Vol.2019-SE-202 No.3
2019/7/12

ICONIX手法を統合したフィーチャ指向ドメインエンジニアリングプロセス

浅野 雅樹[1,a)]　西浦 洋一[1,b)]　中西 恒夫[2,c)]　藤原 啓一[3,d)]

概要：本稿は，アイシン精機株式会社における自動車ボディ系製品向けプロダクトラインを開発するためのドメインエンジニアリングプロセス改定の事例を示す。このプロセスでは，抽象化や関心事の分離に長けた限られた数の開発者がフィーチャ分析を実施し，仕様からアーキテクチャ設計を含む他の作業は製品に精通する一般の開発者が実施する。フィーチャ分析は，可変性との整合性を担保しつつ抽象化と関心事分離を実現し，以降の工程で作成する開発文書をその分離構造に従わせるために実施する。要求と仕様は，ユースケース，ユースケースシナリオ，階層化された表形式による仕様記述 (USDM) によって詳細化される。さらに，USDM における仕様記述は，ロバストネス図のコントロール間の結合度を表す設計構造マトリクス（DSM： Design Structure Matrix）を用いた，定量的かつ説明可能な手法によってシステム分割へと詳細化される。今回改善したドメインエンジニアリングプロセスにより，仕様やアーキテクチャ設計に係るソフトウェアレビューの問題指摘件数が削減され，アーキテクチャ設計の期間が短縮され，かつ変更しやすいアーキテクチャが維持できたことが確認できた。

キーワード：ソフトウェアプロダクトライン, 自動車ボディ系製品, フィーチャ分析, ユースケース分析, ロバストネス分析, 設計構造マトリクス (DSM: Design Structure Matrix)

1. はじめに

アイシン精機株式会社は，1965 年に設立されたトヨタグループに属する自動車部品メーカである。ボディ系製品は社の中核製品である。社は世界各国の車両メーカに向けてボディ系製品を設計，製造，供給してきた。社は欧州ならびに開発途上国の市場でボディ系製品の販売を強化しており，それら製品の変種は増える一方であった。一方で，社のある中京地域におけるエンジニアの獲得競争は激しく，開発リソースは増やせないままである。そのためボディ系製品のプロダクトライン開発を企図した。

文献 [1] で報じたように，社はあるボディ系製品の開発をプロダクトライン開発に移行させることに成功した。文献 [1] で述べた我々の開発手法は，産業界において多くの成功事例 [2], [3], [4], [5] が報告されているフィーチャ指向のアプローチである。既存製品の複数の変種の制御仕様書を分析し，変種間の共通性と可変性をフィーチャモデルとして記述する [6]。このアプローチによって製品のソフトウェア構造の複雑度は顕著に下がった。それまで新しい変種を開発する際はクローン＆オウンによる開発を行っており，既存変種のソフトウェアに対する変更の影響分析に相当のコストを費やしていたが，プロダクトライン開発への移行によりそのコストが生じないようになった結果，大幅なコスト削減が実現できた。この成功事例により，社内ではプロダクトライン開発への関心が高まり，経営層はドメインエンジニアリングを担う開発支援部門にプロダクトライン開発を他の製品にも広げるよう求めだした。

開発支援部門は，他の製品に対しても同じアプローチを適用し，プロダクトライン開発への移行を進めており，品質，コスト，開発工期の改善を達成している。しかしながら，我々のアプローチはドメインエンジニアリングに 1〜2 年の時間を要している。ボディ系製品のライフサイクルは約 3〜5 年と長く，一度作成したソフトウェアは，次のライフサイクルが来るまで大きな変更を加えることができない。そのため，ドメインエンジニアリングの開発期間が長いことは，他製品をプロダクトライン開発へ移行させる

[1] アイシン精機（株）
2–1 Asahicho, Kariya, Aichi, 448–8650, Japan
[2] 福岡大学
8–19–1 Nanakuma, Jonan, Fukuoka 814–0180, Japan
[3] 三菱スペースソフトウェア（株）
5–4–36 Tsukaguchi Honmachi, Amagasaki, Hyogo 661–0001, Japan
[a)] asano_m@elec.aisin.co.jp
[b)] nishiura@elec.aisin.co.jp
[c)] tun@fukuoka-u.ac.jp
[d)] Fujiwara.Keiichi@mss.co.jp

ⓒ 2019 Information Processing Society of Japan

時期を逸し，会社全体のソフトウェア開発の生産性が向上しない状態が継続することを意味する。

今回アイシン精機は全社的なプロダクトライン開発の導入を進めるべく，従来のドメインエンジニアリングプロセスの改善に取り組んだ。全社的な導入を促進するうえで，これまでのドメインエンジニアリングであまりにかかっていた工期を縮めることが欠かせない。第 2 節では，社で確立されたドメインエンジニアリングプロセスがなぜ長期化しているのか，可能な理由を分析し，新しいドメインエンジニアリングプロセスで試みられたその解決策について言及する。解決策は，第 3 節で述べる，表による階層的な要求／仕様記述法，USDM を組み合わせたユースケースモデリング手法と，第 4 節で述べる設計構造マトリクス（DSM：Design Structure Matrix）を用いたロバストネス分析による定量的かつ説明可能なシステム分割である。第 5 節ではこれらの解決策によって改善されたドメインエンジニアリングプロセスを評価する。最後に第 6 節において本稿を総括する。

2. 課題の特定と解決策の立案

2.1 ドメインエンジニアリングの長期化要因

プロダクトライン開発を迅速に全社的に導入するには，各製品のドメインエンジニアリングに要する期間を短くする必要がある。社のこれまでのドメインエンジニアリングにおける経験から，ドメインエンジニアリングの期間が長期化する要因として以下が明らかになった。

- 開発者毎に異なるソフトウェアアーキテクチャの認識： 社内には様々な経験やスキルを有するソフトウェア開発者が在籍している。彼らはソフトウェアアーキテクチャに関して異なる認識を持っている。彼らがよいと考える，開発文書記述の抽象度やソフトウェア構造は一様ではない。ドメインエンジニアリングを加速するためにはより多くの開発者が必要だが，関与する開発者が増えれば，分析の手直しも多くなる。
- フィーチャ分析を担える開発者の不足： 製品ファミリを俯瞰し，その仕様の抽象度や関心事の分離を行い，変更に強いアーキテクチャを構築することがプロダクトライン開発を成功させるうえで必要不可欠である。しかし，すべての開発者がフィーチャ分析に習熟しているわけではない。社にいる多くの開発者は，要求仕様をどのような具体的手段で実現するか，その枝葉末節までを熟考するスキルには極めて長けているが，これまでクローン＆オウンによる派生開発を長く続けてきた経緯もあって，製品群を俯瞰したり，ドメイン技術を抽象化したり，それらをフィーチャとして表現する経験はさほど有していない。
- フィーチャ分析と構造化分析／設計の間の意味的ギャップ： 著者らが文献 [1] で提唱した手法は，図 1 に示すようにフィーチャ分析と構造化分析／設計を並行して実施する必要がある。フィーチャ分析と構造化分析の成果物，すなわちフィーチャモデルとデータフロー図（DFD）には一貫性が求められるが，両者の間には意味的なギャップが存在する。フィーチャモデルが抽象度の異なる概念を可変性制約とともに記述するものであるのに対して，DFD は機能要求を実現するためのデータ変換を表現するものである。この意味的ギャップゆえに，フィーチャ分析と構造化分析／設計を分業することが難しくなっており，数限られたフィーチャ分析ができる開発者に相当の負担を強いる結果となっている。両分析の並行的実施はフィーチャ分析を行える限られた開発者に相当の負担をかけている。
- ボディ系ソフトウェア開発に要求される品質の確保： 自動車業界では Automotive SPICE のようなプロセスに準拠した成果物間で一貫性がとれていること，設計が要求仕様をモレヌケなく実現していることが要求される。それらの成果物の一貫性は基本的に開発者のレビューによって担保されている。したがって，製品の規模や複雑さが増せば，レビューにより一層の工数が必要となる。また，著者らのプロセス [1] では，開発者個々人の経験とスキルに頼って構造化分析／設計を実施している。そのため，要求仕様が漏れなく DFD 上にデータフローとして実現されているか追跡する手順が統一されず，品質保証に係る工数が大きくなってしまう。

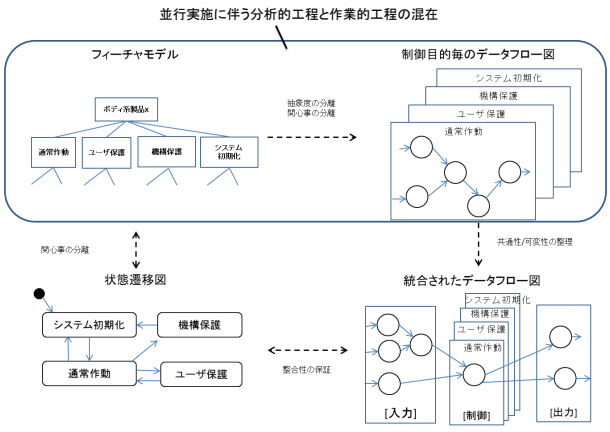

図 1 従来のドメインエンジニアリングプロセス

2.2 解決策の立案

以上のドメインエンジニアリングの長期化要因を踏まえたうえで，ドメインエンジニアリングの開発期間を縮め，より多くの製品系列の開発をより短い時間でプロダクトライン開発に移行できるよう，以下のようにドメインエンジニアリングプロセスの改善を図った。

まず著者らは，改善されたドメインエンジニアリングプ

ロセスにおいて，ユースケースアプローチによる要求／仕様サブプロセスを定義した。このサブプロセスではフィーチャ分析とアーキテクチャ設計が分離されている。フィーチャ分析は抽象化に長けた限られた開発者にしかできない。一方，フィーチャ分析の結果ひとたび抽象化と関心事の分離が実現されれば，他の工程は一般の開発者でも均質な品質で実施することが可能である。この分離は，抽象化に長けた開発者をフィーチャ分析とアーキテクチャの同時実行による過負荷から解放し，複数のプロダクトラインに対するドメインエンジニアリングのリソース制約による直列化を防ぎ，複数の製品群のプロダクトライン開発への移行を容易にする。

また，改善されたドメインエンジニアリングプロセスでは，要求はユースケースとしてモデリングされ，ユースケースシナリオとして詳細化される。さらにそれらの要求は，USDM[7] による階層的な表形式の記述を通して仕様に詳細化される。一連の詳細化プロセスはフィーチャモデルによって定められる抽象化と関心事分離の構造に律せられており，多くいる一般の開発者によって実施可能なものとなっている。USDM では要求とそれを充たす仕様を対にして記述するため，要求が漏れなく仕様として実現されていることを容易にレビューできるようになっている。

また著者らは，ICONIX プロセスのロバストネス分析 [8] による定量的かつ説明可能なアーキテクチャ構築のプロセスを確立した。ロバストネス分析はユースケースシナリオとして記述された要求をシステム分割に変換する，比較的，明示的なプロセスである。社の既存のドメインエンジニアリングプロセスでは，システム分割は開発者個人の勘と経験に頼って行われていた。新しいドメインエンジニアリングプロセスでは，システム分割はロバストネス分析によって，設計構造マトリックス（DSM： Desin Structure Matrix）を用いた定量的なやり方で行われる。USDM の仕様記述からシステム分割への変換を明示的なプロセスで行うため，構築されたアーキテクチャが機能的な仕様を漏れなく充たしているかを容易に検証できるようになる。また，フィーチャモデルで律せられ，DSM によって定量的になされるシステム分割は，開発のステークホルダ間での納得解を比較的容易に導出する。

改められたドメインエンジニアリングプロセスを図 2 に示す。

3. フィーチャ指向ユースケースモデリング

ユースケース分析の目的は，システムがアクタ，すなわちシステムのステークホルダとどのように相互作用するかを記述することによって，システムの機能要求を捉えることにある。ユースケース分析では，開発当初は抽象的で捉えどころのないシステムの機能要求を，システムとアクタの間でやりとりする情報や要求，応答の流れとして具体化できる。そのためシステムの機能要求に関するビジョンを開発者間で共有することが容易になる。一方で，ユースケースそのものをほどよい粒度と抽象度で，かつ製品間の可変性と整合のとれたかたちで定義することは熟練を要する。ユースケースシナリオについてもまた同様である。

そこで本稿で提案するユースケースモデリング手法では，最初にフィーチャモデリングを実施して，システムに関する諸概念や機能をフィーチャとして捉え，それらの間の抽象度の上下関係を整理し，また関心事の分離を図る。あわせて，それらフィーチャ間の部分的な意味的重複を，複数のフィーチャで共有される下位フィーチャとして見出す。

3.1 フィーチャ分析による抽象化と関心事分離

フィーチャ分析工程では，既存製品のシステム仕様書からシステムの要求や仕様，要素技術を代表するフィーチャを抽出し，それらを階層的に体系化して抽象化と関心事の分離を図る。フィーチャモデルには要求とそれを実現する仕様の両方のフィーチャが記述されるべきである。要求のフィーチャがそれを実現する仕様のフィーチャとともに定義されていないのなら補填されなければならないし，その逆も然りである。抽象化と関心事の分離に長けた開発者は多くはおらず，この工程はそうした開発者によって実行されるべきである。しかし，一般の開発者が担える他の工程までもを彼らに負担させるべきではない。本稿で提唱するドメインエンジニアリングプロセスでは，フィーチャ分析の結果に従属するかたちで他の工程を実施するようにしている。フィーチャモデリングを最初に実施する理由は，抽象化に長けた開発者にしかできない仕事と一般の開発者ができる仕事とを分け，人的資源の最適配置によって効率的な分業を図ることにある。

3.2 ユースケースによるシステム要求の記述

フィーチャ分析の結果，システムの機能要求，ならびにそれらを実現する仕様の，抽象度および関心事の分離構造，また各機能の仕様面での重複部分が明らかにされる。本工程では，その分離構造に従って，抽象度と粒度の揃ったかたちで，ユースケース，ユースケースシナリオ，さらにはそれらより詳細な仕様記述を行う。また，仕様の重複部分や製品による可変部分は，ユースケースの汎化，拡張，包含等の手段を用いた切り出しを行う。切り出しはフィーチャモデルの構造に従い，抽象度と粒度の揃ったかたちで行う。

本稿で提案するユースケースモデリング手法のプロセスは以下の通りである。

最初に，フィーチャモデルのサービスレベルの階層からユースケースモデルに表現されるサービスを抽出する。フィーチャモデルには，上層により抽象度の高い目的レベルのフィーチャ，下層により抽象度の低い手段レベルのフィーチャが配置される構成になっている。そこで，サー

ⓒ 2019 Information Processing Society of Japan

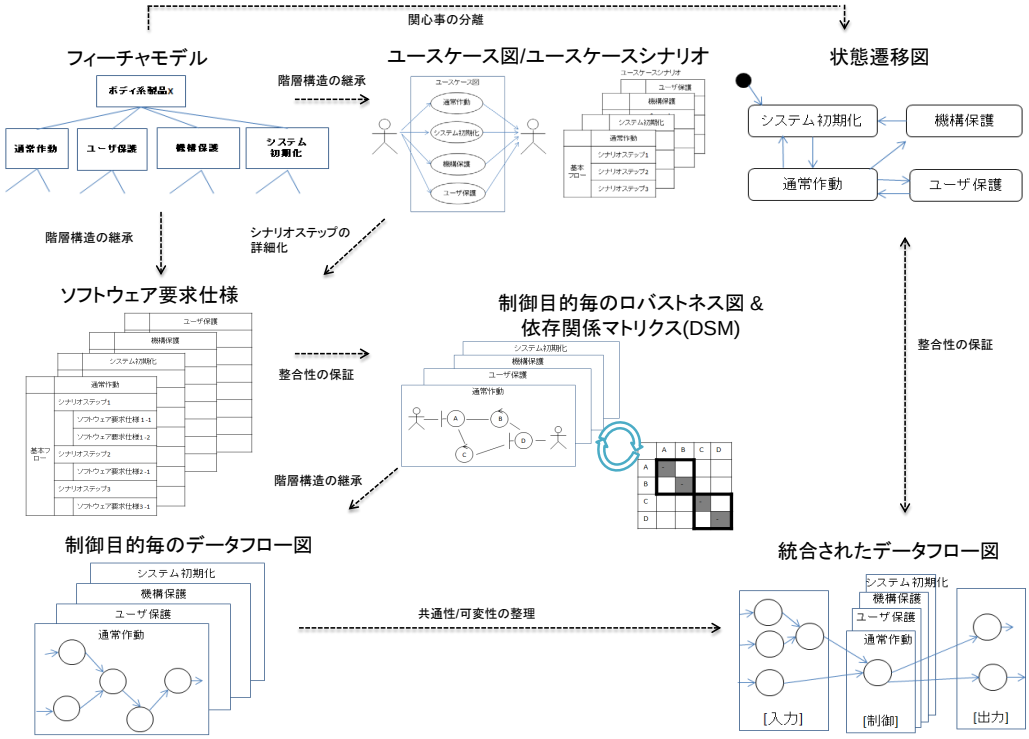

図 2 改善したドメインエンジニアリングプロセス

ビスレベルのフィーチャが並ぶ階層群を選択し，その階層のフィーチャに基づいてユースケースを定義する。サービスとは，アクタやシステム自身によって起動され，何かしらの判断と処理を行って，アクタに何かしらの価値ある応答を返す処理であって，かつシステムの関係者によってひとつの粗粒度の機能として認識されるものを言う。図 3 にフィーチャモデルのルート直下のフィーチャがユースケースに対応している例を示す。

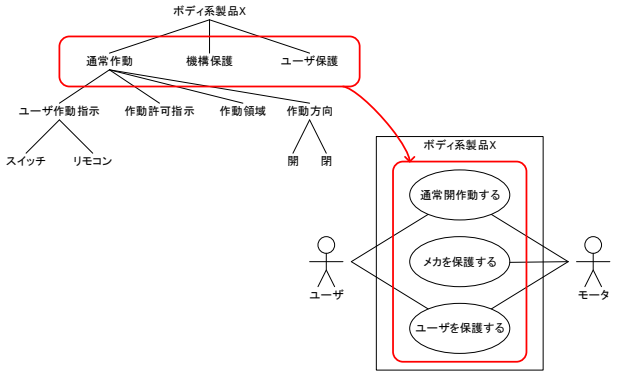

図 3 フィーチャモデルとユースケース図との対応

第二に，ユースケースモデルの個々のユースケースについて，そのシナリオを記述する。ユースケースのシナリオは，抽象レベルのものと具象レベルのものを並列に記述する。抽象レベルのシナリオにはシステムとアクタのやりとりの意図が，具象レベルのシナリオにはそのやりとりに関する具体的な操作が記述される。図 4 の例に示すように，ひとつのユースケースはサービスレベルのフィーチャ（例では第 2 層）に対応し，その抽象レベルのシナリオはその子フィーチャ（例では第 3 層）に対応し，さらにその具象レベルのシナリオはその子フィーチャの子フィーチャ（例では第 4 層）に対応する。第 5 層以下のフィーチャは，ユースケースシナリオの抽象度未満のものであり，ユースケースのシナリオ記述には対応しない。

一般に，ユースケースのシナリオには，ユーザインターフェースなど，システムとアクタのインタラクションに関する具体的な操作を含めるべきではないとされている。具体的な操作を含むユースケースシナリオは，しばしばユーザインターフェースに関するユーザの好みの変化などによって，要求そのものは変わっていない場合であっても変更することを強いられがちだ。抽象ユースケースシナリオは変更に強く，すなわち再利用可能である。しかし，多くの場合，制御工学や機械工学等出身のソフトウェア工学に明るくない開発者は，開発対象のシステムについてソフトウェア開発者と議論をするとき，具象ユースケースシナリオを好む。そのため，本稿で提案するプロセスでは，抽象ユースケースシナリオと具象ユースケースシナリオを併記し，それらをシステム要求を理解するためのすべての開発者の共通基盤としている。

3.3 ユースケースシナリオの詳細化によるソフトウェア要求仕様の記述

システムの要求をユースケース図，ならびにユースケースシナリオとして書いた後，ソフトウェア要求仕様を記述

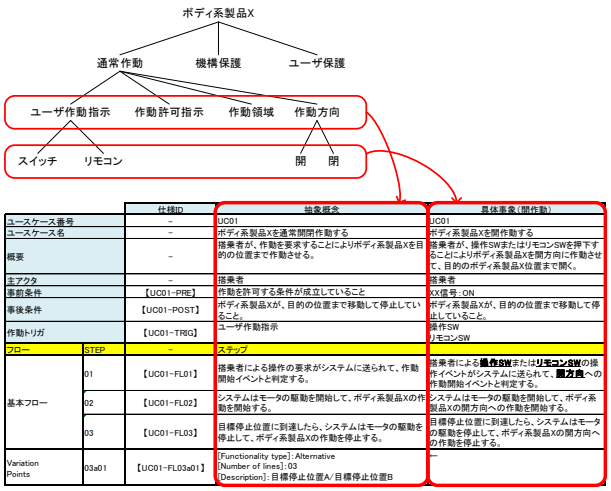

図 4　フィーチャモデルとユースケースシナリオとの対応

する。ソフトウェア要求仕様の記述には，USDM[7] に，プロダクトライン開発のための可変性表現の拡張を施したものを使用する。USDM では，要求は汎化／特化や全体／部分などの意味的関係に基づいて下位要求に分割される。下位要求への分割は階層的に行われ，最下層では要求は，曖昧さを持たない仕様にまで詳細化される。USDM では，要求からその下位要求へ，また要求から仕様への詳細化の関係が入れ子構造で表現される。また，各要求，仕様にはそれらが必要とされている理由が併記される。機械的機構の開閉動を行うボディ系システムの制御ソフトウェアの要求，仕様を USDM で記述した例（一部）を図 5 に示す。

図 5　USDM の例

USDM についても，ユースケース図やユースケースシナリオ同様に，要求や仕様の記述の抽象度を適切に選択できるか，要求を適切に下位要求に分割できるかは，開発者の能力に依存するところが大である。そのため，本稿で提案するプロセスでは，ユースケースシナリオに記述された要求を USDM によって仕様化する過程においても，フィーチャモデルに基づく抽象度分離を図るようにしている。その具体的なプロセスは以下の通りである。

最初に，ユースケースシナリオのステップを USDM 中に階層的に記述する。抽象ユースケースシナリオのステップを USDM の第一レベルの要求として記述する。USDM の第一レベルの要求の各々について，USDM の第二レベルの要求として，対応する具象ユースケースシナリオのステップを記述する。そのような具体ユースケースシナリオのステップが存在しない場合は，第二レベルの要求は記述しない。

第二に，フィーチャモデルの階層構造に基づいて，USDM にソフトウェア仕様を階層的に記述する。より具体的には，フィーチャモデル上のユースケースやユースケースシナリオに反映されていないフィーチャ（図 6 の例では第 4 層以降）に相当する仕様を，USDM の第三レベル以下に記述する。社はプロダクトライン開発への移行を進めているところであり，そのため USDM に記述される仕様は，既存のシステム仕様から改修のうえ移されている。

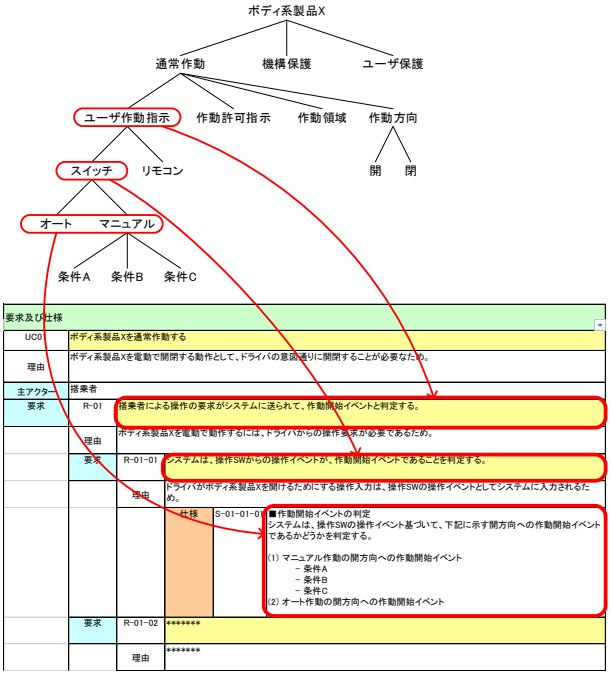

図 6　フィーチャモデルと USDM との対応

第三に，フィーチャモデルに記述されている共通性／可変性を USDM に反映する。これまで我々はソフトウェアの要求と仕様をフィーチャモデルとの整合性を保つかたちで USDM に記述してきており，各可変仕様はフィーチャ選択によって抜き差しできるようになっている。

4. ロバストネス分析を用いたアーキテクチャ分析

第 2 節で述べたように，アイシン精機では，フィーチャモデルとデータフローの意味的ギャップの大きさ，ならびに共通指針のないアーキテクチャ構築が，ドメインエンジニアリングの工期を延ばす理由となっていた。そこで社では，ユースケースとして記述された要求をシステム分割に

落とす手法として知られる ICONIX プロセスの導入を検討した。ロバストネス分析は強力な手法であるが，「原子的な」振舞い要求を意味するコントロールをコンポーネントに割り付ける方法は，やはり開発者の経験に頼っている。本稿の共著者が所属する三菱スペースソフトウェアでは，このロバストネス分析において，設計構造マトリクス（DSM： Design Structure Matrix）を用いて定量的に説明可能なコンポーネント割り付け法を実践してきた。アイシン精機は，三菱スペースソフトウェアのロバストネス分析をアーキテクチャ構築に採り入れることにした。

4.1 定量的かつ説明可能なシステム分割

三菱スペースソフトウェアで拡張されたロバストネス分析手法は，USDM で書かれたソフトウェア仕様からロバストネス図を作成する。ロバストネス図は次の 3 種類の要素で構成されている。

- バウンダリ： システムとその外側のインターフェースを意味する
- コントロール*:* システムの何かしらの動きを意味する
- エンティティ*:* システムで操作されるデータを意味する

USDM で記述されたソフトウェア仕様から以下の要領でロバストネス図を構築する。(1) ソフトウェア要求仕様書に記載した各シナリオステップの仕様をコントロールとして定義する。(2) ユースケース図の副アクタをバウンダリとして定義する。(3) ソフトウェア要求仕様書に記述されたデータをエンティティを定義する。(4) コントロール，バウンダリー，エンティティをソフトウェア要求仕様書の記述に基づき接続する。(1)〜(4) を全てのユースケースに対して実施し，一つのロバストネス図に集約する。要素間の依存関係を点数化し，点数の塊によってコンポーネントの境界を定義する。

まずロバストネス図中のコントロールの結合度を表す DSM を構築する。行列の行と列はロバストネス図中のコントロールと一対一の対応を持つ。すなわち，i 番目の行と i 番目の列はコントロール i に対応する。コントロール i とコントロール j の結合度を行列の (i, j) 成分にあてる。ここで，$c(c_a, c_b)$ と表される，ロバストネス図中の任意のふたつのコントロール c_a，c_b（$c_a \neq c_b$）の結合度を，次式のように定義する。

$$c(c_a, c_b) = \begin{cases} \sum_{p \in P(c_a, c_b)} w(p) & (\text{if } P(c_a, c_b) \neq \emptyset) \\ 0 & (\text{if } P(c_a, c_b) = \emptyset) \end{cases}$$

$P(c_a, c_b)$ は c_a と c_b の間の，0 個か 1 個のバウンダリかエンティティを含むパスの集合であり，$w(p)$ はその中の任意のパス $p \in P(c_a, c_b)$ の重みである。$w(p)$ は c_a と c_b が直接接続されている場合は 1，p の中にひとつのバウンダリかエンティティがある場合は 2 と定義される。

次に，できあがった DSM に対してある種のブロック対角化を施す。すなわち，コントロールの順番を並べ替えて，大きな数字の塊を行列の対角成分に寄せる。コントロールの並べ替えは，DSM の行と列の同順の並べ替えに相当することに注意されたい。このブロック対角化は決して数学的に厳密なものではない。少量の非零要素が非対角ブロックにあってもよいし，少量の零要素が対角ブロックにあってもよい。ブロック対角化は行列の色による視覚化の助けを借りて人手で行う。対角ブロックに相当するコントロールはお互いに強い結合を有しており，つまり，それらコントロールの機能は同じコンポーネントに割り付けられるべきものである。コンポーネントとはすなわち，構造化分析／設計で構築されるデータフロー図中のプロセスである。

4.2 エンティティ定義のルール化

三菱スペースソフトウェアの定量的 ICONIX プロセスは時に，多くのコントロールをひとつのコンポーネントに割り付けることがある。これはコンポーネントの役割が十分に分割されていないことを意味する。バウンダリはユースケース図のアクタ，コントロールは USDM によるソフトウェア仕様書の各シナリオステップに対応しているものの，エンティティの定義についてはそうしたルールがなく，開発者によって変わり得る。そのため，エンティティの定義次第では，エンティティ周囲のコントロール間の結合度が過度に大きくなり，結果として分割不十分なコンポーネントを生じることになる。

この問題を解決し，フィーチャ図の定める抽象化ならびに関心事の分離構造にあったコンポーネントを生成できるよう，アイシン精機では三菱スペースソフトウェアの定量的 ICONIX プロセスを用いるにあたり，エンティティの定義に関する追加のルールを設けた。そのルールはソフトウェア要求仕様書の最上位要求で出力するデータをエンティティとして定義するというものである。

図 7 は，アイシン精機のエンティティ定義に関する追加ルールとともに，三菱スペースソフトウェアの定量的 ICONIX プロセスを用いてブロック対角化をおこなった結果を示している。図 8 はブロック対角化の結果で定義される，コントロールの分割を示している。ロバストネス図に大きな分割がないことに注意されたい。つまりコントロールの関数はコンポーネントによりよく分散されている。

5. 評価

アイシン精機はあるボディ系製品 X の開発を，文献 [1] のドメインエンジニアリングプロセスに基づく，プロダクトライン開発に移行した。この初期開発では，フィーチャ分析は抽象化に長ける開発者によってなされた。ソフトウェアアーキテクチャ設計は，開発工期を縮めるべく，当初は製品に精通する他の一般の開発者によってなされた。

ⓒ 2019 Information Processing Society of Japan

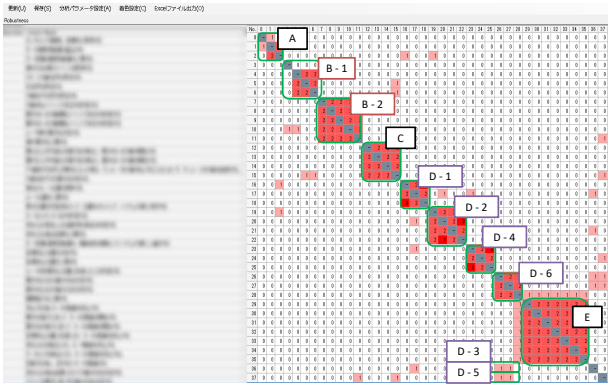

図 7 ロバストネス図から生成した最終的な DSM[*1]

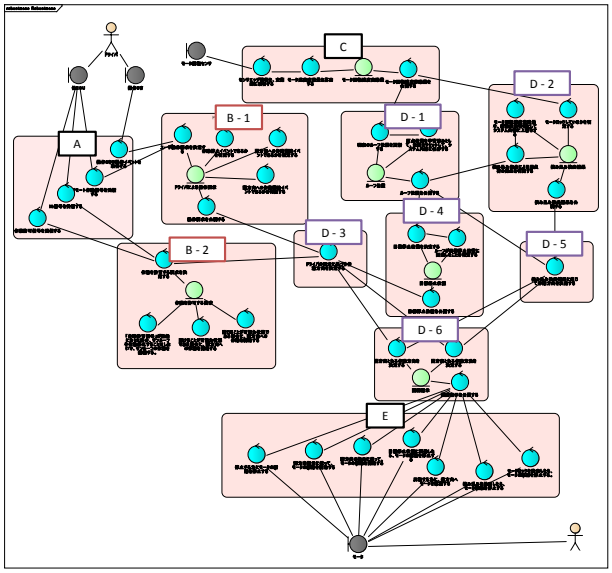

図 8 最終的なロバストネス図[*1]

しかしながら，少なからぬ手直しを生じたため，最終的にはフィーチャ分析に携わった開発者がアーキテクチャの設計を担う結果となった。

社はその後，初期開発では想定されていなかったフィーチャを追加するべく，プロダクトライン X の成長開発を行った。社は本稿で述べたようにドメインエンジニアリングプロセスを改良し，プロダクトライン X の成長開発に適用した。その成長開発では，フィーチャ分析は抽象化能力のある開発者によって，アーキテクチャ設計は製品をよく知る開発者によってなされた。フィーチャ分析とアーキテクチャ設計は，改良されたドメインエンジニアリングプロセスの通り，完全に最後まで分離したかたちで実施された。

5.1 フィーチャー分析に基づいたアーキテクチャ設計

著者らは，新しいドメインエンジニアリングプロセスにおいて，フィーチャ分析を行った開発者が期待した通りに，構造化分析／設計によって一般の開発者がアーキテクチャ設計をできたかどうかを検証するべく，アーキテクチャ設計のためのソフトウェアレビューにおける問題指摘件数を比較した。図 9 にその結果を示す。初期開発と成長開発の規模は異なるので，各々の開発のソフトウェアレビューにおける問題指摘数はその開発規模で割っている。すなわち，これらの数はソフトウェアの単位規模に対する指摘数である。また，これらの数は初期開発の総問題指摘数で正規化している。

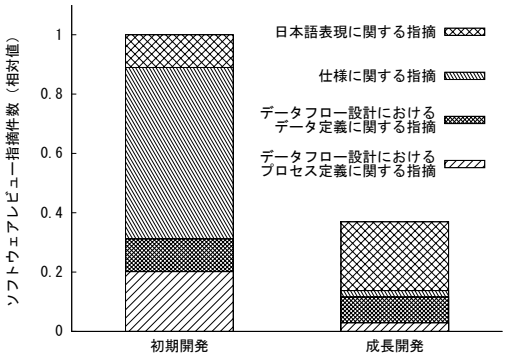

図 9 アーキテクチャ設計のためのソフトウェアレビュー結果

.

図が示すように，進化開発のソフトウェアレビューにおける指摘件数は，初期開発の 2/5 未満となった。初期開発のソフトウェアレビューでは，仕様に関する誤り，データフロー設計におけるプロセス定義に関する誤りが多かった。一方，成長開発のソフトウェアレビューではこれらの指摘は劇的に下がっている。多くの指摘は誤記や表現に関する軽微な誤りに因るものだった。しかしながら，データフロー設計におけるデータの定義に関する指摘は下がってはいるものの依然残っている。これらの結果から，変更されたドメインエンジニアリングプロセスによって，フィーチャモデルが規定するように，USDM で仕様を書いたり構造化分析／設計でシステムをプロセスに分割することができるようになったものの，データフロー設計については課題が残っているということが言えよう。

5.2 アーキテクチャ設計のリードタイム

アーキテクチャ設計のためのリードタイム，すなわちフィーチャ分析に始まり構造化分析／設計に終わる期間を計測した。初期開発と成長開発の規模が異なるので，各々の開発で計測されたリードタイムはそれぞれの開発の規模で割られている。つまり，開発されたソフトウェアの単位規模あたりのリードタイムが計算されている。成長開発で作られたソフトウェアの単位規模あたりのリードタイムは，初期開発で作られたソフトウェアの単位規模あたりのリードタイムのおおよそ半分となっている。

この削減はフィーチャ分析とアーキテクチャ設計の完全分離と，定量的かつ説明可能なシステム分割プロセスの定義によって実現された。ドメインエンジニアリングプロセ

[*1] 機密上，図の画質を意図的に落としている。

スにおけるこれらの改良は，ソフトウェアレビューに要する時間，ならびにアーキテクチャ設計における手戻りを削減するのに寄与した。

5.3 ソフトウェアアーキテクチャ

従来のドメインエンジニアリングプロセスでは，開発者の勘と経験に頼ったシステム分割が行われていた。新しいドメインエンジニアリングプロセスは，ロバストネス分析を用い，かつ DSM を用いた定量的，かつ説明可能なシステム分割が行われる。これらのプロセスによって構築されるソフトウェアアーキテクチャが，どれほど変更に耐えられるものになっているかを比較する。ソフトウェアの構造面における評価指標として，あるモジュールが変更されたときにどの程度の範囲にその影響が及ぶかを示す構造影響度を使用する。

文献 [9] で提案されている構造影響度が今回の比較に使用されている．構造影響度はモジュールの変更によってもたらされる変更範囲の度合いを示す．構造影響度 IR は以下によって定義される．

$$IR = \frac{1}{SLoC} \sum_{m \in S} \frac{\mathrm{LoC}(m)}{SLoC} \sum_{m' \in \mathrm{dep}(m)} \mathrm{LoC}(m')$$

S はモジュールの全体集合を，$\mathrm{dep}(m)$ は m 自身を含む $m \in S$ に依存しているモジュールの集合を，$\mathrm{LoC}(m)$ はモジュール $m \in S$ におけるソースコード行数を，そして $SLoC$ はソースコード全体の行数を示す。

構造依存度は単に依存しているモジュール数の平均では定義されず，より明確な定義を行うためにソースコード行数を基にしている。より大きいモジュールは変更される可能性が高いため，$\mathrm{LoC}(m)/SLoC$ を，変更の影響を受けるモジュールのソースコード行数の総和に乗ずる定義となっている。

進化開発におけるソフトウェアの構造影響度は従来開発と比較して 1.1 倍の 11%であった．今回改善したドメインエンジニアリングプロセスは，ソフトウェアの構造的な改善効果を維持したまま，ドメインエンジニアリングにおける開発期間を短縮できた。

6. まとめ

本稿では，アイシン精機における，車載系ボディ系製品のプロダクトライン開発のための，新しいドメインエンジニアリングプロセスを提案した。社の既存のドメインエンジニアリングプロセス [1] はフィーチャ分析と構造化分析／設計を並行に実施するものであり，それゆえドメインエンジニアリングを分業で行うことが難しく，抽象化と関心事分離のセンスを持った限られた開発者が，いっときにひとつの製品系列のドメインエンジニアリングに注力する必要があった。この状況はプロダクトラインの迅速な導入を妨げるものだった。

そのため，新しいドメインエンジニアリングプロセスでは，著者らは一部の開発者のみがうまくできるフィーチャ分析と，製品をよく知る一般の開発者にできる仕様からアーキテクチャ設計を含む他の仕事とを分離した。フィーチャ分析は抽象化の階層を定め，関心事分離を実現し，そして要求と仕様のユースケース，ユースケースシナリオ，USDM による段階的詳細化と構造化分析／設計によるアーキテクチャ設計とを律する。ロバストネス分析が仕様とシステム分割の意味的ギャップを埋めるために用いられる。ロバストネス図は USDM による要求から機械的に構築され，システムはその図上で定量的，かつ説明可能な方法で設計構造マトリクス（DSM: Design Structure Matrix）の助けを借りて分割される。この機械的なアプローチは，関連する開発者内でのアーキテクチャ設計に関する合意形成を容易にし，またアーキテクチャ設計に関する手戻りを減らす。仕様やアーキテクチャ設計に係るソフトウェアレビューの問題指摘件数は本プロセスで劇的に削減された。

参考文献

[1] Y. Nishiura, M. Asano, and T. Nakanishi, “Migration to Software Product Line Development of Automotive Body Parts by Architectural Refinement with Feature Analysis,” *Proc. 25th Asia-Pacific Software Engineering Conference (APSEC 2018)*, pp. 522–531, Dec. 2018.

[2] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, “FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architecture,” *Annals of Software Engineering*, Vol. 5, No. 1, pp. 143–168, 1998.

[3] K. C. Kang, S. Kim, J. Lee, and K. Lee, “Feature-Oriented Engineering of PBX Software for Adaptability and Reusability,”” *Software: Practice and Experience*, Vol. 29, No. 10, pp. 875–896, Oct. 1999.

[4] T. Iwasaki, M. Uchiba, J. Ohtsuka, K. Hachiya, T. Nakanishi, K. Hisazumi, and A. Fukuda, “An Experience Report of IntroducingProduct Line Engineering across the Board,” *Proc. 14th Int. Conf. onSoftware Productline Conf. (SPLC 2010)*, pp. 255-258, Sep. 2010.

[5] J. Otsuka, K. Kawarabata, T. Iwasaki, M. Uchiba, T. Nakanishi, K. Hisazumi, and A. Fukuda, “Small Inexpensive Core Asset Con-struction for Large Gainful Product Line Development: Developinga Communication System Firmware Product Line,” *Proc. 15th Int.Software Product Line Conf. (SPLC2011)*, 5 pages, Aug. 2011.

[6] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Analysis,” Technical Report, CMU/SEI-90-TR-21, SEI/CMU, Nov. 1990.

[7] 清水 吉男,「要求を仕様化する技術・表現する技術」技術評論社, 2010 年.

[8] D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007.

[9] 岡本 渉,「ソフトウェア設計の保守性を評価し改善する構造診断手法」東芝レビュー, Vol. 68, No. 9, pp. 42–45, 2013 年 9 月.

ⓒ 2019 Information Processing Society of Japan