自動車ボディ系製品のプロダクトライン開発への移行に関する事例報告

西浦 洋一[1,a)]　浅野 雅樹[1,b)]　中西 恒夫[2,c)]

概要：アイシン精機ではこれまで，自動車ボディ系製品を制御するソフトウェアをいわゆるクローン＆オウンによってバリアントを開発してきた。しかし，度重なるバリアント開発でソフトウェアはわかりづらいものとなり，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ってきたため，プロダクトライン開発への移行を図ることにした。移行では，既存ドキュメントを対象にフィーチャ分析を行い，ドキュメントから失われていた各仕様項目の目的（すなわち要求）を回復するとともに，仕様項目の抽象度に基づく階層化と関心事の分離を図った。得られたフィーチャモデルが定める抽象度の階層構造と関心事の分離構造に基づき，データフロー分析と状態遷移分析を実施し，プロダクトラインアーキテクチャをはじめとするコア資産を開発した。ソフトウェア構造が著しく改善し，ソフトウェアの理解性が向上した結果，バリアント開発に要するコストを大きく削減することができた。一連のリエンジニアリングに投じた費用は 3 つのバリアント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 Shift to Product Line Development of Automotive Parts

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

Abstract: Aisin Seiki Co., Ltd. had developed software variants controlling automotive body parts in the so-called clone-and-own fashion. Since repeated development of variants made software incomprehensible, impact analysis, modification, and verification of codes need a considerable amount of time even for small change of specification. That drove the comapany to develop variants under the paradigm of software product lines. Feature analysis applied to the existing documents facilitated hierarchy based on abstraction levels and separation of concerns of the specification items as well as recovered objectives of specification items (namely requirements) lost in the documents. Data flow and state transition analyses were conducted based on the hierarchy and separation given by the feature model to construct core assets including the product line architecture. The software structure was improved and became comprehensible by these reengineering activities. The cost for variant development was successfully reduced. 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

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。現在はトヨタグループ内外，国内のみならず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。ボディ系製品は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，供給してきた。社は欧州ならびに開発途上国の市場にて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ちがいから，製品のバリアントが従来以上に増大する見込みであった。しかし，バリアントが増え，開発により多くのリソースが必要

[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 University, Johnan, Fukuoka 814–0180, Japan
[a)] nishiura@elec.aisin.co.jp
[b)] asano_m@elec.aisin.co.jp
[c)] tun@fukuoka-u.ac.jp

になっているにも拘わらず，地域や国内における技術者の不足から，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る。図 1 はあるボディ系製品におけるバリアント数，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]。

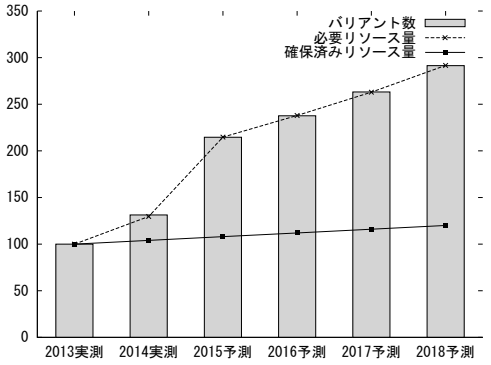

図 1 バリアント数と必要リソース量，確保済みリソース量の推移

アイシン精機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。車種展開で新たなバリアントを開発する際，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頃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伴う開発作業量の増加に伴い，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，システム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。ソフトウエ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。

[*1] 本稿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。

アイシン精機では，車種展開の際のソフトウェアのバリアント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，新しいバリアントに最も近いバリアントの製品を選び，ちが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，よく知られているように，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。開発現場はまさにその状況に陥っており，短期化する開発工期と限られたリソースでのバリアント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 としたときの車種展開毎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，分子はクローン元のコードから追加，変更，削除された行数を，製品開発期間全体の時間で割った値である。ボディ系製品開発におけるこのような状況が，開発現場にプロダクトライン [1], [2] のパラダイムを導入する大きな動機となった。

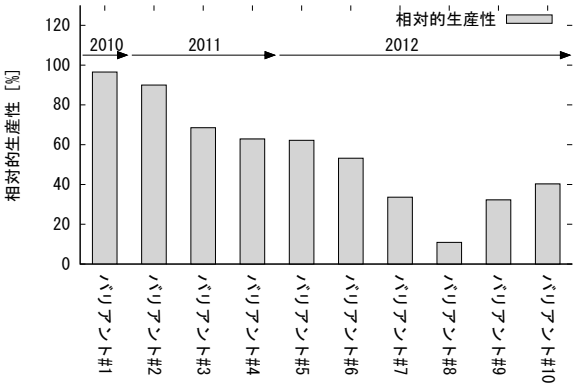

図 2 バリアント開発における相対的な生産性の推移

以下本稿では，第 2 節において自動車ボディ系製品の仕様に関する一般的な問題について述べる。今回，アイシン精機は自動車ボディ系製品へのプロダクトラインの導入に取り組んだ。第 3 節ではその組織面での取り組み，第 4 節では技術面での取り組みについて報告し，第 5 節では導入の効果を評価する。最後に第 6 節において本稿を総括する。

2. 自動車ボディ系製品の仕様

アイシン精機は電動スライドドア，サンルーフ，メモリシートなど，自動車のさまざまなボディ系製品を開発してきた。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。アイシン精機は，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製品群を製造している。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圧が極めて高い。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わけにはいかず，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係る仕様は複雑な条件判断を伴うものとなる。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うわけにはいかず，その目的のためにはモータの回転数がカウントされる。機構が期待した状態になったことを知らせるイベントが期待された時間内に来ない状況はさまざまに考えられる。機構の状態を見るセンサが故障しているのかもしれないし，機構が人や物を挟んでいるのかもしれない。我々は他の条件と組み合わせて真の状況を知るか，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しなければならない。

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る性質上，製品仕様の最終決定権は社にはない。メーカによって安全に係る設計思想が異なることもあって，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なってくる。これも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。

3. プロダクトラインの組織的導入

以上に述べてきたボディ系製品の仕様記述の問題を踏まえたうえで，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辿れるようにする。
- 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，物理レベルに近い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，そのため異なるバリアント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役員と部長の合意の下，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 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。

4. プロダクトラインの技術的導入

4.1 プロダクトライン開発プロセス確立以前

社はこれまでクローン＆オウンによるバリアント開発を行ってきたため，既有のドキュメントとコードは複雑化しており，短い期間で製品の仕様，構造，振舞いを正確に理解することは困難になっていた。最も近いバリアントから新しいバリアントを開発するときはいつも，ドキュメントのレビュー，特に影響分析に相当の時間が費やされていた。多くの先行事例 [3], [4], [5] がそうしているように，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 [6] 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。このことはフィーチャ分析において既存製品の内部の振舞いを詳細に調べなければならないことを意味する。そこで，開発支援部門はフィー

チャモデリングと並行して構造化分析を実施し，ソフトウェア内部の振舞いをデータフロー図として表現することにした。

対象製品のこれまでの開発に携わっていない技術者がこれらの分析を実施し，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，すなわちひと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ため，再度，モデリングのあり方を見直すこととなった。

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，もっと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。
- システムの振る舞いは，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，バリアント間の可変性は条件とアクションに多く含まれる。

4.2 ドメインエンジニアリングのプロセス

前小節に述べた観察結果を踏まえて，図 3 に示すプロセスで，ボディ系部品の既存製品のドキュメントからのコア資産開発，すなわちドメインエンジニアリングを行った。この図はあくまでプロセス中の各工程間の情報の流れを示すものであり，工程の順序を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が，成果物の諸要素の命名や抽象度の調整等で並行的に行っている部分は少なくない。

このドメインエンジニアリングプロセスにおけるファーストクラスモデルは，バリアントのちがいを表現し，また仕様項目の抽象度に基づく階層化と関心事の分離を図るフィーチャモデルである。そこで既存製品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，ならびに製品開発部門の知見者の製品知識に基づいてフィーチャモデルを構築する。

一方で，フィーチャモデリングでは既存製品群の内部の振舞いも調べる必要があり，それらは構造化分析によって

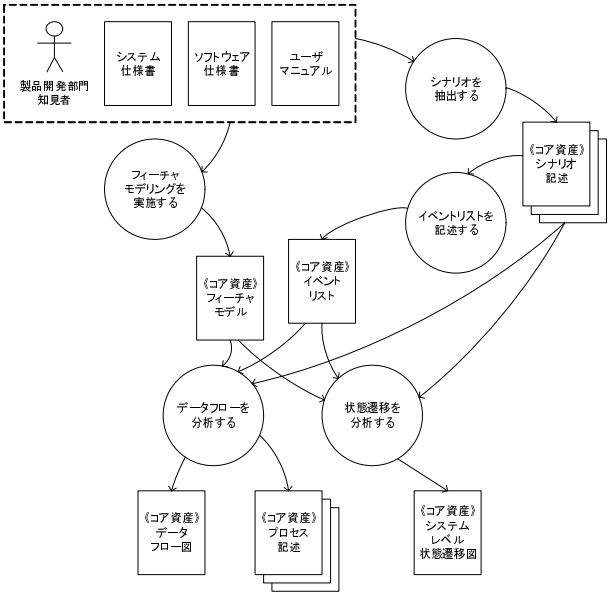

図 3 ドメインエンジニアリングのプロセス

行う。

前小節で述べたように，社で開発している多くのボディ系製品は，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。システムの振る舞いは，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。そこでこれら 4 状態の間の状態遷移を記述するシステムレベルの状態遷移図を作成する。

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，さらにそれらをひとつのイベントリストとして俯瞰的にまとめ上げる。各機能の振舞いは階層化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スティミュラスとしてシステムに与えられ，システムからレスポンスが得られるまでの，システム内部での一連のデータ変換を階層化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。各機能の振舞いは基本 4 状態ごとに階層データフロー図とその図中の各プロセスの仕様が作成される。これらをひと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，制御プロセスについては各機能基本 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が，すでに作成されたフィーチャモデルを鑑とすることによってそれを実現した。

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。

4.3 フィーチャ分析

フィーチャ分析を実施し，既存製品の仕様の全体像を把握し，そこから要求を抽出した。そうして得られたフィーチャモデルを改善し，製品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も行った。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。見ての通り，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と登場する。また，すでに述べたように，既存のドキュメントでは，派生開発の繰り返しの末，仕様項目のちが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。

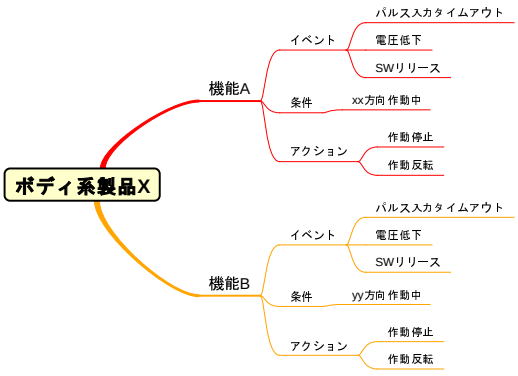

図 4 構造改善前のフィーチャモデル

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，全車種の全仕様をカバーするまで繰り返し行った。その結果，フィーチャモデルは図 5 のように改善された。

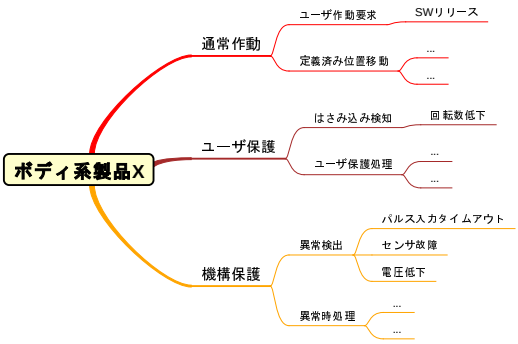

図 5 構造改善後のフィーチャモデル

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，これまでのドキュメントには十分に記述されていなかった事柄に相当し，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発掘され，仕様との対応が明るみになっていった。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。特に，イベントと条件の理解容易性は著しい向上が見られた。

目的レベルのフィーチャにはバリアント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」はすべてのバリアントに共通に現れる概念であり，バリアント間のちが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），それらの組み合わせから定まるアクションの定義に現れた。フィーチャモデルで明らかになった共通／可

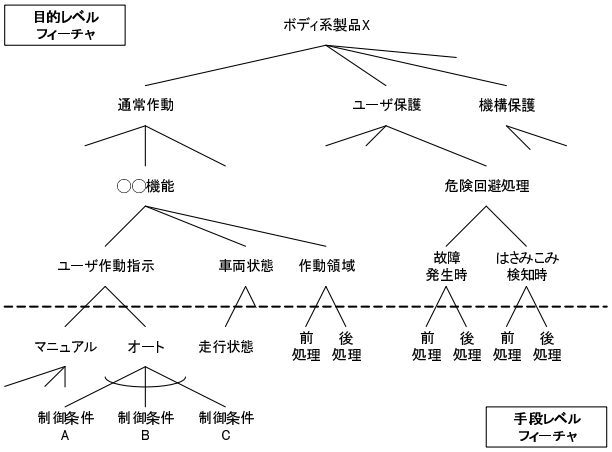

図 6 最終的なフィーチャモデル

変構造により，この後に実施するデータフロー分析，ひいてはプロダクトラインアーキテクチャの方針が定まった。

4.4 データフロー分析と状態遷移分析

ドメインエンジニアリングでは，フィーチャ分析と並行して構造化分析，すなわちデータフロー分析と状態遷移分析を実施した。当初，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，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。そこで，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採って，制御プロセスに入力されるデータの種類を低減することにした。

- 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留め，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／禁止指示など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など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，想定されるすべてのシナリオを，記述されたデータフローで実現できるかを検証した。
- 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並行実施の概要を示す。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かたちで，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。

5. 評価

5.1 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 ある巨大なものとなった。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。例えば，従来のフェールセーフに関する仕様記述では，異常が発生し得るいくつかの制御条件毎に，検出対象の故障とその後の処理が規定されていた。それらの組合わせに加え，さらに機構や OEM からのフェールセーフ要求のちがいを考慮する必要があり，膨大な条件の組み合わ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，このような組合わせが，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み合わせで，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。

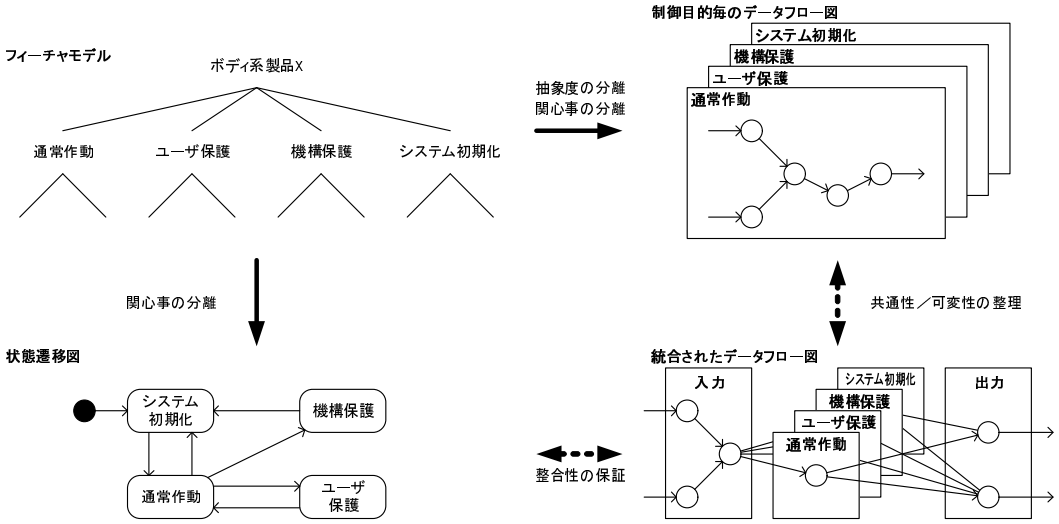

図 7　フィーチャ分析と構造化分析の並行実施

5.2　アーキテクチャ

フィーチャ分析と構造化分析の並行実施により，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。評価に用いた影響度は文献 [7] で提案されているものを用いている。この影響度は，単に依存モジュール数の平均をとるものではなく，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因る変更の生じやすさを加味する定義がなされている。社では毎年おおよそひとつのバリアントを出しているが，図 9 では，2012 年に出したバリアントの構造影響度を 100 として，各年に出された製品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 年のバリアント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 年以降のバリアントの構造影響度はそのときの 1/8 となっている。

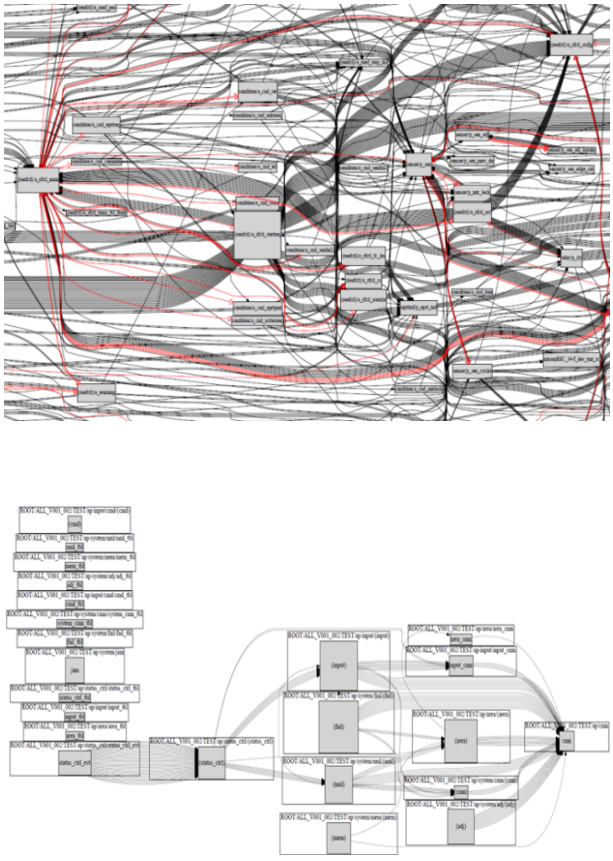

図 8　アーキテクチャの改善: 改善前（上）と改善後（下）

5.3　開発コスト

課題となっていたバリアント毎の開発工数についても，図 10 に示す通り，コア資産からのバリアント開発に要した工数実績値は，従来のクローン＆オウンでバリアントを開発した場合に要するであろう工数の見積もり値のおおよそ 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バリアントのものである。すなわち，プロダクトライン開発による工数実績値には，コア資産からのバリアント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言われるように [8]，我々の事例においても 3 つ目のバリアントにおいて，コア資産開発に投じた費用を回収することができた。

5.4　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

*2　機密上，図の画質を意図的に落としている。

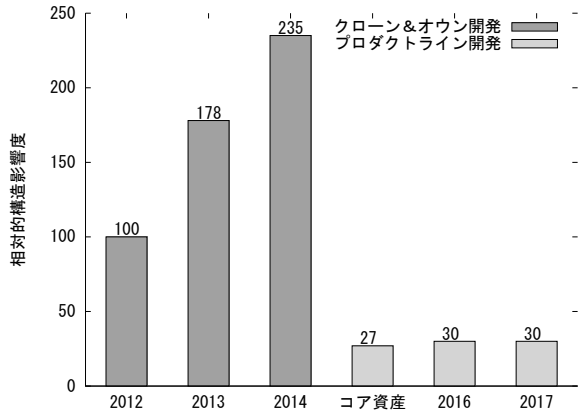

図 9 相対的構造影響度

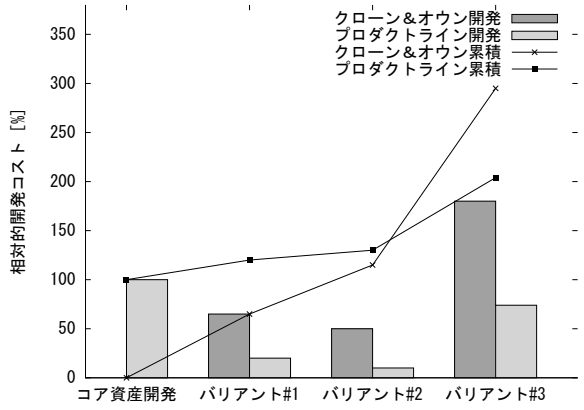

図 10 相対的開発コスト

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: これまではひと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採れるようになった。
- 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。

6. まとめ

本稿では，アイシン精機における，車載ボディー系製品のクローン＆オウン開発からプロダクトライン開発への移行事例を紹介した。導入においては，まずフィーチャ分析を実施し，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。フィーチャモデルに基づいて，構造化手法によるデータフローと状態遷移の解析も進め，プロダクトラインアーキテクチャを構築した。一連の活動により，対象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，バリアント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 つ目のバリアントをリリースするまでには回収できた。

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

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

[3] 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, Jan. 1998.

[4] K. Lee, K. C. Kang, W. Chae, and B. W. Choi, “Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse,” *Software: Practice and Experience*, Vol. 30, No. 9, pp. 1025–1046, June 2000.

[5] 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 年 10 月.

[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] 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.

[8] J. D. McGregor, L. M. Northrop, S. Jarrad, and K. Pohl, “Initiating Software Product Lines,” *IEEE Software*, Vol. 19, No. 4, pp. 24–27, July/Aug. 2002.