情報処理学会論文誌 Vol.61 No.2 417–428 (Feb. 2020)

フィーチャ指向アプローチによる自動車ボディ系製品のプロダクトライン開発への移行事例

西浦 洋一[1,a)] 浅野 雅樹[1,b)] 中西 恒夫[2,c)]

受付日 2019年5月23日, 採録日 2019年11月7日

概要：本論文では，ある自動車ボディ系製品の制御ソフトウェアの変種群の開発をフィーチャ指向のプロダクトライン開発方法論による開発に移行した事例を通して，従来の事例報告では詳述されることのなかった，ドメインエンジニアリングプロセスが確立されるまでの過程を詳述する．移行の過程で実施したフィーチャ分析と構造化分析によるリバースエンジニアリングの結果，開発関係者のシステム理解が促進され，製品の特性を考慮したドメインエンジニアリングプロセスが確立されるとともに，対象製品のプロダクトラインアーキテクチャが実現された．最終的に，変種開発に要するコストはクローン&オウン開発を行う際の見積り値の 1/3 に削減し，一連のリエンジニアリングに投じた費用は 3 つの変種をリリースするまでに回収できた．

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，構造化分析

A Case Study on Migration to Product Line Development of an Automotive Part by Feature Oriented Approach

Yoichi Nishiura[1,a)] Masaki Asano[1,b)] Tsuneo Nakanishi[2,c)]

Received: May 23, 2019, Accepted: November 7, 2019

Abstract: This article describes a case study on migration to feature oriented product line development of an automotive body part, especially establishment of domain engineering process which has not been described in details so far. In the migration, we conducted reverse engineering with feature analysis and structured analysis. That facilitated system comprehension by developers, establishment of the domain engineering process having affinity with product's properties, and construction of the product line architecture. The cost for variant development after migration to product line development was successfully reduced to one-thirds of the estimated cost of clone-and-own development. The investment to construct core assets was recouped by release of the third variant.

Keywords: software product line, automotive body parts, feature analysis, structured analysis

1. はじめに

異なる顧客の要求，仕向地ごとの法令や文化にあわせて，さまざまな製品を開発，供給する製造業において，同一製品の異なる変種をまたいでプロセス化されたソフトウェア資産再利用を実現するプロダクトライン開発方法論 [1], [2] は，品質，コスト，工期を効果的に改善する開発のあり方として長きにわたって研究，実践されている．企業の開発現場における実践事例については，初期はもっぱら欧米を主とする海外企業によるもの [3], [4], [5], [6] であったが，この 10 年の間にわが国での事例 [7], [8], [9], [10] も多く報告されるようになった．しかしながら，たとえこうした成功事例が多く報告されていても，プロダクトライン開発方法論の開発現場への導入は，社内での成功事例がない間は

1 アイシン精機株式会社
Aisin Seiki Co., Ltd., Kariya, Aichi 448–8650, Japan
2 福岡大学
Fukuoka Univeristy, Fukuoka 814–0180, Japan
a) nishiura@elec.aisin.co.jp
b) asano_m@elec.aisin.co.jp
c) tun@fukuoka-u.ac.jp

© 2020 Information Processing Society of Japan

関係者に相当の覚悟を要するのが実情であろう．

プロダクトライン開発方法論はパラダイムであり，共通性と可変性の分離，アーキテクチャ指向開発，ドメインエンジニアリングとアプリケーションエンジニアリングの分離といった原理原則となる考え方は確立されていても，開発現場にそのまま導入できるような一般的かつ具体的な開発プロセスは与えられていない．プロダクトライン開発方法論の開発現場での実践には，上述のパラダイムをよく咀嚼したうえで，対象となる製品，さらには企業や開発現場の技術的，非技術的な制約を考慮し，開発プロセスを確立していかなければならない．プロダクトライン開発方法論に初めて取り組む企業や開発現場において，こうした開発プロセスを確立していく過程は探索的なものとなるはずであるが，既存の実践事例ではその過程があまり詳細には示されていない．

筆者らは，自動車部品メーカであるアイシン精機株式会社において，ある自動車ボディ系製品の変種全体を対象に，同社で初めてのプロダクトライン開発方法論の実践に取り組んだ．この自動車ボディ系製品は，10 年以上にわたって継続的に開発されてきた，多くの変種をかかえるものであり，過去に繰り返された流用・差分開発によって仕様や設計はきわめて見通しの悪いものとなっており，製品間でどこの仕様がどのように異なっているのかドキュメントやコードを精読しなければ分からない状態になっていた．プロダクトライン開発方法論導入の取り組みは成功を収め，現在，我々は他の自動車ボディ系製品についてもプロダクトライン開発方法論の導入を進めている．

本論文は，上述の自動車ボディ系製品の開発へのプロダクトライン開発方法論の導入に関して，これまでの実践事例では具体的に著されることのなかった，プロダクトライン開発方法論に基づくドメインエンジニアリングプロセスの確立に至るまでの探索的な過程を整理し，詳らかにすることを目的とする．著者らが採用したプロダクトライン開発方法論はフィーチャ指向アプローチであり，既存ソフトウェア資産のリバースエンジニアリング，ならびにプロダクトラインアーキテクチャの構築にはフィーチャ指向アプローチを採り入れた構造化分析手法 [11] を用いている．

以下本論文では，2 章において自動車ボディ系製品の一般的な特徴を述べ，当社におけるプロダクトライン開発方法論導入に至る背景と経緯を述べる．3 章では，フィーチャ分析と構造化分析によるリバースエンジニアリングを通して既存ソフトウェア資産を理解し，取り組み着手当時は抽象的かつ一般的であったドメインエンジニアリングプロセスを，リバースエンジニアリングの結果をもとに，製品の特性を考慮したものに詳細化していく過程について詳述する．4 章では本取り組みの効果を評価し，5 章では本取り組みを通して得られた教訓を検討として述べる．6 章で関連研究を紹介し，最後に 7 章において本論文を総括する．

2. プロダクトライン開発方法論導入に至る背景と経緯

2.1 ボディ系製品の特徴

アイシン精機はトヨタグループに属する自動車部品メーカであり，現在はトヨタグループ内外の，国内だけでなく海外の多数の一般自動車メーカや業務用輸送機器メーカに自動車部品を納めている．電動スライドドア，サンルーフ，メモリシート等のボディ系製品は当社の中核製品であり，これまで顧客たる車両メーカに，各社固有の要求を満足するボディ系製品を設計，製造，供給している．

消費者によって直接操作され認識される機会の多いボディ系製品は，自動車の利便性，ひいては価値を高めるものである．そのため一般消費者向けの自動車を製造するメーカ各社は，消費者に訴求するボディ系機能を数多く，また互いに競うように自社製品に搭載する傾向が強い．当社は，互いに競争する国内外複数のメーカにボディ系製品を納入しており，消費者から見れば同じような機能であっても内部は異なるところの多い変種群を製造している．ボディ系製品はしばしばユーザのスイッチ操作に応じてモータを回すだけの「簡単な」マイコン制御システムと見られがちであり，実際，ソフトウェアのソースコードだけを見ていればそのようなシステムに見える．しかし，ボディ系製品の開発上の難しさはソフトウェアのつくりよりも，その仕様の整理のしにくさにある．

一般消費者向け自動車のような大量生産品においてボディ系製品はコストダウン要求がきわめて強い．そのため，ボディ系製品は，安価で原始的な限られたセンサを用いて，ユーザ，車両，環境の状況を認識し，適切にその振舞いを変えるように設計されることが多い．何かしらの検知を行うために安易にセンサを追加することは許されず，むしろ後継製品では製造コストを下げるためにセンサが減らされていく傾向にある．さらに，どのような状況にあってもシステムは安全でなければならない．

こうしたシステムの物理的状況の認識や振舞いに関する仕様は複雑な条件判断をともなうものとなる．たとえば，各種機構の位置を知るのに安易に高価なポテンショメータを使えず，その目的のためにはモータの回転数がカウントされる．回転数カウントの結果，機構が所定時間内に期待した位置に至らない状況は，人や物の挟み込みやセンサの故障等さまざまに考えられる．他の条件と組み合わせて機構の位置を特定するか，それができない場合は少なくともシステムが安全に振る舞うようにする仕様を定めなければならない．

当社は，異なるメーカの異なる車種向けに部品として製品を納めており，製品仕様の最終決定権を有していない．メーカによって安全に係る設計思想が異なることもあり，同じ機能のボディ系製品であっても細かい点で仕様が異

© 2020 Information Processing Society of Japan

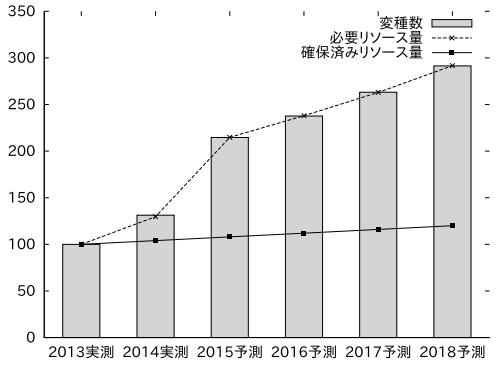

図 1　変種数と必要リソース量，確保済みリソース量の推移
Fig. 1　Transition of the annual number of variants, annual man-hour to be required for their development, and annual man-hour to be ensured.

なってくる．納入先からの要求の細かな相違も，ボディ系製品の仕様を複雑にし理解しにくいものとする一要因である．

2.2　開発体制上の背景

プロダクトライン開発方法論の導入を検討し始めた当時，当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，これら市場の地域的，文化的なニーズや法的規制の違いから，製品変種数が従来以上に増大する見込みであった．しかし，変種が増え，開発により多くのリソースが必要になっているにもかかわらず，地域や国内における技術者の不足から，当社が開発に投じられるリソースは抑えられたままとなっていた．**図 1** は，本論文実践事例の対象ボディ系製品における変種数，開発に必要なリソース量，確保が見込まれているリソース量の推移について，それらの 2014 年までの実績値と 2014 年時点での 2015 年以降の予測値を，2013 年を 100 とする相対値で示したものである[*1]．

製品変種数の増大と獲得を見込める技術者数の逼迫は，マネジメント層にプロダクトライン開発方法論の導入を決意させる動機となった．

2.3　ソフトウェア資産劣化の経緯

当社のボディー系製品の多くは製品立上げから 10 年以上経過しており，その間，ハードウェアもソフトウェアも継続的に変更されてきた．車種展開で新たな変種を開発する際，当社では，まずシステム技術者が顧客の要求を分析したうえで，対象車種向けのシステム仕様書を記述している．そのシステム仕様書に基づいてハードウェアとソフトウェアの担当者がそれぞれ，ハードウェアとソフトウェアの仕様書を記述する．各ボディ系製品が提供すべきサービスやフェールセーフ，フールプルーフといった制御の基本的思想は，当該製品群の立上げのころから大きくは変化していない．どのボディ系製品の設計思想も，立上げ当初においては，少人数のシステム，ハード，ソフトウェアの担当者間のすりあわせによってよく共有されていた．その後，車種展開や機能追加にともなう開発作業量の増加にともない，関係する開発者も増えたことで，開発者間で設計情報をより文書化されたかたちで共有することが必要となった．しかし，その際，制御方式を仕様書に詳細に記述することに主眼が置かれ，それまで担当者間のすりあわせによって共有できていた，仕様の目的や設計意図が仕様書には十分に記述されなかった．システム担当者からソフトウェア担当者が受け取るシステム仕様書には，状態遷移図や詳細なレベルの条件文が記載され，結果，ソフトウェア開発における要求分析工程は簡略化されるようになった．ソフトウェア担当者は受け取った仕様書の内容を忠実にソースコードに落とし込むことが是とされるようになった．

当社では，車種展開の際のソフトウェアの変種開発はいわゆるクローン&オウンによって行われてきた．すなわち，新しい変種に最も近い変種の製品を選び，違いが出ている箇所に限って，仕様書やコードを変更するような開発スタイルをとっていた．クローン&オウンは目標とする機能を最小の変更で実現しようとする点で一見合理的なように思える．しかし，クローン&オウンは場当たり的な設計や特定の車種向けの極端な局所最適化を招きがちであり，さらにそれが繰り返し行われることで仕様の背反を生じ，その解決のためにさらなる変更が必要となる．最終的に，ソフトウェアは複雑な依存関係を秘めた理解し難いものとなり，小さな仕様変更であっても影響分析，コードの修正，検証に相当の時間が費やされるようになる．開発現場はそのような状況に陥っており，特に影響分析には相当の時間が費やされるようになっていた．2.1 節で述べたように，限られた原始的なセンサの入力に基づいて複雑な条件判断のうえでシステムの状態を認識するようなボディ系製品においては，仕様（たとえばモータ回転数のカウント）とその目的（たとえばはさみこみの検知やセンサの故障の検知）との意味的乖離が大きい．仕様の目的が書かれていない仕様書しかない状況では，変更の影響の有無の判断には既存ソフトウェア資産の深い調査が必要になる．

短期化する開発工期と限られたリソースでの変種開発を実現するために，ソフトウェアの再利用のやり方を抜本的に改める必要があった．**図 2** は 2009 年を 100 としたときの，今回の取り組みの対象となったボディ系製品の変種開発時の生産性の推移を表している．ここで示されている生産性は，クローン元のコードから追加，変更，削除された行数を工数で割った値である．

ボディ系製品の変種開発の際の工数増大と生産性低下は，プロダクトライン開発方法論 [1], [2] の導入を開発現場に決意させる動機となった．

*1　本論文で報告する取り組みで 2015 年以降の必要リソース量の実測値はこの予測値より大きく削減できた．

© 2020 Information Processing Society of Japan

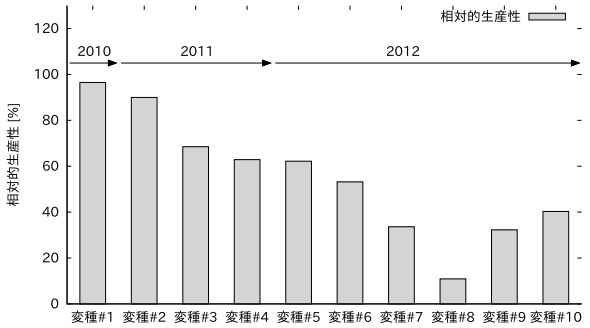

図 2　変種開発における相対的な生産性の推移

Fig. 2　Transition of relative productivities for variant development of a product.

2.4　プロダクトライン開発方法論導入の指針策定

以上に述べてきたボディ系製品の仕様に関する問題をふまえたうえで，当社の製品開発部門と開発支援部門とで以下の活動に取り組み，プロダクトラインの導入を進めることとなった．

- 既存の仕様書からの要求の復元：両部門は既存の仕様の目的，すなわち要求を獲得し，それらの要求を仕様とは明確に分けて記述し，それらの要求から仕様をたどれるようにする．
- 要求と仕様の再整理：要求と仕様をそれらの抽象度と共通性/可変性に基づいて再整理する．既存の仕様は，電圧や回転数といった物理量に近い表現の抽象度で精細に書かれており，一方で中間概念はあまり表現されておらず，そのため異なる変種で再利用しにくいものとなっていた．
- ソフトウェアアーキテクチャの定義：整理された要求，仕様に基づいて，可変性を局所化し，製品間での再利用性を高めたプロダクトラインソフトウェアアーキテクチャを定義する．

プロダクトラインの導入にあたって，製品開発部門単独で量産製品の開発と同時にコア資産の開発を行うことがリソース面で困難であった．このリソース問題を解決すべく，製品開発部門とは独立した開発支援部門にドメインエンジニアリングの責務を負わせることとなった．製品開発部門と開発支援部門のマネジメント層の合意の下，アプリケーションエンジニアリングを担う製品開発部門は既存資産の分析や調査に必要な人材と情報の提供，製品化決定の最終判断の責務を，ドメインエンジニアリングを担う開発支援部門は分析の実施，コア資産の開発，KPI（Key Performance Indicator）の定義と報告の責務を担うこととした．両部門のスポンサーシップのもと，この体制で少なくとも量産製品 1 車種目の開発完了までは，ドメインエンジニアリングとアプリケーションエンジニアリングを並行実施することが明確にされた．この体制はその後も継続しているが，今後，社内の組織改革で変更される可能性がある．

プロダクトライン開発の導入成否を測る KPI としては，再構築したコア資産のソフトウェア構造上の品質を評価するメトリクス，投資コスト回収可否を判断するコア資産開発に投資した費用の合計，開発完了時期，進捗状況が設定された．

3.　ドメインエンジニアリングプロセスの確立

3.1　ドメインエンジニアリングプロセス確立の過程

プロダクトライン開発方法論の導入にあたって，先行事例 [3], [4], [5], [7], [8], [9] に倣ってフィーチャ指向アプローチをとることとした．ドメインエンジニアリングを担う開発支援部門はフィーチャ分析 [12] を実施し，可変性を記述し，フィーチャの抽象度に基づく階層化と関心事の分離を図ることにした．ボディ系製品は比較的簡単な機能を有しており，最終ユーザに見える外部可変性はあまりなく，最終ユーザには見えない内部可変性が量的には支配的である．2.3 節で述べたように，すでに既存ソフトウェア資産の内部の製品変種間の違いは把握しがたいものとなっていたため，フィーチャ分析にあたっては既存ソフトウェア資産の内部の振舞いを詳細にリバースエンジニアリングする必要があった．そこで開発支援部門は，製品開発部門の知見者の製品知識に基づくトップダウンでのフィーチャ分析を進めると同時に，リバースでの構造化分析，すなわち既存ソフトウェア資産の内部の振舞いをデータフローとして整理，データフロー図として表現することを行った．このリバースエンジニアリングを通した既存ソフトウェア資産の理解の過程で，フィーチャを発見し，フィーチャモデルに加えていった．

プロダクトライン開発方法論の導入着手時に，当然ながら，ドメインエンジニアリングプロセスを定義したが，それは一般的な構造化分析手法にフィーチャ指向アプローチを採り入れた抽象的，かつ一般的なものであった．最終的にドメインエンジニアリングのプロセスは，**図 3** に示すようなものに落ち着いたが，導入着手時は図中の枠で囲まれていた部分のみがドメインエンジニアリングプロセスとして認識されていた．ドメインエンジニアリングプロセスが図 3 全体のように詳細化されたのは，既存ソフトウェア資産のリバースエンジニアリングでの観察結果が得られた後である．

図 3 はプロセス中の各工程間の結果がどの工程の実施を束縛するかを示すものであり，工程の順序を厳格に示すものではないことに注意されたい．各工程はおおよそこの図の依存関係で定まる順序で実施されてはいるものの，構造化分析の各プロセス，すなわちデータフロー分析，状態遷移分析の実施途中にソフトウェアの構造再検討や成果物各要素の命名や抽象度の調整のための工程の進退が生ずる．

3.2　既存ソフトウェア資産のリバースエンジニアリング

対象ボディ系製品のこれまでの開発に携わっていない技

© 2020 Information Processing Society of Japan

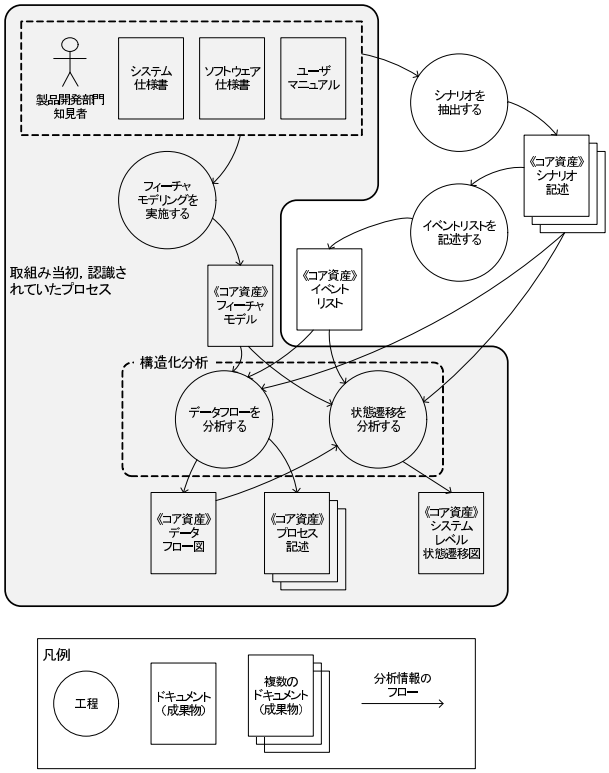

図 3　ドメインエンジニアリングのプロセス
Fig. 3　Domain engineering process.

術者が，既存ソフトウェア資産のフィーチャ分析ならびに構造化分析を実施し，実際に製品の開発に携わった知見者を交えたレビューを繰り返し行ったが，フィーチャの定義の仕方やデータフロー図の全体的な構造のあり方で議論は発散することが多かった．

当初は，システムを状態遷移システムとして，すなわち1つの制御プロセスが状態とイベントに基づいて制御を行うようなシステムとしてモデリングすることを試みたが，状態数の多い複雑なシステムとなることが予想されたため，再度，モデリングのあり方を見直すこととなった．

このような探索的な分析とレビューを繰り返す中で，最終的に以下の事柄が明らかになった．

- 確かにシステムは状態遷移システムであるが，どのボディ系製品でも通常作動状態，ユーザ保護状態，機構保護状態，システム初期化の4状態が，最も基本的な状態となっている．また，多くの機能がこの4状態で振舞いを変えている．
- 上述4状態以外の状態もあるものの，データフロー中の各プロセスで行う処理は逐次実行的なものがほとんどであり，各プロセスを状態遷移システムとして作る必要性はない．もしそうするとかえってシステムが複雑化する．
- システムの振舞いは，上述の4基本状態，ならびにイベント，条件，アクションの観点で画一的に整理でき，変種間の可変性は条件とアクションに多く含まれる．

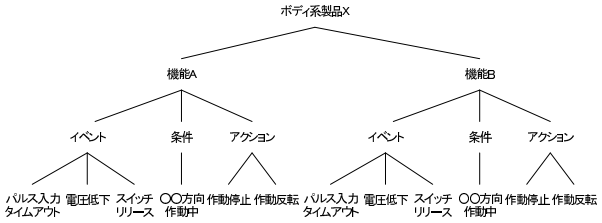

図 4　構造改善前のフィーチャモデル
Fig. 4　Feature model before structural improvement.

3.3　フィーチャ分析

既存ソフトウェア資産のリバースエンジニアリングでは，対象ボディ系製品の変種群のシステム仕様書，ソフトウェア仕様書，ユーザマニュアル等のドキュメントに対してフィーチャ分析を施し，フィーチャモデルを構築し，既存製品の仕様の全体像を把握し，そこからさらに2.3節で述べた経緯でドキュメントからは失われていた仕様の意図，つまりは要求を回復することとした．こうして得られたフィーチャモデルを改善し，製品変種間の共通性と可変性を分離することはもちろん，仕様項目の抽象度に基づく階層化および関心事の分離，粒度の均一化，網羅性の向上，ならびに仕様項目間の依存解析の整理を図ることとした．プロダクトラインにおけるフィーチャモデルの第一義の目的は共通性と可変性の分離であるが，今回の取り組みでは，抽象度に基づく階層化と関心事の分離が大きな効果を生み出すこととなった．

当初，開発支援部門は，システム仕様書，ソフトウェア仕様書，ユーザマニュアル等のドキュメントからフィーチャを見つけ出し，それらをフィーチャモデル上に配置することを試みた．システムの諸機能の制御をイベント，条件，アクションの観点で整理することでフィーチャを容易に見つけることができる．こうして見つけたイベント，条件，アクションに関するフィーチャを各機能のフィーチャの下に配置していった結果，**図 4** のようなフィーチャモデルができあがった．図に示されるように，このフィーチャモデルでは，同じフィーチャが異なる機能の下に何度も登場する．また，すでに述べたように，既存のドキュメントでは，流用・差分開発の繰返しの末，仕様項目の違いが何のために生じているのか不明確になっている．そのため，ドキュメントから見つかるフィーチャは概して抽象度が低く，大局的に理解しやすいフィーチャモデルを作ることは困難であった．たとえば，あるボディー系製品の仕様には，パルス入力タイムアウトのイベントが機能横断的に現れていたが，それらが何を想定して設定されている条件かはドキュメントには言及されていなかった．

大局的に理解しやすいフィーチャモデルを描くためには，ドキュメントに記述されている各条件を検査する目的，たとえば制御の結果を確認するためのものなのか，機構や回路の故障を検知するためのものなのか，意図しないユーザ

© 2020 Information Processing Society of Japan

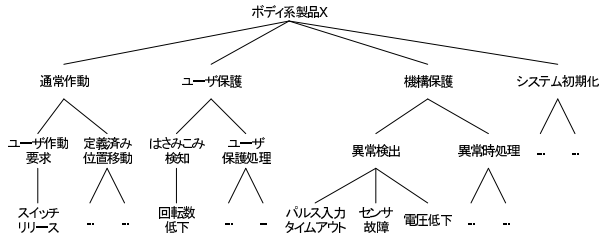

図 5　構造改善後のフィーチャモデル
Fig. 5　Feature model after structural improvement.

の操作に対応するためのものなのかを把握するような必要があった．そこで，製品開発部署のシステムおよびハードウェア担当者を知見者としてレビューに招き，ドキュメントに記載されていない上記の情報をヒアリングしながら，上位概念に相当するフィーチャの抽出を試みた．こうして抽出されたフィーチャをフィーチャモデル上に配置し，それを製品開発部門の知見者が確認するサイクルを，全車種全仕様をカバーするまで繰り返し行った．この過程において，システムの振舞いは通常作動状態，ユーザ保護状態，機構保護状態，システム初期化の基本 4 状態とイベント，条件，アクションの観点で画一的に整理できるという認識が，開発の関係者間で共有された．結果，フィーチャモデルは**図 5** のようなものに改善された．

最終的に**図 6** のようなフィーチャモデルが得られた．フィーチャモデルの上の層には Why，あるいは What に相当する目的レベルのフィーチャ，下の層には How に相当する手段レベルのフィーチャが配置され，それらが全体/部分，あるいは汎化/特化の関係で関連付けられている．目的レベルのフィーチャは，ドキュメントからは失われていた要求に相当するものであり，今回の活動で製品開発部門の知見者から聞き出して明らかになったものである．一方，手段レベルのフィーチャはこれまでのドキュメントに詳細に書かれていたものであり，開発支援部門が当初から見つけ出していたものである．こうしたフィーチャモデル改善の過程で，ドキュメント中に失われていた要求が回復され，それらの仕様との対応が明るみに出ていった．また，仕様やコードの複雑化の要因となっていた仕様項目の部分的重複が明らかになり，整合のとれた仕様項目の定義が進み，また仕様項目間の依存関係が整理された．立証に寄与するデータをとっていなかったため主観的な評価となるが，イベントと条件の理解容易性は著しい向上があった．

目的レベルのフィーチャには変種間の可変性はあまり見られず，可変性は手段レベルのフィーチャに見られた．図 6 の例では，「◯◯機能」を構成する，「ユーザ作動指示」「車両状態」「作動領域」はすべての変種に共通に現れる概念であり，変種間の違いはこれらのフィーチャにバインドされる値の定義（すなわちイベントと条件），それらの組合せから定まるアクションの定義に現れた．

フィーチャ分析では，発見されたフィーチャを抽象度に

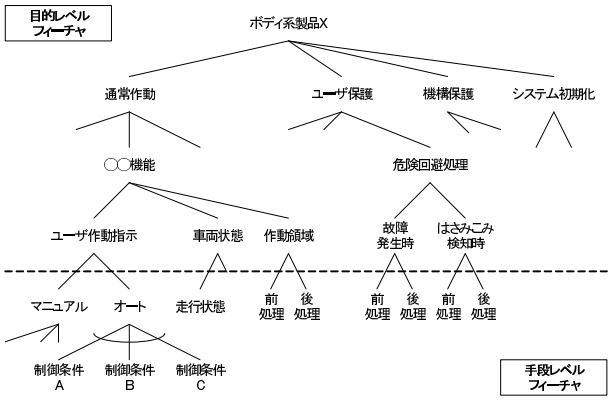

図 6　最終的なフィーチャモデル
Fig. 6　Final feature model.

基づいて階層化し，また何かしらの関心事に関して異なる事柄を異なるフィーチャとして定義し，フィーチャモデルを構築する必要がある．フィーチャモデルは，Kang による FORM の 4 層モデル [13]，ならびに文献 [14], [15] で紹介されているガイドラインに従って，フィーチャの階層構造を定めるようにしている．構造改善前のフィーチャモデルはおおよそ機械的に構築できるものであるが，その後，受け入れ可能な抽象化構造のフィーチャモデルに改善できるかは分析者の能力に依るところが大きい．これは本ドメインエンジニアリングプロセスに限らず，フィーチャ指向アプローチに共通する課題といえる．

フィーチャモデルで明らかになった共通/可変構造により，実現案のレベルであったデータフロー分析，ひいてはプロダクトラインアーキテクチャを完成形に詳細化させる方針が定まった．

3.4　構造化分析

フィーチャ指向アプローチでは，フィーチャモデルが主たるモデルとなり，フィーチャモデルは他のモデルの有り様を束縛し，また他のモデルへの追跡可能性保証の起点となる．既存ソフトウェア資産のリバースでの構造化分析にあたっては，フィーチャ分析の定める抽象化構造と関心事の分離構造に従うように，すなわちフィーチャの意味境界とデータフロー中のプロセスの定義，データフローの分離/集約構造，プロセスのサブプロセスへの分割を揃えるようにした．しかし，フィーチャ分析と構造化分析の実施は決して一方通行ではなく，データフロー分析，状態遷移分析工程の結果が違和感を覚えたり，複雑なものになったりする場合は，関係者が考えるそれらのあるべき姿をふまえて，フィーチャモデルを修正することも多かった．そのため，フィーチャ分析と構造化分析は交互に実施されるイタレーティブなものとなる．

構造化分析のプロセスは文献 [16] で示されているものを採用している．文献 [16] に示されている構造化分析プロセスは，まずはデータの加工，抽象化/具象化を分析するデー

© 2020 Information Processing Society of Japan

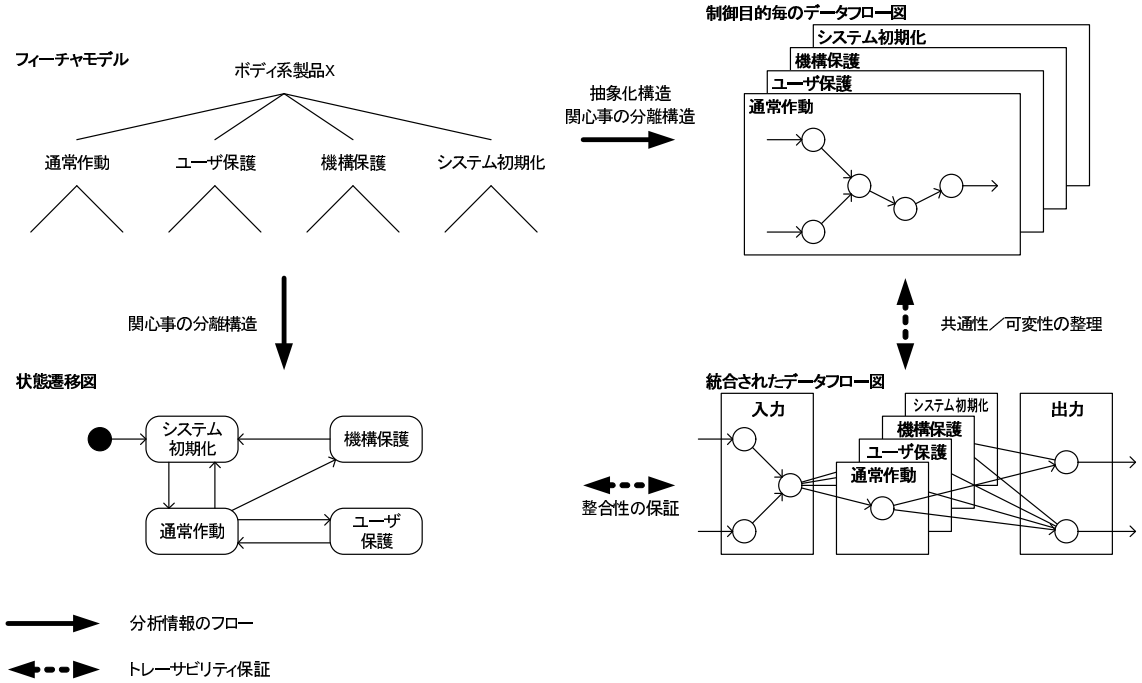

図 7　フィーチャ分析と構造化分析の相互影響
Fig. 7　Simultaneous implementation of feature analysis and structured analysis.

タフロー分析を実施してデータフロー図を作成し，さらに必要に応じてプロセスの活性/不活性や振舞いの状態依存性の変化を制御フローとともに分析する状態依存分析を実施し，データフロー図に付随する状態遷移図を記述する．

当初は，コンテキストレベルからトップダウンでデータフロー図を構築し，システムのプロセスへの分割を試みた．しかし，目的の異なる，複数の利用シーンにおける制御を記述しようとしたため，抽象度や粒度が統一されていない多様な入力が特定の制御プロセスに集約され，制御プロセスが肥大化，複雑化する結果となった．

リバースエンジニアリングによってシステムの基本 4 状態が見出された後には，データフローを簡潔にするべく，以下に述べるアプローチをとって制御プロセスに入力されるデータの種類を低減させることにした．

- 制御目的にあわせたデータフロー図の分離：同じ機能であっても，フィーチャモデルに最上位概念として現れる通常作動，ユーザ保護，機構保護，システム初期化のための制御を，それぞれで独立のデータフロー図として分離してモデリングした．各データフロー図の入力はその目的を実現するうえで必要なものにとどめ，制御プロセスの肥大化を抑制した．
- 入出力データの抽象化：データフローにおける入力データが何のために必要な情報なのかをフィーチャモデルと比較しながら抽象化した．抽象化されたデータは最終的には以下のいずれかに分類される．
  - 入力データ
    - ∗ イベント：ユーザの操作や機構の作動の許可/禁止指示等の機構の作動の契機．
    - ∗ 条件：機構の現在の位置や作動の方向等機構の作動の前提条件となる状態．
  - 出力データ
    - ∗ アクション：機構の基本作動，前作動，後作動，停止に係る制御出力．
- 抽象化した入力データによる制御判断：制御プロセスには極力，抽象化されたデータを入力するように設計した．その際，記述されたデータフローですべての想定されるシナリオを実現できるか検証した．
- 状態遷移：通常作動，ユーザ保護，機構保護，システム初期化の各制御目的の制御プロセスが動いている状態，ならびにそれら制御プロセスを起動，あるいは停止させるようなデータフローをイベントに対応させて状態遷移図を作成した．ここでもより抽象度の高いデータフローを状態遷移図におけるイベントに対応づけるようにすることで，イベントと状態が増加しないように検討を進めた．

図 7 にフィーチャ分析と構造化分析の相互影響の概要を示す．フィーチャ分析においてフィーチャ単位での仕様項目の抽象度の階層化と関心事の分離が図られる．フィーチャモデルの幅方向で表現される関心事分離に基づいて，データフロー図の記述の観点（すなわち制御目的）とシステムレベルの状態遷移図の状態が見出された．また，フィーチャモデルの深さ方向で表現される抽象度階層に基づいて，データフロー図の記述の際の入出力に係るデータフローの抽象化/具象化フローが見出された．制御目的別のデータフロー図におけるデータフローの共通性と可変性を加味し，またシステムレベルの状態遷移図との整合をとるよう

© 2020 Information Processing Society of Japan

に，システムレベルの統合されたデータフロー図を完成させた．

3.5 確立されたドメインエンジニアリングプロセス

フィーチャ分析と構造化分析による既存ソフトウェア資産のリバースエンジニアリングを進めるうちに，図 3 のドメインエンジニアリングプロセスが最終的なものとして確立された．

システムの機能は，さまざまな条件下でのシステムの振舞いをシナリオ記述として記述し，さらにそれらを 1 つのイベントリストとして俯瞰的にまとめあげる．各機能の振舞いはデータフロー図と同図中の各プロセスの仕様として記述される．具体的には，イベントリストに書かれたイベントがシステムに入力され，システムからの応答が得られるまでの，システム内部での一連のデータ変換をデータフロー図としてモデリングし，データフロー図中の各プロセスの仕様をプロセス仕様としてまとめる．また，データフロー分析中明らかになった状態依存性のあるデータ変換やプロセス挙動は，構造化分析中の状態遷移分析の工程で明らかにする．このプロセスは文献 [16] のものと変わらない．

対象ボディ系製品は，通常作動状態，ユーザ保護状態，機構保護状態，システム初期化の 4 状態を基本的な状態として持ち，多くの機能がこの 4 状態で振舞いを変えている．システムの振舞いは，この 4 基本状態，ならびにイベント，条件，アクションの観点で画一的に整理できる．各機能の振舞いは，基本 4 状態ごとにデータフロー図とその図中の各プロセスの仕様として作成される．これらを 1 つのモデルとしてまとめると，複雑で可読性に劣るモデルとなるためである．最終的には，制御プロセスは各機能基本 4 状態ごとに用意され，イベントを順次抽象化しそれら制御プロセスに入力するまでの処理，それら制御プロセスの出力をレスポンスまで具体化するまでの処理は共通化される．

構造化手法によるモデリングでは，抽象度に基づく階層化や関心事の分離に基づくプロセスの分割や階層化を適切に行う必要がある．確立されたドメインエンジニアリングプロセスでは，プロセスの分割と階層化は，フィーチャモデルの定める抽象化ならびに関心事分離の構造と整合をとる．

最終的にコア資産となるのはフィーチャモデル，システムレベル状態遷移図，イベントリスト，データフロー図，プロセス仕様書，そしてこれらに基づいてスクラッチから書き起こされたコード資産である．

その後，当社で開発している他のボディ系製品のプロダクトライン開発においても，この確立されたドメインエンジニアリングプロセスは適用され，その後さらなる改善も施されている．これらのボディ系製品はいずれも通常作動状態，ユーザ保護状態，機構保護状態，システム初期化の 4 状態を有している．確立されたドメインエンジニアリングプロセスでは状態遷移分析を実施しているが，この 4 状態以外の状態が定義される事例は現時点ではない．

4. 評価

4.1 フィーチャモデル

最初に作成したフィーチャモデルはフィーチャが約 650 ある大きなものとなった．対象ボディ系製品を分析した結果，現在のソフトウェア制御仕様は条件の重複や依存が多数あり，仕様そのものが複雑化していることが明らかになった．たとえば，従来のフェールセーフに関する仕様記述では，異常が発生しうるいくつかの制御条件ごとに，検出対象の故障とその後の処理が規定されていた．それらの組合せに加え，さらに機構や OEM 供給先からのフェールセーフ要求の違いを考慮する必要があり，膨大な条件の組合せを検討してゆく必要があった．

フィーチャモデルの改善を行う過程で，このような組合せが，仕向け地や作動条件とは無関係に，ユーザの作動指示，機構の作動領域，車両の状態といった限定的な条件の組合せで，異常検出後の振舞いが決定できることが明らかになった．制御条件の統廃合を行いフィーチャモデルの整理を進めた結果，従来と同等の機能を保ったまま，フィーチャ数を当初の約 22%となる約 140 まで削減し，フィーチャモデルの規模と構造を簡単化することに成功した．

4.2 アーキテクチャ

フィーチャ分析と構造化分析による，抽象度の高い情報をもとにした設計を行ったことで，各モジュールの責務が明確で，モジュール間の依存が疎な全体アーキテクチャを新たに構築することができた．モジュール間の依存関係を**図 8** に示す[*2]．

ソフトウェアの構造面の改善を定量的に評価すべく，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調査した．評価に用いた構造影響度は文献 [17] で提案されている，式 (1) において IR と定義される量である．

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

ここで S はソフトウエアを構成する全モジュールを要素とする集合を，$dep(m)$ は m 自身も含め，モジュール m に依存している全モジュールを要素とする集合を，$LoC(m)$ はモジュール m のソースコード行数を，$SLoC$ はソフトウエア全体のソースコード行数を表す．

この影響度は，依存モジュール数の単純な平均ではなく，変更の影響をモジュール数ではなくソースコード行数ベースで評価し，かつモジュールの大きさによる変更の生

*2 機密上，図の画質を意図的に落としている．

© 2020 Information Processing Society of Japan

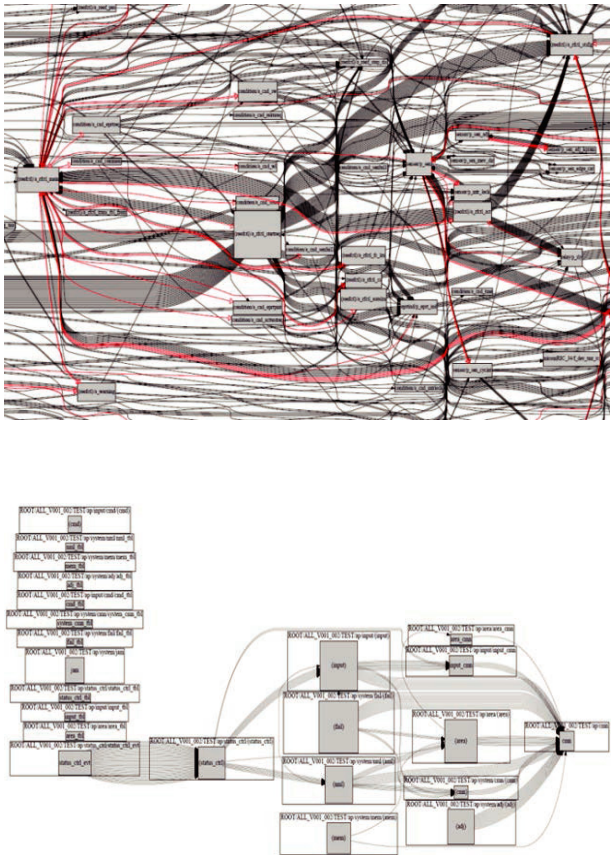

図 8　アーキテクチャの改善：改善前（上）と改善後（下）
Fig. 8　Architecture improvement: before improvement (above) and after improvement (below).

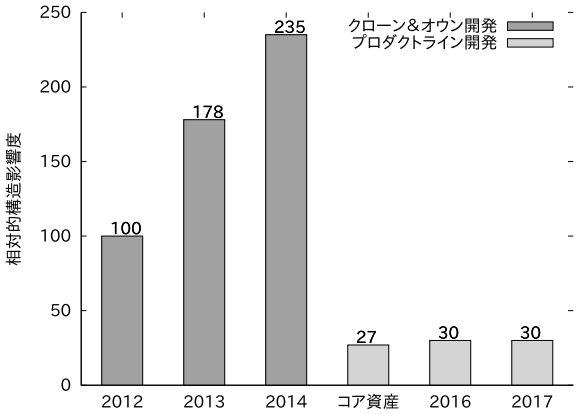

図 9　相対的構造影響度
Fig. 9　Structure impact ratio.

じやすさを加味する定義がなされている．より大きいモジュールは変更される可能性が高いため，上記の定義では $LoC(m)/SLoC$ を，当該モジュールの変更によって影響を受けるソースコード行数に乗じている．

図 9 では，2012 年に出したある代表的な変種の構造影響度を 100 として，各年の代表的な変種の相対的な構造影響度を示している．また，図中のコア資産はコア資産部分の相対的構造影響度を意味する．クローン&オウンを続けてきた従来製品は，2014 年の変種が最悪の構造影響度となっているが，プロダクトライン開発に移行した 2016 年以降の変種の構造影響度はそのときの 1/8 となっている．

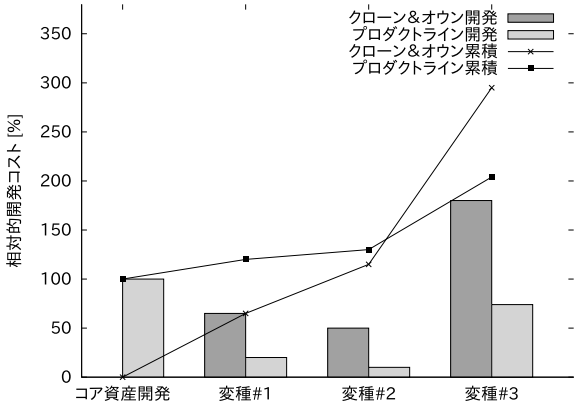

図 10　相対的開発コスト
Fig. 10　Relative development cost.

4.3　開発コスト

課題となっていた開発コスト，すなわち変種ごとの開発工数も削減できた．図 10 に示すとおり，コア資産からの変種開発に要した工数実績値は，従来のクローン&オウンで変種を開発した場合に要するであろう工数見積り値のおおよそ 1/3 となっている．ここに示されている工数は同一製品の異なる変種のものであり，プロダクトライン開発による工数実績値には，コア資産からの変種導出に要した工数のみが表示されている．プロダクトライン分野でよくいわれるように [18]，我々の事例においても 3 つ目の変種において，コア資産開発に投じた費用を回収することができた．

4.4　副次的な効果

アーキテクチャにおける各コンポーネントの責務とコンポーネント間のインタフェースを明確にしたことで，以下のような副次的効果も得られた．

- シミュレーション・自動化技術の導入：コンポーネント単位でのシミュレーション環境を用いた評価が容易になった．また，これらの導入が進むにつれてこれまで実機を用いて行っていた評価を，シミュレーション環境で自動実行するようになり，さらなる工数低減が見込めるようになってきた．
- 分散拠点での開発：これまでは 1 つの開発拠点でソフトウェア全体の開発を行う必要があったが，改善後はコンポーネント単位で，分散された拠点で開発を行うことが可能となり，状況に応じて最適な開発体制をとれるようになった．
- MCU の 32 ビット化や Autosar への対応：車載分野においては，製品によって 16 ビットから 32 ビットの MCU への載せ替えが必要となる．従来は，ほぼフルスクラッチでの開発が必要となっていたが，今回の取

© 2020 Information Processing Society of Japan

り組みの結果，Autosar 等のプラットフォームのコンポーネントを差し替えるだけで容易に対応できるようになった．

- システム開発工程からソフトウェア開発工程への移行プロセスの改善：これまで目が向けられていなかったシステム仕様書の記載内容の後工程への影響に目が向けられる契機となり，システム担当者とソフトウェア担当者が共同でプロセスを見直すこととなった．

5. 検討

これまでに述べたプロダクトライン開発方法論導入の取組みの中で得られた教訓を以下にまとめたい．

教訓 1：フィーチャ分析では，ソフトウェア以外の知見者も含め，開発関係者間で合意できる重要概念の発掘に注力する．

プロダクトライン開発の目的は，コア資産を再利用することにより変種開発の品質，コスト，開発期間を短縮することである．しかしながら当社では，製品開発部署でのソフトウェア設計者とシステムおよびハードウェア設計者との，仕様や設計の意図に関わる意思疎通が，変種の開発プロセスそのものよりも重要な問題であった．そのため，これらの開発関係者と共同でのフィーチャ分析を通して，変種開発の過程でドキュメントから失われた要求の回復とそれに基づく仕様の再整理を行うことがプロダクトライン開発方法論の導入に取り組んだ当初の重要な活動となった．

3.3 節で述べたように，フィーチャ分析の過程において，従来は明示的に記述，共有することのなかった仕様に関する重要な概念，すなわち制御仕様がユーザ操作，ユーザ保護，機構保護，初期化の 4 つの概念の下に整理できることが明らかになった．これら重要概念の発掘により，開発関係者間の意思疎通が改善し，これらを基軸とした制御仕様の整理が進んだ結果，相互の依存の少ない整理された構造のフィーチャモデルが完成し，さらにフィーチャモデルの与える抽象化と関心事分離の構造に従って構築されるソフトウェア構造の複雑さの軽減につながった．

教訓 2：アーキテクチャの定義は重要概念の発掘を待ってから実施する．

3.4 節で述べたように，本取り組みでは，フィーチャ分析の結果に従うよう構造化分析を行うことを原則としているが，構造化分析の結果が開発関係者にとって好ましいものとならない場合はフィーチャ分析の修正が行われ，フィーチャ分析と構造化分析はイタレーティブなかたちで実施された．制御仕様上の重要概念が発掘されるまでにできた構造化分析の結果は，違和感を覚えるものであったり，複雑なものとなったりで活かされることはなかった．一方，重要概念発掘後，構造化分析は円滑に進んだ．フィーチャとプロセスの抽象度や責務面での境界合わせのために，フィーチャ分析とアーキテクチャ定義のための構造化分析を部分的にイタレーティブに実施しなければならない状況は避けえないが，フィーチャ分析の初期は重要概念の発掘を優先して構造化分析は必要最小限にとどめ，重要概念が見つかってからアーキテクチャ定義のための本格的な構造化分析に着手するほうが，構造化分析の指針も立ち，手戻りが小さくなることが教訓として得られた．

6. 関連研究

フィーチャ分析の第一義は製品の違いの記述にあり，報告されるほとんどのケーススタディにおいて，フィーチャモデルがその目的のために使用されている．車載システムにおけるプロダクトライン開発の初期の例である，Thiel らの事例 [3] では，プロダクトラインの可変性とプロダクトラインアーキテクチャの間の変化点の管理を目的にフィーチャモデルを用いている．

フィーチャモデルを用いずに可変性を記述する例としては，Gomaa によるプロダクトライン開発のための UML によるモデリング方法論があげられる [19]．フィーチャ分析の前にユースケースモデリングを実施する同方法論は，ユースケースシナリオの記述を通して製品間の可変性を明らかにし，その後，何かしらの機能の選択にあわせて同時に活性化される変化点をフィーチャとして定義する．同方法論では，フィーチャの選択制約は記述しても相互の意味的関係を記述することはなく，フィーチャは一緒に再利用され活性化される変化点をタグづけするための概念にすぎない．

Kang らの FORM [13] に代表されるフィーチャ指向アプローチは，フィーチャを単なる選択制約の記述や変化点へのタグのためのみに用いるのではなく，その意味や意味に基づくフィーチャ間の関係をソフトウェアの理解や構築に積極的に活かしていこうとする方法論といえる．

Kang らは文献 [4] において，レガシーの家庭向けサービスロボットのフィーチャ指向アプローチによるリエンジニアリングのケーススタディを発表している．Kang らはレガシーアプリケーションから現状のアーキテクチャを発掘するとともにフィーチャモデルを構築し，構築されたフィーチャモデルの改善を通して新しいアーキテクチャを構築し，プロダクトライン向けのコンポーネントを開発している．また，文献 [5] では，クレジットカードの認証システムにおいて，そのフィーチャモデルを構築し，仕様の変更が生じやすい箇所を特定し，変更に強いかたちにコンポーネントの階層構造を整理している．Kang らはこれらのケーススタディを通して，フィーチャモデルは，ソフトウェアを俯瞰し，階層化されたアーキテクチャを構築するうえで有益な情報を与えるツールであることを主張している．本論文はその主張を支持する立ち位置にある．加えて本論文では，対象製品群の要求，仕様，設計，実装のあらゆる成果物を通して支配的な概念を発掘すること，ならび

© 2020 Information Processing Society of Japan

にそれらの概念が手戻りの少ないアーキテクチャの構築を可能とすることを主張する．

大塚らはフィーチャ指向アプローチによるネットワーク機器のファームウェアのプロダクトライン開発を行っている [7], [8]．大塚らは FORM [13] の提唱するフィーチャの 4 層モデルに基づく抽象化，ならびにフィーチャレベルでの関心事の分離を行っている．大塚らもまた，フィーチャモデリングがステークホルダ間の意思疎通やソフトウェアを俯瞰するうえで有用であり，プロダクトライン内で再利用される部品の共通化が促進されることを述べている．また，フィーチャモデル上で変更の影響の大きい箇所を特定して整理し，当該箇所をコード生成するプロセスを確立することで，開発コストの大幅な削減を実現している．

Lee らは，プロダクトラインのソフトウェアアーキテクチャをオブジェクト指向で設計するための，フィーチャ指向のアプローチを提唱している [20]．そのプロセスでは，フィーチャ分析は抽象化と関心事分離を行うために用い，ソフトウェアアーキテクチャの設計につながるオブジェクトの発見は構築されたフィーチャモデルで律するようにしている．Lee らは，フィーチャがそのカテゴリに応じてどのようにオブジェクトに詳細化されるかのガイドラインを定義している．カテゴリは役割や抽象度によってレイヤリングされている．本論文で提案するドメインエンジニアリングプロセスは，Lee らのアプローチに学んでいるが，オブジェクトモデルの構築以前に要求や仕様の段階的詳細化によりフォーカスをあてており，システム分割は構造化分析に基づいて行っている．自動車システムの開発では安全上の説明責任が厳しく求められる．本論文のプロセスは，要求からコンポーネントへ至る成果物のトレーサビリティを保証することで，その説明責任を果たせるよう設計されている．

上述の手法は要求や仕様を起点にフィーチャモデルを構築していく手法であるのに対し，Iida らの電子制御ブレーキのプロダクトライン開発事例 [9] では，SysML の内部ブロック図として記述されるシステムモデルを起点にフィーチャモデルを構築し，さらにフィーチャモデルが与える可変性情報を参考に，システムモデルからソフトウェアアーキテクチャを構築するアプローチをとっている．このアプローチでは，フィーチャモデルは内部ブロック図中のコンポーネントの接続や配置の可変性の記述，実機テストの結果で生じる影響の把握に使用される．フィーチャモデルからソフトウェアアーキテクチャの導出を図る点で，本論文と同じくフィーチャ指向アプローチといえるが，抽象化構造や関心事分離はフィーチャモデル由来というよりは，フィーチャモデルの起点となる内部ブロック図の入れ子構造に由来するところが少なからぬものと思われる．

7. まとめ

本論文では，ある車載ボディー系製品を，従来のクローン&オウン開発からプロダクトライン開発方法論に基づく開発へと移行する事例を詳述した．導入においては，まずフィーチャ分析と構造化分析によるリバースエンジニアリングを実施し，ドキュメントから失われていた仕様項目の目的，すなわち要求を回復し，仕様項目の抽象度に基づく階層化と関心事の分離，仕様項目間の依存関係の記述を実施した．その過程において，システムの重要概念が発掘され，開発関係者のシステム理解が深化した．結果，当初は抽象的かつ一般的だったドメインエンジニアリングプロセスが対象製品の特質をふまえたものに具体化されると同時に，フィーチャ分析と構造化分析の指針が立ち，プロダクトラインアーキテクチャの構築が進んだ．一連の活動により，対象ボディ系製品のソフトウェア構造の変更に対する構造影響度は従来の 1/8 となり，変種開発に要するコストは 1/3 になった．一連の活動に要した費用も，3 つ目の変種をリリースするまでには回収できた．

今回確立されたドメインエンジニアリングプロセスは，他の車載ボディー系製品のプロダクトライン開発方法論による開発への移行にも適用されている．今後は生産性の悪化した製品を今回のようにプロダクトライン開発方法論への開発に移行するだけでなく，新規の製品においても最初からプロダクトライン開発できるような仕組みを整備していきたい．

参考文献

[1] Clements, P. and Northrop, L.: *Software Product Lines: Practice and Patterns*, Addison-Wesley (2001).

[2] Pohl, K., Böckle, G. and Linden, F.v.d.: *Software Product Line Engineering: Foundation, Principles, and Techniques*, Springer (2005).

[3] Thiel, S. and Hein, A.: Modeling and Using Product Line Variability in Automotive Systems, *IEEE Software*, Vol.19, No.4, pp.66–72 (2002).

[4] Kang, K.C., Kim, M., Lee, J. and Kim, B.: Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets: A Case Study, *Proc. 9th Int. Conf. Software Product Lines* (*SPLC2005*), pp.45–56 (Sep. 2005).

[5] Kang, K.C., Lee, J.J., Kim, B., Kim, M., Seo, C.W. and Yu, S.L.: Re-engineering a Credit Card Authorization System for Maintainability and Reusability of Components: A Case Study, *Proc. 9th Int. Conf. Software Reuse* (*ICSR2006*), pp.156–159 (June 2006).

[6] Linden, F.J.v.d., Schmid, K. and Rommes, E.: *Software Product Lines in Action*, Springer (2007).

[7] 大塚　潤, 河原畑光一, 岩崎孝司, 内場　誠, 中西恒夫, 久住憲嗣，福田　晃：大規模移動体ネットワーク機器ファームウェア開発へのソフトウェアプロダクトライン適用事例，組込みシステムシンポジウム 2011（ESS2011）予稿集，pp.26-1–26-10 (2011).

[8] Otsuka, J., Kawarabata, K., Iwasaki, T., Uchiba, M., Nakanishi, T., Hisazumi, K. and Fukuda, A.: Small Inexpensive Core Asset Construction for Large Gainful Prod-

© 2020 Information Processing Society of Japan

uct Line Development: Developing a Communication System Firmware Product Line, *Proc. 15th Int. Software Product Line Conf.* (*SPLC2011*), p.5 (Aug. 2011).
[9] Iida, T., Matsubara, M., Yoshimura, K., Kojima, H. and Nishino, K.: PLE for Automotive Braking System with Management of Impacts from Equipment Interactions, *Proc. 20th Int. Systems and Software Product Line Conf.* (*SPLC2016*), pp.232–241 (Sep. 2016).
[10] Hayashi, K., Aoyama, M. and Kobata, K.: Agile Tames Product Line Variability: An Agile Development Method for Multiple Product Lines of Automotive Software Systems, *Proc. 21st Int. Systems and Software Product Line Conf.* (*SPLC2017*), pp.180–189 (Sep. 2017).
[11] DeMarco, T.: *Structured Analysis and System Specification*, Prentice-Hall (1979).
[12] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E. and Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Analysis, Technical Report CMU/SEI-90-TR-21, SEI/CMU (Nov. 1990).
[13] Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E. and Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architecture, *Annals of Software Engineering*, Vol.5, No.1, pp.143–168 (Jan. 1998).
[14] Lee, K., Kang, K.C. and Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, *Proc. 7th Int. Conf. Software Reuse* (*ICSR*), pp.62–77 (2002).
[15] 中西恒夫，久住憲嗣，福田　晃：ソフトウェアアーキテクチャ事前設計を目的とするフィーチャモデリングのアンチパターン，組込みシステムシンポジウム 2009（ESS2009）論文集，pp.179–187 (Oct. 2009).
[16] SESSAME WG 2：組込みソフトウェア開発のための構造化モデリング，翔泳社 (2006).
[17] 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 (Sep. 2013).
[18] McGregor, J.D., Northrop, L.M., Jarrad, S. and Pohl, K.: Initiating Software Product Lines, *IEEE Software*, Vol.19, No.4, pp.24–27 (2002).
[19] Gomaa, H.: *Designing Software Product Lines with UML*, Addison-Wesley (2005).
[20] Lee, K., Kang, K.C., Chae, W. and Choi, B.W.: Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse, *Software: Practice and Experience*, Vol.30, No.9, pp.1025–1046 (2000).

西浦 洋一

1997 年千葉大学工学部卒業．1999 年千葉大学大学院自然科学研究科物質工学専攻修士課程修了．2005 年アイシン精機株式会社入社．車載組み込みソフトウェアの開発に従事．

浅野 雅樹

2009 年名古屋工業大学工学部卒業．同年アイシン精機株式会社入社．車載組み込みソフトウェアの開発に従事．

中西 恒夫　（正会員）

1993 年大阪大学工学部通信工学科卒業．1998 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（工学）．2014 年より福岡大学工学部電子情報工学科教授．組み込みシステムを対象とするソフトウェア工学の研究に従事．

© 2020 Information Processing Society of Japan