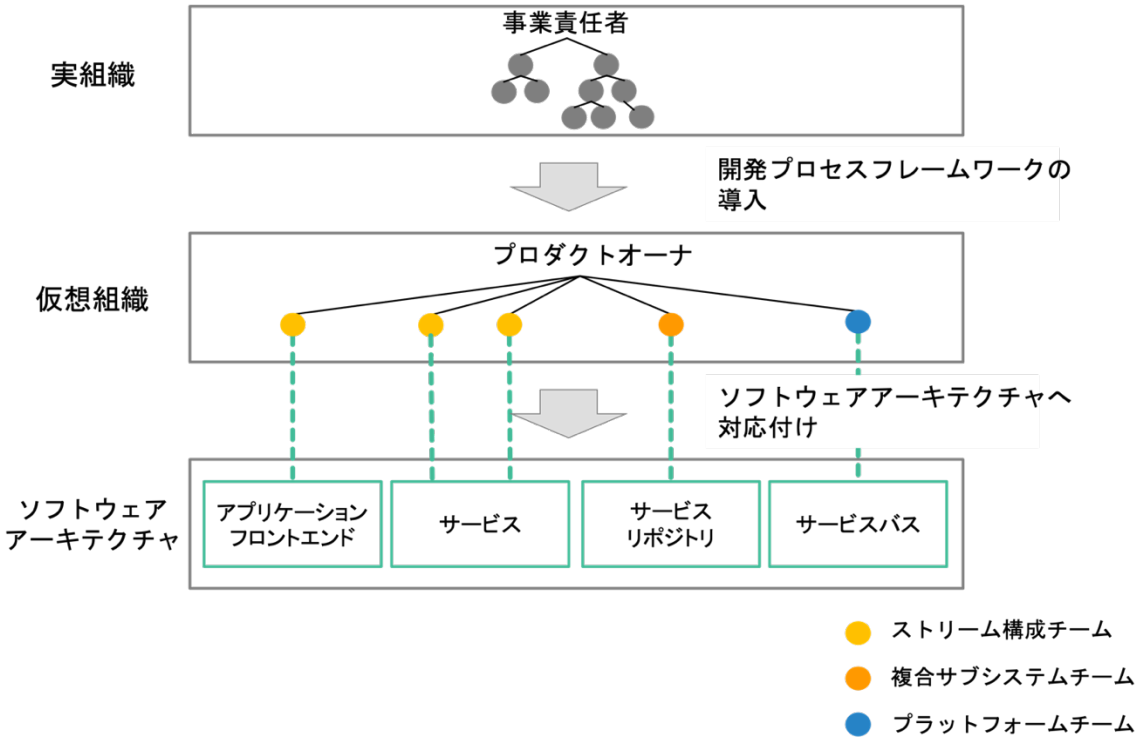


図 54 パターン 3

### 4.5.2 エンタープライズアジャイル開発向けソフトウェア設計プロセス

定義したパターンを用いたソフトウェア設計プロセスを図 57 に示す．提案方法は次の段階から構成される．

1. 開発プロセスフレームワークを用いた仮想組織の構築
2. 仮想組織からパターンを選択
3. パターンに応じたプロセスの実施
    a. パターン 3 の場合
        i. ドメインの抽出と設計
        ii. システム詳細設計
    b. パターン 3 以外の場合

段階 1 と 2 では開発プロセスフレームワークを用いて構築された仮想組織に適するパターンを選択する．段階 3 ではパターンごとにその後のプロセスを進め，特にパターン 3 ではドメイン分割を行うためのシステム詳細設計を行う．

このような手順とした理由は，開発組織の構造が SA に影響を与えるので，それに対処する方法としてソフトウェア設計フェーズにおける対応が有効であるという考えによる．パターン 1 と 2 では開発組織と SA のトポロジが適合しているので，ソフトウェア設計段階での対応は不要とした．パターン 3 では複数のチームが一部の SA コンポーネントを担当するので，並行開発時に同一コンポーネントの改修作業が発生し，その解消をするための作業，すなわちコンフリクト発生による解消作業が発生する．コンフリクトの発生を防ぐため，ドメインを分割するためのソフトウェア設計を行うことでこの課題へ対処することとした．

段階 1 では，EA 開発プロセスフレームワークを用い仮想組織を構築する．EA 開発向け開発プロセスフレームワークは様々あるが 4.3.1 で示した方法を用いることで，いずれのフレームワークを利用している場合も一般化することが可能である．段階 1 では一般化された仮想組織を構築するチームが定義される．

段階 2 では，仮想組織に合うパターンを選択する．仮想組織が扱うソフトウェアの SA に対し SOA を用いて整理し，仮想組織を構成するチームとの対応付けを行う．SOA の各コンポーネントと担当するチームの対応付けを明確にすることで，で開発組織と SA のトポロジの不適合が発生するか確認を行う．開発組織と SA のトポロジの不適合が発生する場合の対処は次の段階で実施する．

段階 3 では，決定したパターンに基づきその後のプロセスを行う．パターン 3 を選択した場合，ドメインの抽出と設計を行い，それらを実現するためのシステム詳細設計を実施する．マイクロサービス化の技術を用いたコンポーネントを分割するシステム設計は有効な方法の一つである．

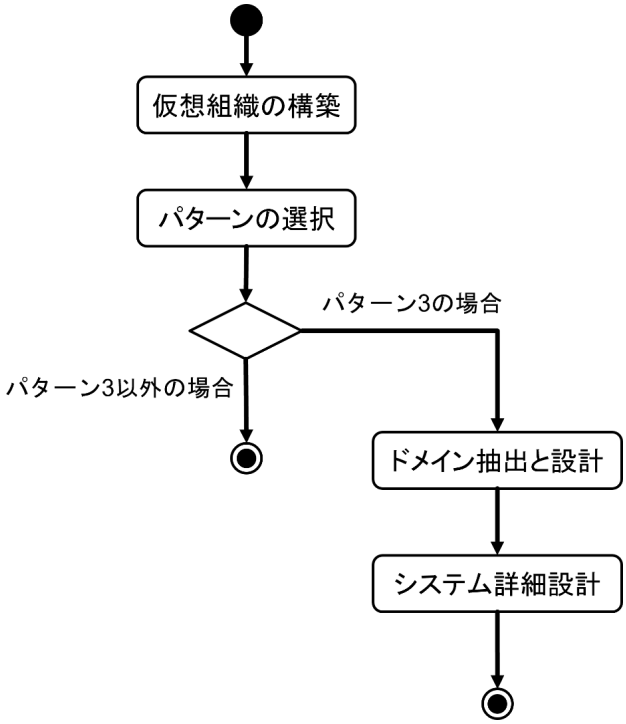


図 55 EA 開発向けソフトウェア設計プロセス

## 4.6 事例検証

観光情報を提供するサービスを提供する開発組織がエンタープライズアジャイル開発向けの開発プロセスフレームワークを導入することを事例として扱う．提案方法を用いて設計したソフトウェアに対し，SAAM for EA を用いた評価を行うことで提案方法の有用性を述べる．

### 4.6.1 観光情報を提供するサービスのシステム設計

図 58 に Large Scale Scrum，すなわちエンタープライズアジャイル開発向けの開発プロセスフレームワークを導入し構築した仮想組織を示す．図中では Large Scale Scrum を導入し構築された仮想組織を Skelton らが提案した方法[34]を用い一般化した．

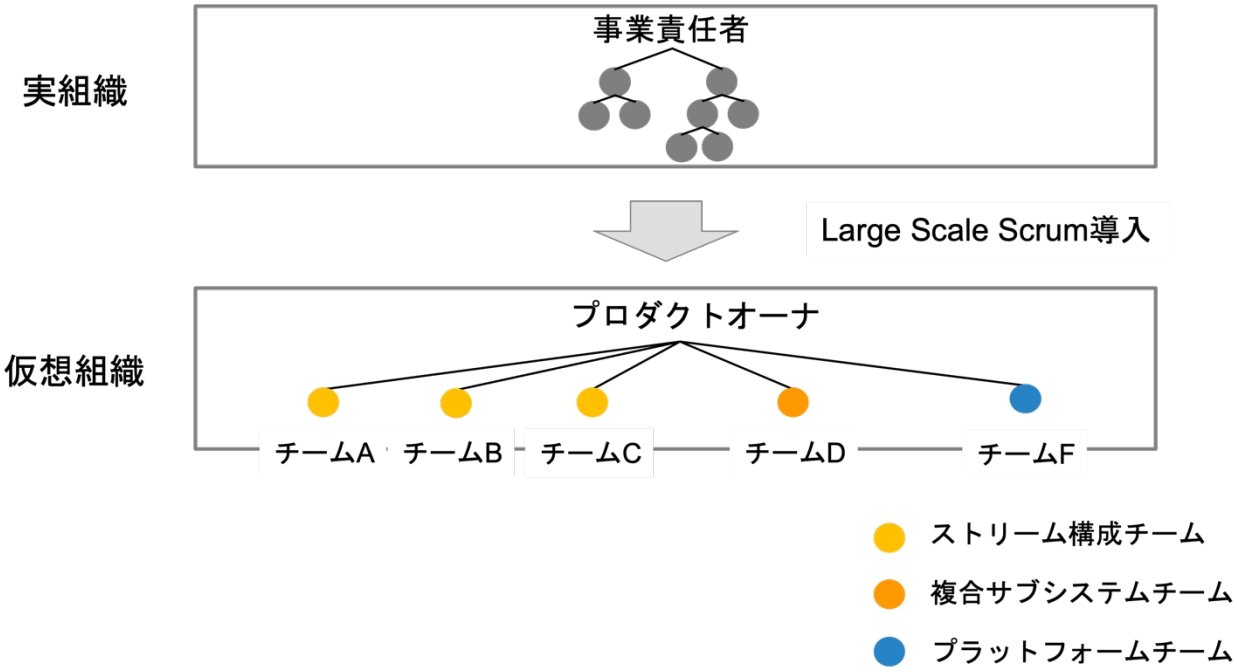


**図 56 LeSS を導入する組織**

この仮想組織が担当するソフトウェアは表 3 で示す機能を持ち，その SA を図 59 に示す．アプリケーションフロントエンドはスマートフォン向けアプリケーション，サービスはそのアプリケーションが利用する API を提供する．サービスリポジトリはサービス全体で共有する情報をまとめるリポジトリとなり，サービスバスはサービス間の非同期通信を実現するインフラを管理するコンポーネントを示す．

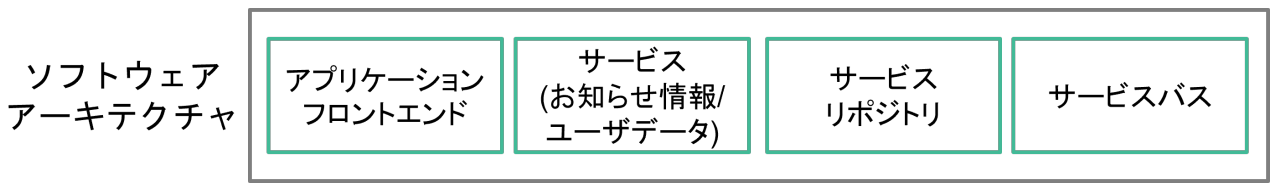


**図 57 仮想組織が担当するソフトウェアの SA**

仮想組織が対応する要求として表 4 で示すような機能開発を対応する必要がある．このとき，今後の開発計画ではアプリケーションフロントエンドにおける機能開発が多いことがわかるので，図 60 で示すように開発リソースを割り振る．アプリケーションフロントエンドを担当するチーム数が 2 つとなるので，提案方法におけるパターン 3 となる．表 4 で示す今後の開発予定の機能リストおよびアプリケーションが持つ各ドメインの複雑さを考慮し，提案方法で示したようにドメインの抽出と設計，すなわち本事例においては地図に関係するドメインを抽出しシステム詳細設計を行う．

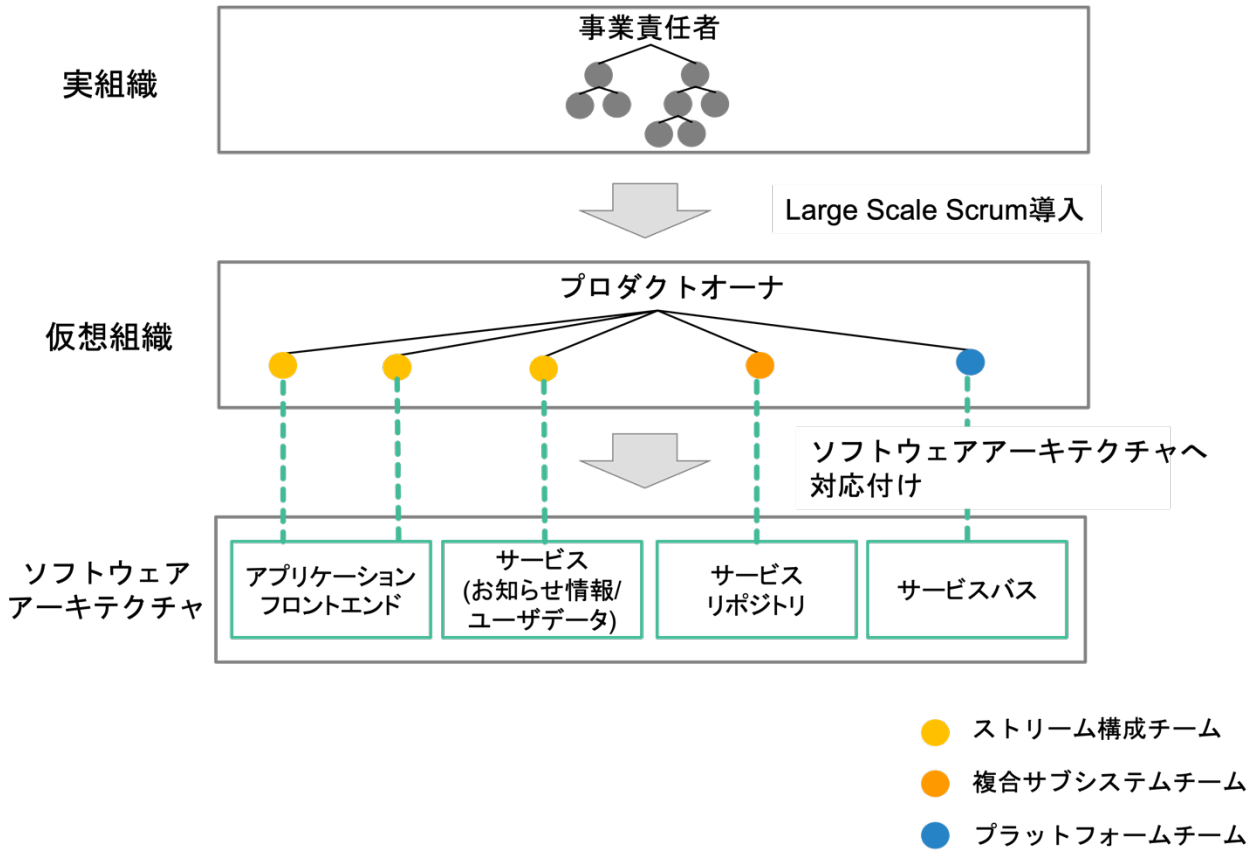


**図 58 仮想組織と SA の対応付け**

地図に関係するドメインを分割する．図 61 に分割する前の詳細設計，図 62 にドメインの概念を追加することで分割を行った詳細設計を示す．ドメイン分割を行うことで，ドメイン固有の処理を担当する Model コンポーネントを二つのグループに分けることができた．

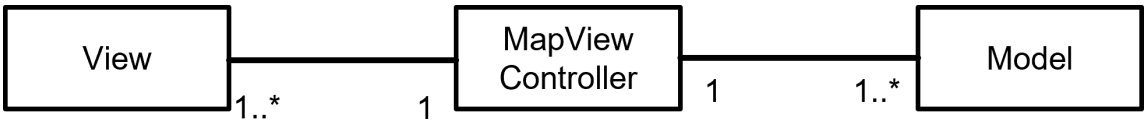


**図 59 詳細設計（ドメイン分割前）**

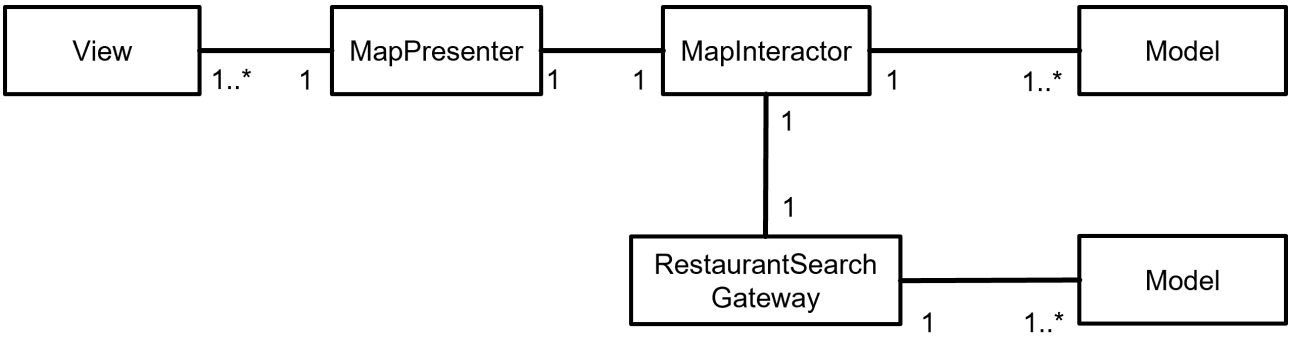


図 60 詳細設計（ドメイン分割後）

### 4.6.2 SAAM for EA を用いた設計したソフトウェアの評価

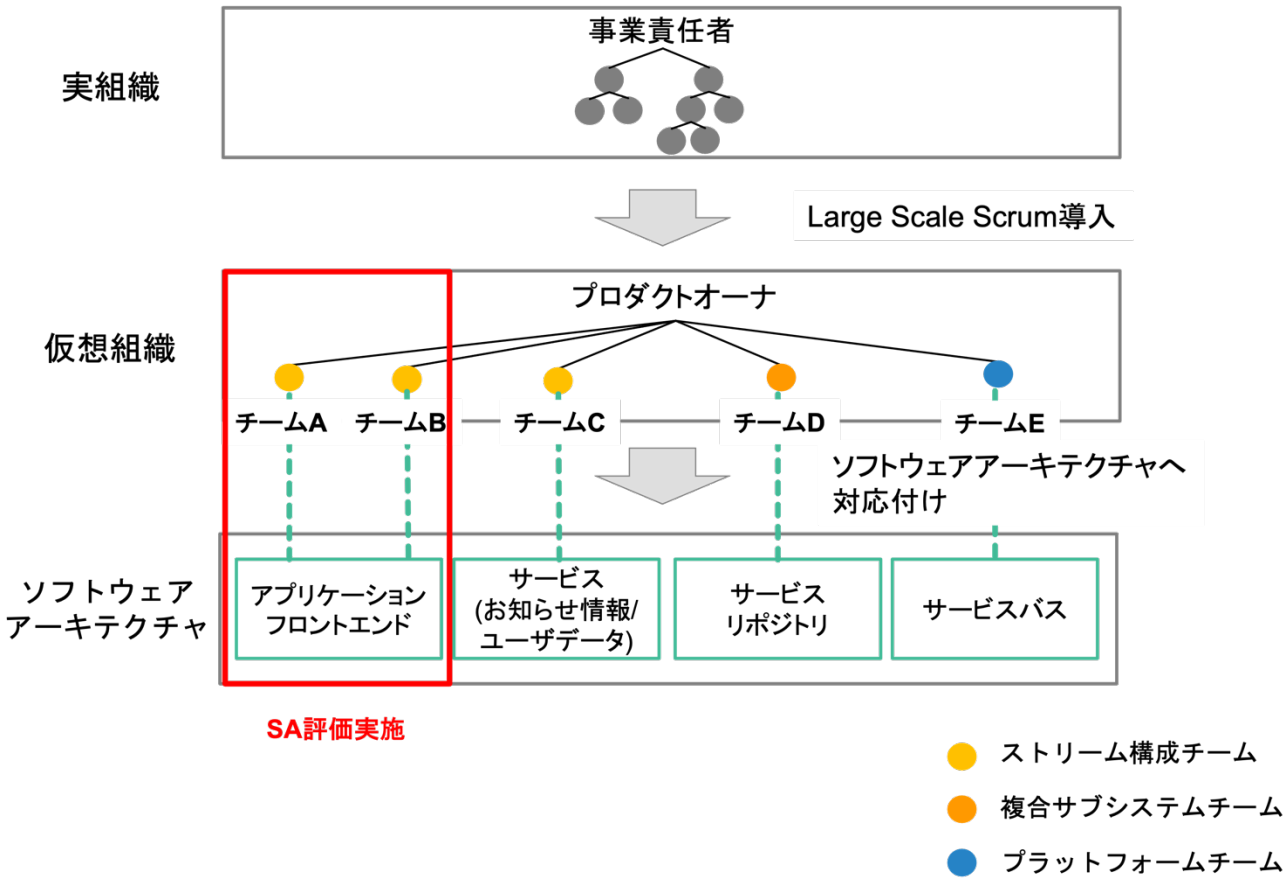


図 61 開発担当チームと SA

表 4 に示す今後の開発予定の機能リストを組み合わせ作成した評価シナリオを用い，設計したソフトウェアに対し SA 評価を実施する．比較対象としてドメイン分割をしなかった場合のソフトウェアを取り上げる．開発を担当するチームと SA の対応付けを図 63 で示し，評価シナリオを表 8 で示す．チーム A は地図に関係する機能の開発を担当するシナリオとすることで SA の差による影響を検証する．

表 8 評価シナリオ

| シナリオ No. | チーム A | チーム B |
|---|---|---|
| 1 | (2), (4) , (6)を開発 | (1), (3), (5)を開発 |
| 2 | (2), (4), (6)を開発 | (1), (3) , (5) , (7)を開発 |
| 3 | (2), (4), (8)を開発 | (1), (3), (7)を開発 |
| 4 | (4), (8)を開発 | (1), (3) , (5)を開発 |

ドメイン分割をしなかった場合の評価シナリオに対する評価結果を図 64 から図 67 に示す.

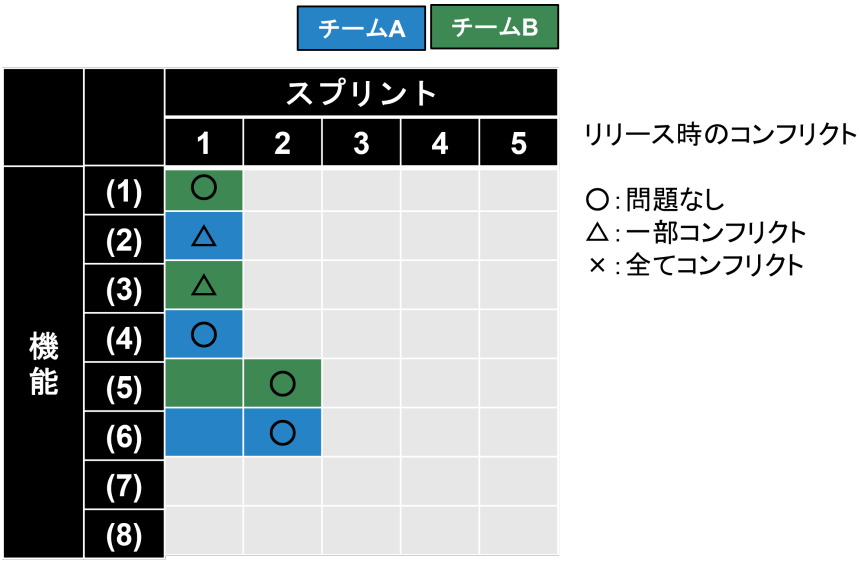


図 62 シナリオ 1（ドメイン分割なし）

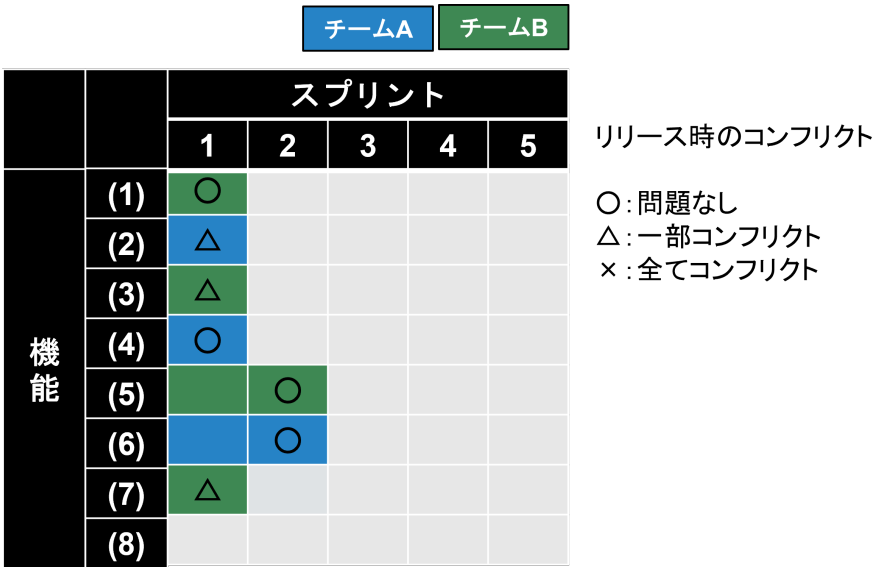


図 63 シナリオ 2（ドメイン分割なし）

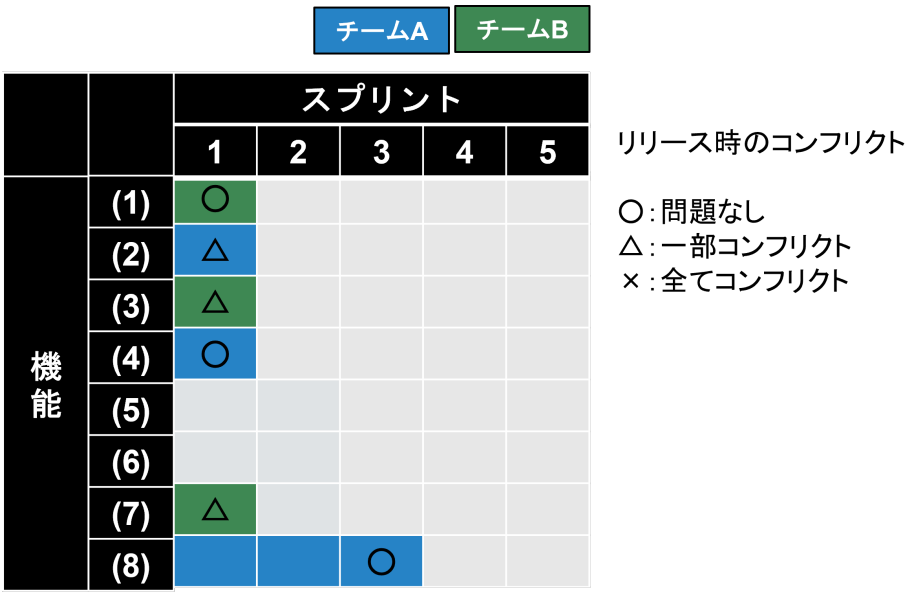


図 64 シナリオ3（ドメイン分割なし）

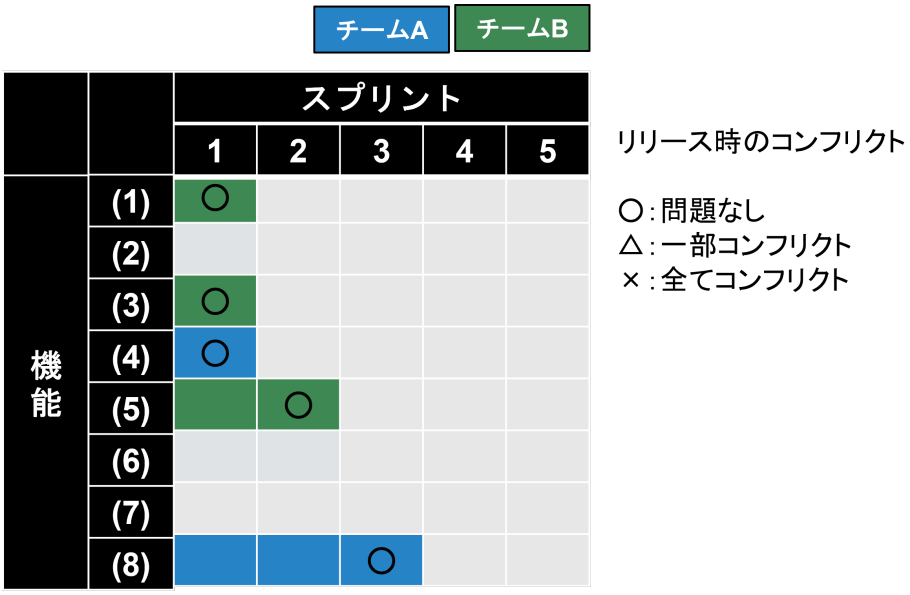


図 65 シナリオ4（ドメイン分割なし）

ドメイン分割をした場合の評価シナリオに対する評価結果を図 68 から図 71 に示す.

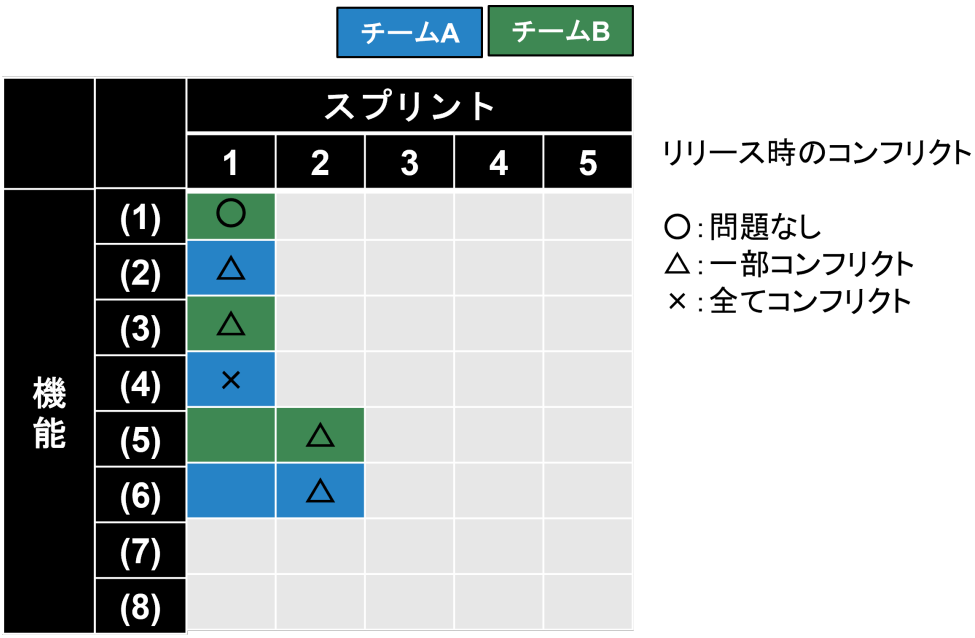


図 66 シナリオ1（ドメイン分割あり）

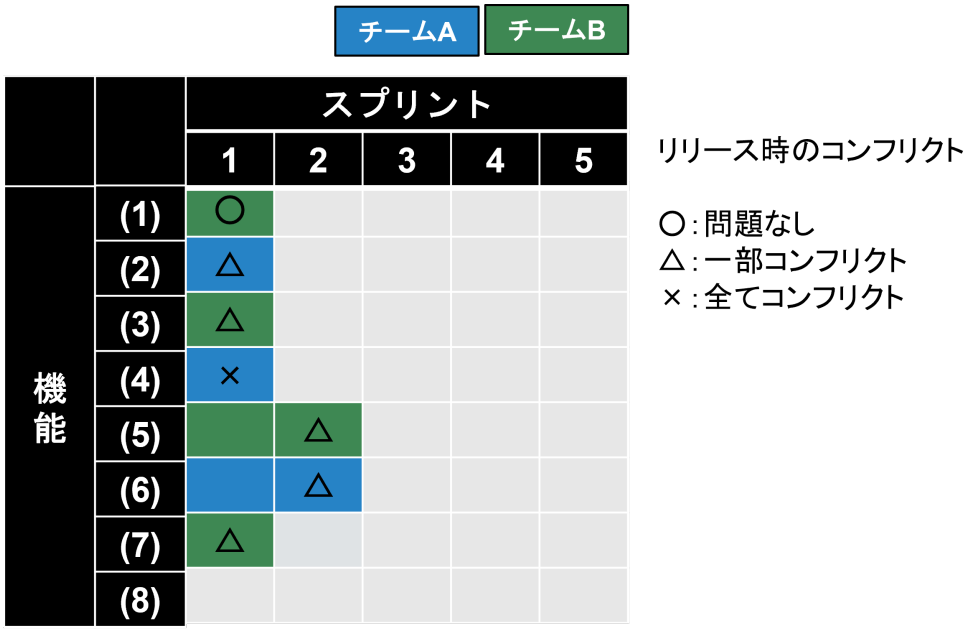


図 67 シナリオ2（ドメイン分割あり）

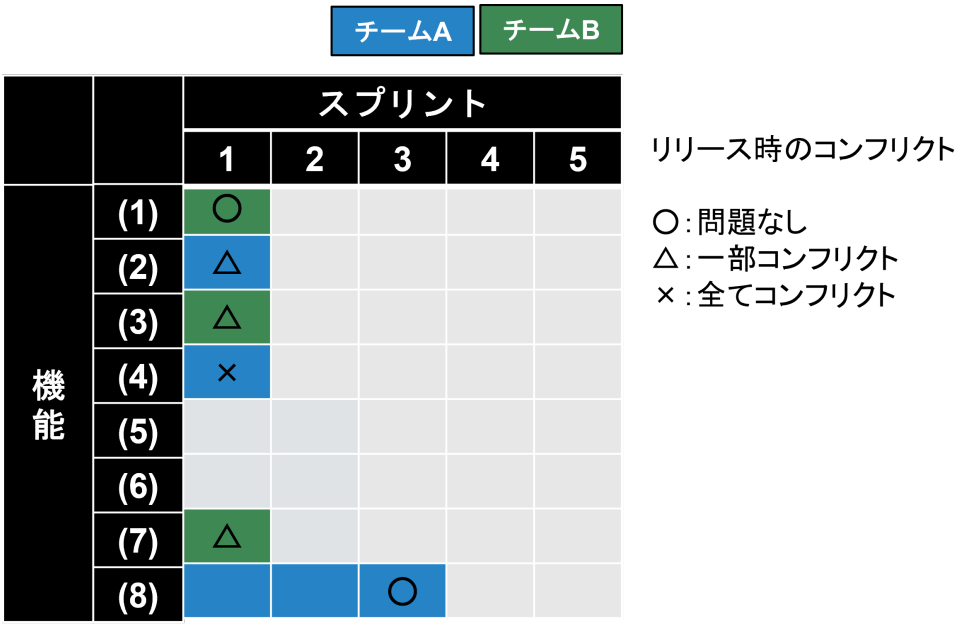


図 68 シナリオ3（ドメイン分割あり）

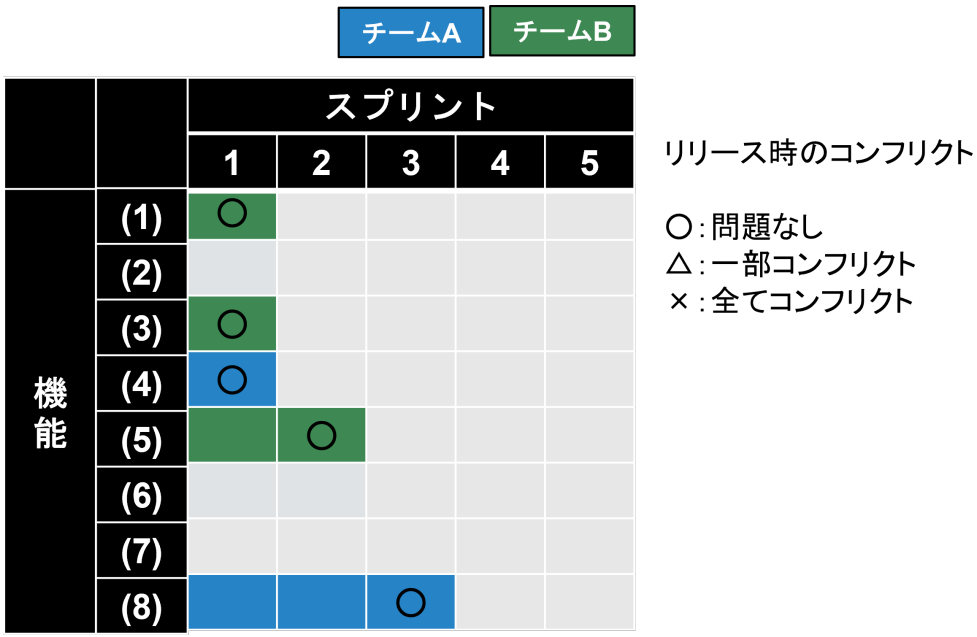


図 69 シナリオ4（ドメイン分割あり）

これらを踏まえ評価結果を表 9 にまとめる．表中の+は相対的に変更が評価シナリオへの対応であることを示し，-は相対的に評価シナリオへの対応が困難であることを示す．0 は差がないことを示す．ドメイン分割ありの場合がシナリオ 2，3，4 においてドメイン分割をしていない場合より良い結果となった．シナリオ 1 については差を確認することはできなかった．この結果か

ら，評価シナリオに対してはドメイン分割を行う方が相対的に良い結果となったと評価できる．

**表 9 評価結果まとめ**

| シナリオNo. | ドメイン分割なし | ドメイン分割あり |
|---|---|---|
| 1 | 0 | 0 |
| 2 | - | + |
| 3 | - | + |
| 4 | - | + |

## 4.7 考察

### 4.7.1 研究課題(1)に対する考察

EA 開発を行うとき，組織は様々な要求に対応する必要があり，大規模なソフトウェアの並行開発と市場の変化への柔軟な対応はその一つである[45]．現在課題となっているチーム間の連携の問題[42]，すなわち作業間の依存関係の発生はソフトウェアアーキテクチャおよびソフトウェア設計で対応をする必要がある．その理由は，作業間の依存関係の発生はソフトウェアアーキテクチャと開発組織の対応関係を原因とする場合に頻発するという考えによる．よって，提案方法では開発組織とソフトウェアアーキテクチャの対応関係を整理したのちに，その対応関係のパターンに応じてソフトウェア設計の実施を行うこととした．事例検証から，段階的に，すなわち開発組織とソフトウェアアーキテクチャの対応関係を整理したのちに，ソフトウェア設計を行うことで作業間の依存関係の発生が少ないことを確認した．よって，EA 開発に適するソフトウェア設計プロセスはその開発組織の構造と SA の対応付けを考慮した設計プロセスである必要がある．これは，SA は開発組織の構造の影響を受けるという理由による[14][28].

### 4.7.2 研究課題(2)に対する考察

提案方法では SA として SOA を採用しソフトウェア設計プロセスを提案した. SOA は Web システムに適する SA であり，例えば組込みソフトウェアを担当する開発組織が提案方法をそのまま利用することは難しい．提案方法と事例検証の結果から，EA 開発に適する SA に必要な特徴として SA の各コンポーネント間が疎結合であることが挙げられる．提案方法では，開発組織と SA の対応付けを行うときに SOA が定義する 4 つの SA コンポーネントと対応付けを行う．開発組織と SA のトポロジが適切，すなわちパターン 1 またはパターン 2 のときは並行開発時の同一コンポーネントの変更が発生することが少ないのでシステム詳細設計が不要とした．この設計プロセスとすることができた理由は，SOA では 4 つの SA コンポーネントが定義され，そのコンポーネント間が疎結合であることによる．よって，EA 開発に適する SA はその SA コンポーネント間が疎結合である必要がある．

### 4.7.3 EA 開発における作業間の依存関係発生とその対処

EA 開発では，複数チームで並行開発するので作業間の依存関係の発生が課題となる．これはチーム間の連携（Inter-Team Coordination）の課題としてエンタープライズアジャイル開発では注目を集める研究テーマである[42]．Nord らは EA 開発においてアーキテクト向けの開発組織と SA に関するガイドラインを整理し[28]，Uludağ らはさらにドメイン駆動設計[13]を用いレイヤードアーキテクチャを構築することを提案した[25]．他方，依存関係の発生する作業への対処法については課題であり，本章ではその対処法として提案方法を提案した．提案方法は有用と考えるが，開発作業が発生するのでそれが完了するまで作業間の依存関係が発生する．すなわち，EA 開発における作業間の依存関係の発生に関しては Bick らが示したように依存関係の存在に気づけるよう開発プロセスで対処しつつ[31]，提案方法を用い依存関係の発生を SA およびソフトウェア設計の改善を行うことで軽減することが重要である．これは SAFe のアーキテクチャ滑走路

[20][30]や Nord らが述べるように[28]，アーキテクチャを継続的に改善し続けることが EA 開発において重要であるということと合致し，その具体的な方法を示している．

## 4.8 検証上の問題点と残された課題

### 4.8.1 コンポーネントの規模とチーム数に関するさらなる事例検証

本章では評価シナリオを用いた定性的な検証を行なった．十分実用的なシナリオであるが定量的な事例検証は不足している．特にさまざまなSAコンポーネントの規模での事例検証は必要である．すなわち，SAコンポーネントの規模をファンクションポイント等で表現し，SAコンポーネントの規模を考慮したソフトウェア設計プロセスを検証する必要がある．例えば，提案方法では開発組織とSAの対応付けを整理した後に，開発組織とSAの関係をパターン3と分類した場合，ドメイン抽出を行い，システム詳細設計を行うこととした．コンポーネントの規模，すなわちファンクションポイントが大きく，担当するチーム数が少ない場合にはドメイン抽出とシステム詳細設計を行う必要がない可能性がある．すなわち，チーム数とファンクションポイントを図 57のプロセス内で変数として扱うことが考えられる．様々な規模のソフトウェア，チーム数での事例検証の実施は今後の課題とする．

### 4.8.2 汎用的なSAを用いたソフトウェア設計プロセスの構築

4.7.2で述べたように提案方法はSOAをSAとして採用しているので，Webシステム以外のソフトウェアへの適用が難しい．例えば，組込みソフトウェアなどを扱う開発組織においても提案方法を利用することができるよう，開発組織とSAを紐づける汎用的なSAを定義する必要がある．SOAのようにSAのコンポーネント間が疎結合であるSAが候補として考えられる．組込みソフトウェアをはじめ，Webシステム以外のソフトウェアを扱う開発組織でも利用可能なSAの検討は今後の課題とする．

### 4.8.3 ドメイン駆動設計法を用いたドメイン設計の実施とその事例検証

提案方法では特定の SA コンポーネントを担当するチームが複数存在する場合にドメインの抽出とそのソフトウェア設計をプロセスとして定義した（図 57）．本章で実施した事例検証では対象のソフトウェアの機能を考慮し地図に関する機能を分割することで対処を行なった．対象のソフトウェアが扱うドメインについて詳しい場合はこのような対処が可能であるが，そうでない場合はより具体的な方法が必要であり，その方法としてドメイン駆動設計法[13]が適切と考える．Uludağ らは Nord らが示したアーキテクト向けガイドライン[28]を参考にドメイン駆動設計を用いた事例検証を提示した[25]．このことから，提案方法で示したソフトウェア設計プロセス（図 57）においてもドメイン抽出とその設計を行う方法としてドメイン駆動設計法が適すると考えるがその事例検証は必要である．

## 4.9 まとめ

EA 開発において，チーム間の連携（Inter-Team Coordination）が課題となり注目を集めている[42]．作業間の依存関係はチーム間の連携の課題の一つであり，SA 観点[28]および開発プロセス観点[31]から研究が進められているが，作業間の依存関係が存在するときの対処法については課題である[28]．本章では作業間の依存関係が発生する原因として開発組織と SA の構造，すなわちトポロジの不適合が原因であると考え，開発組織と SA の関係を SOA の考えを用い 3 つのパターンへ集約し，その対応関係に応じたソフトウェア設計プロセスを提案した．これまで，作業間の依存関係に着目しソフトウェア開発法として具体的な提示はされていない．提案方法は開発組織と SA を考慮したソフトウェア開発法であり，作業間の依存関係の解消へつながる．

EA 開発を可能とする多くの開発プロセスフレームワークが提案され，その実践事例が報告されている[10]．EA 開発では複数のチームがアジャイル開発を行い，並行開発するので，チーム間の連携は注目を集める研究テーマである[31][42]．提案方法では，EA 開発におけるチーム間の連携という課題に対し，特に作業間の依存関係の存在に着目し SA およびソフトウェア設計観点から提案方法を設計した．これまで，開発組織と SA の関係についてアーキテクト向けのガイドラインは示され[28], SAFe ではEA 開発におけるアーキテクチャの重要性が言及される[20][30]．他方，アーキテクチャに起因する EA 開発の失敗と見られる報告もあり，EA 開発におけるチーム連携の課題は大きい．提案方法は組織と SA の関係をパターン化しつつ，それらを統合的に捉えたソフトウェア開発法とすることで課題へ対処した．

EA 開発においてチーム間の連携の課題，例えば作業間の依存関係の存在は開発の非効率さへ繋がるので注目を集めるテーマである[31]．開発組織とアーキテクチャの関係を考慮することはこの課題への対処の一つであり，SAFe ではアーキテクチャの重要性を言及し[20][30]，Nord らはアーキテクト向けに EA 開発のガイドラインをまとめた[28]．他方，チーム間の連携に関する課題は現在も報告され，アーキテクチャの不適合による EA 開発の失敗の報告がある[24]．提案方法は組織とアーキテクチャを統合的に捉え，組織とアーキテクチャのパターンに応じた依存関係への対処法を定義することで作業間の依存関係を解消する．提案方法を用いることで組織はその構造を大きく変更することなく，アジャイル開発が可能となる．

# 5 考察

本章では，1.2 で示した研究目的に対する考察を 2 章から 4 章の成果を踏まえまとめる．本論文では，複数チームでアジャイル開発を行う際に発生する課題，特に作業間の依存関係を原因として発生するチーム間の連携が開発効率に影響することに着目した．2 章では，事例検証を通して依存関係の存在がチーム間の連携に繋がり，開発効率に影響を与えることを確認した．3 章では，特に作業間の依存関係へ着目した評価方法を提案し，評価シナリオも用いた定性的な検証を実施することでエンタープライズアジャイル開発に適するソフトウェアアーキテクチャを評価する方法に関して議論をした．4 章では，組織とアーキテクチャを統合的に捉えることで依存関係へ対処する開発方法を提案し，評価シナリオを用いた定性的な検証を行うことで，組織の構造を大きく変更することなくアジャイル開発を行う方法を示した．これらを踏まえ，研究目的に対する考察を述べる．

## 5.1 エンタープライズアジャイル開発に適するソフトウェアアーキテクチャ

エンタープライズアジャイル開発に適したソフトウェアアーキテクチャは，特性としてコンポーネント間が疎結合である必要がある．本論文では，3.9.4 でサービス指向アーキテクチャを利用した組織とアーキテクチャの関係を整理するアイデアを示し，4.5 でそのアイデアをもとにエンタープライズアジャイル開発に適するソフトウェア開発法をまとめた．サービス指向アーキテクチャが示すアプリケーションフロントエンド，サービス，サービスリポジトリ，サービスバスの 4 つのコンポーネントは Web システムというドメインにおいて必要な責務と役割を持ち，コンポーネント間が疎結合である[18]．提案したソフトウェア開発法で組織とアーキテクチャの関係を整理することができた理由は，これらサービス指向アーキテクチャの持つ特性による．したがって，Web システムに限らないエンタープライズアジャイル開発に適したソフトウェアアーキテクチャを考えた時，必要な特性としてコンポーネント間が疎結合であることが挙げられる．さらに，マイクロアーキテクチャ観点では組織構造に従ってドメイン分割が必要である．4.5 で組織のパターンに応じた開発法を示し，複数チームが同一コンポーネントを担当するときにドメインを分割することを提案した．組織構造に従ってドメイン分割をすることで，組織構造を変更することなくアジャイル開発が可能である．すなわち，エンタープライズアジャイル開発に適したソフトウェアアーキテクチャとして，マイクロアーキテクチャ観点では組織構造に従ったドメイン分割が必要となる．

## 5.2 エンタープライズアジャイル開発に適するソフトウェア開発法

エンタープライズアジャイル開発に適するソフトウェア開発法は，組織とアーキテクチャの関係を踏まえる必要がある．Nord らは組織とアーキテクチャの関係を整理し，チーム間の連携を必要とする依存関係への対処方法を課題とした[28]．本論文では，チーム間の連携を必要とする作業間の依存関係[9]への対処へ着目し，組織とアーキテクチャの関係を整理することでソフトウェア開発法としてまとめた（4.5）．作業間の依存関係の発生は組織とアーキテクチャの関係性によって発生するので，4.5.1 では組織とアーキテクチャの関係性のパターンを整理し，

作業間に依存関係がある場合にシステム設計を行う手順を示した（図 57）．どの程度の粒度を持つコンポーネントが組織に合うかを判断することは難しい．提案方法では組織とアーキテクチャの関係性を考慮することで作業間の依存関係に着目し，対処が必要なコンポーネントの粒度を小さくする．したがって，エンタープライズアジャイル開発に適するソフトウェア開発法は，組織とアーキテクチャの関係を考慮することが挙げられる．

# 6 まとめと今後の展望

本論文では複数チームでアジャイル開発を行うエンタープライズアジャイル開発を実践する際の課題，特にチーム間の連携（Inter-Team Coordination）に関する課題に着目し取り組んだ．複数チームで 3 つのプロダクトを並行開発したいという組織の要求に合う開発プロセスフレームワークの設計と 1 年間の実践を 2 章で示した[39][40]．Dingsøyr らはエンタープライズアジャイル開発フレーワークを過信することなく，小さな規模でフレームワークを導入することから始めることの重要性を述べた[12]．提案方法は段階的にフレームワークの利用を拡大する際の新たな選択肢であるので，Dingsøyr らの考えと一致しており，組織がより大規模なアジャイル開発を行うことを支援する．さらに，2.7.4 で示したように 1 年間の事例検証[39][41]から作業間の依存関係の課題に着目し，その依存関係を明らかにする方法を 3 章でソフトウェアアーキテクチャ評価方法としてまとめ[40]，作業間の依存関係を解消するためのソフトウェア開発法を 4 章で示した．チーム間の連携の課題，特に作業間の依存関係に関する課題に対しソフトウェアアーキテクチャ観点の研究はなく，その分析方法と対処方法は課題であった[28]．本論文では課題であった作業間の依存関係の分析方法とその対処方法をソフトウェア開発法としてまとめ，その妥当性を評価シナリオを用いた定性的な検証で確認した．

本論文では，チーム間の連携，特に 2.7.4 で示した作業間の依存関係の存在により発生する課題に着目した．Nord らはエンタープライズアジャイル開発に携わるアーキテクト向けに組織とアーキテクチャの関係をガイドラインとしてまとめ，依存関係の分析と開発時におけるその対処法についての研究は今後の課題とした[28]．本論文では Nord らの提案をサービス指向アーキテクチャが示すアーキテクチャを用いることで組織とアーキテクチャの関係を整理し，作業間の依存関係の発生を防ぐソフトウェア開発法を提案した．提案方法はサービス指向アーキテクチャをアイデアとして用いることで，組織とアーキテクチャの関係を 3 つのパターンへ整理し，作業間の依存関係の発生に応じたソフトウェア設計段階での対処法を示すことができた．

デジタイゼーションへの注目の高さもあり，ビジネスを支えるソフトウェアの重要性とその巨大で複雑なソフトウェアを開発する方法論，すなわちエンタープライズアジャイル開発への注目も高まっている[12]．他方，エンタープライズアジャイル開発では開発の非効率化やスケジュールへの影響につながるチーム間の連携に関する課題への取り組みが続いている[12]．特に作業間の依存関係の発生に関しては課題であり，プロジェクトマネジメント観点での対処が提案されている[12][31]が，ソフトウェアアーキテクチャ観点での研究はない．Nord らはエンタープライズアジャイル開発に携わるアーキテクト向けにガイドラインをまとめたが依存関係への対処は今後の研究テーマとした[28]．そこで本論文ではチーム間の連携の課題につながる作業間の依存関係の発生に着目し，その課題に対しサービス指向アーキテクチャが提示するソフトウェアアーキテクチャを用い組織とアーキテクチャの関係を整理することでソフトウェア開発法としてまとめ，依存関係の発生を防ぐことを可能とした．組織とアーキテクチャの関係の整理をしたのちに，ソフトウェア設計段階でドメイン抽出を行うプロセスとすることで依存関係の発生を防ぎ，チーム間の連携の課題へ対処する．提案方法を用いることで，チーム間の連携の課題から

生じる開発の非効率さや開発スケジュールへの影響を軽減する．さらに，組織はその構造を大きく変更することなくアジャイル開発を可能とする．

今後の課題として，提案したソフトウェア開発法の事例検証とより汎用的なソフトウェアアーキテクチャを導入することがある．事例検証を行うことで，提案方法で示したソフトウェア設計プロセスでチーム数とソフトウェアの規模，すなわちファンクションポイントを変数として扱うことが見込める．例えば，開発組織とソフトウェアアーキテクチャの対応付けを考えるとドメイン分割が必要と判断されるとしても，チーム数が少なく，該当のソフトウェアアーキテクチャコンポーネントのファンクションポイントが大きい場合はドメイン分割が必要ないと判断できることが挙げられる．すなわち，提案方法で示したソフトウェア設計プロセス内でチーム数とファンクションポイントを変数として扱い，その値に応じた判定プロセスへ改善が見込める．次に提案方法への汎用的なソフトウェアアーキテクチャ導入について，現在はサービス指向アーキテクチャを用い開発組織とソフトウェアアーキテクチャを行っているので，Web システムを扱う開発組織で利用が可能となっているのでその点が課題である．この課題に対しては，サービス指向アーキテクチャが持つソフトウェアアーキテクチャのコンポーネント間を疎結合である特徴を持つソフトウェアアーキテクチャを定義し，特に組込みソフトウェアを扱う開発組織での事例検証を行うことでその妥当性を検証可能と考える．ソフトウェアアーキテクチャのコンポーネント間を疎結合である特徴を持つソフトウェアアーキテクチャを用いる理由は，ソフトウェアアーキテクチャのコンポーネント間が疎結合であることが開発組織とソフトウェアアーキテクチャの対応付けを可能とし，開発組織に属するチーム間のコミュニケーション方法とコミュニケーション頻度に影響を与える，という考えによる．

# 7 参考文献


[1] Agile Alliance, Scrum of Scrums, https://www.agilealliance.org/glossary/scrum-of-scrums（参照 2022-02-19）.

[2] S. W. Ambler, M. Lines, Disciplined Agile Delivery, IBM Press, 2012.

[3] Atlassian, Jira Software, https://ja.atlassian.com/software/jira（参照 2022-02-19）.

[4] J. M. Bass, A. Haxby, Tailoring Product Ownership in Large-Scale Agile Projects, IEEE Software, Vol. 36, No. 2, pp.58-63, 2019.

[5] K. Beck, Test Driven Development: By Example, Addison-Wesley Professional, 2002.

[6] M. Bregenzer , LeSS Adoption at a Bavarian Car Manufacturer, https://less.works/case-studies/bmw-group（参照 2022-02-19）.

[7] P. Clementsal, R. Kazman, and M. Klein, Evaluating Software Architectures, Addison Wesley, 2001.

[8] M. Cohn, Agile Estimation and Planning, Prentice Hall, 2005.

[9] D. E. Strode, S. L. Huff, A taxonomy of dependencies in agile software development, in 23rd Australian Conference on Information Systems, pp. 1-10, 2012.

[10] Digital.ai Software, 14th Annual State of Agile Report, 2020, https://stateofagile.com/#ufh-c-7027494-state-of-agile（参照 2022-02-19）.

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

[12] T. Dingsøyr, D. Falessi, and K. Power, Agile Development at Scale: The Next Frontier, IEEE Software, Vol. 36, No. 2, pp. 31-36, Mar./Apr. 2019.

[13] E. Eric, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2003.

[14] N. Ford, R. Parsons, and P. Kua, Building Evolutionary Architectures, O'Reilly Media, 2017.

[15] K. Hayashi, M. Aoyama, and K. Kobata, Agile Tames Product Line Variability: An Agile Development Method for Multiple Product Lines of Automotive Software Systems, Proc. of SPLC 2017, pp. 180-189, Sep. 2017.

[16] M. Kalenda, P. Hyna, and B. Rossi, Scaling Agile in Large Organizations: Practices, Challenges, and Success Factors, J. of Software: Evolution and Practice, Vol. 30, No. 10, pp. 1-24, Oct. 2018.

[17] G. Kim et al., The DevOps Handbook : How to create world-class agility, reliability, & security in technology organizations, IT Revolution Press, 2016.

[18] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall, 2004.

[19] C. Larman, B. Vodde, Large-Scale Scrum More with LeSS, Addison-Wesley, 2016.

[20] D. Leffingwell, SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises, Scaled Agile, Inc., 2018.

[21] R. C. Martin, Clean Architecture, Pearson, 2017.
[22] M. E. Moreira, Being Agile, Apress, 2013.
[23] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2015.
[24] Uludağ, Ö. et al., Evolution of the Agile Scaling Frameworks, International Conference on Agile Software Development, Springer, pp. 123-139, 2021.
[25] Uludağ. Ömer et al., Supporting large-scale agile development with domain-driven design, International Conference on Agile Software Development, Springer, pp. 232-247, 2018.
[26] M. Paasivaara, S. Durasiewicz, and C. Lassenius, Using Scrum in Distributed Agile Development: A Multiple Case Study, Proc. of GSD 2009, IEEE Computer Society, pp. 195-204, Aug. 2009.
[27] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley, 2012.
[28] R. L. Nord, I. Ozkaya, and P. Kruchten, Agile in distress: Architecture to the rescue, International Conference on Agile Software Development, Springer, pp. 43-57, 2014.
[29] D. Rosenberg et al., Parallel Agile, Springer, 2020.
[30] Scaled Agile Inc, What's New in SAFe 5.0, https://www.scaledagileframework.com/whats-new-in-safe-5-0/（参照 2022-02-19）.
[31] Bick, Saskia et al., Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings, IEEE Transactions on Software Engineering, pp. 932-950, 2017.
[32] K. Schwaber, J. Sutherland, The Scrum Guide, Nov. 2017, https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf （参照 2022-02-19）.
[33] Scrum Alliance, Scaling Works to Fit Your Needs, https://www.scrumalliance.org/get-certified/scaling （参照 2022-02-19）.
[34] M. Skelton, M. Pais, Team Topologies: Organizing Business and Technology Teams for Fast Flow, It Revolution Pr, 2019.
[35] J. Smed, K. Nybom, I. Porres, DevOps: A Definition and Perceived Adoption Impediments, Proc. XP 2015, Vol. 212, Springer, pp. 166-177, May. 2015.
[36] D. Šmite, N. B. Moe, and P. J. Ågerfalk, Agility Across Time and Space : Implementing agile methods in global software projects, Springer, 2010.
[37] J. Sutherland et al., Distributed Scrum: Agile Project Management with Outsourced Development Teams, Proc. of HICSS 2007, IEEE Computer Society, pp. 1-10, Jan. 2007.
[38] M. Tanaka, Kyoto Trip, https://github.com/masayuki5160/KyotoTrip（参照 2022-02-19）.
[39] M. Tanaka, M. Aoyama, A Distributed Large-Scale Agile Software Development for Multiple Products and Its Practical Evaluation, 2021 IEEE/ACIS 19th International Conference on Software Engineering Research, Management and Applications, IEEE, pp. 66-72, 2021.

[40] 田中 優之，青山 幹雄，エンタープライズアジャイル並行開発に適したソフトウェアアーキテクチャ評価方法の提案と評価，ソフトウェアエンジニアリングシンポジウム 2021 論文集，pp. 93-100，2021.
[41] 田中 優之，青山 幹雄，複数プロダクトのエンタープライズアジャイル開発方法の提案と実践，情報処理学会 デジタルプラクティス，Vol. 11，No. 3，pp. 569-588，Jul. 2020.
[42] T. Dingsøyr，N. Moe，Research challenges in large-scale agile software development，ACM SIGSOFT Software Engineering Notes，Vol. 38，pp. 38-39，2013.
[43] L. Williams et al. ，Scrum + Engineering Practices: Experiences of Three Microsoft Teams，Proc. of ESEM 2011，IEEE Computer Society，pp. 463-471，Sep. 2011.
[44] E. Woodward，S. Surdek，and M. Ganis，A Practical Guide to Distributed Scrum，IBM Press，2010.
[45] D. Zoran，B. Saša，Agile architecture in the digital era: Trends and practices，Strategic Management，Vol.24，No.2，pp. 12-33，2019.