

技術

TECHNICAL GUIDE BOOK 2010

組込みソフトウェア開発のための
テクニカルガイドブック2010



株式会社エクスモーション

現場改善に役立つ開発技術

エクスモーションでは、開発現場の改善に役立つ技術を以下のように分類しています。それぞれに長所・短所があり、時には同じように効果があるというわけではありません。長所をまわして活用することで、きっと皆さんの開発現場の改善につながると思います。

きれいに作る！



構造化手法
オブジェクト志向

設計の見える化



Page 3-4.
ポトΔPツツ
から始める状態
モデリング

ADL
-UML・SysML

正しく作る！



MISRA-C
形式仕様記述

Page 9-12.
モデル検査の
使い方

機能安全
検証

Page 13-16.
機能安全における
障害解析と
要求管理の統合化

きちんと作る！



USDM
XDDP

Page 21-24.
XDDPによる
リファクタリングの
実践プロセス

作る



設計の動く化



Page 5-8.
Simulinkモデル
の品質を改善しよう

MDD
MATLAB/Simulink

次ページからは、これら技術の中から、選りすぐった技術についての記事が始まります。皆さんの開発の合間のちょっとした読み物として、ご活用ください。

部分の再利用



デザインパターン
アーキテクチャパターン
部品化
フレームワーク

全体を再利用



ソフトウェア
プロダクトライオン工学
(SPLE)

Page 25-28.
Ripple~
既存ソフトウェアを活用した
プロダクトライオン開発の実現

利用する



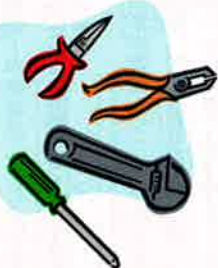
多品種
展開

経年劣化

直す



手直し



リファクタリング

Page 17-20.
工学的アプローチによる
リファクタリング

ボトムアップから始める状態モデリング

UMLで「振舞い」の側面を表現するチャートにはいろいろな種類がありますが、みなさんはスタートマシン図を使っていますか？私の経験からすると、「シーケンス図は使っているけど、スタートマシン図は使っていない」という方が多いようです。

なぜスタートマシン図が使わないのでしょうか？

多くの人が「シーケンス図で動くモデルが作れるから」ということを理由としてあげると思います。

しかし、シーケンス図は“ある”シナリオや状況を表現するためのチャートであり、抽象化されたモデルを説明するための、いわば補助的な位置付けにすぎません。それに対して、スタートマシン図は、クラス図の抽象化された振舞いを定義する唯一のチャートです。組込み機器は、いつどんなイベントが来ても仕様どおりの動作をすることを保証しなくてはなりません。そのため、並行性や応答性など、振舞いの側面はこのほか重要です。そのベースとなるのがスタートマシン図になるのです。

そんな重要なモデルであるにも関わらず、モデルが作れないのはなぜなのでしょう？

なかには「構造側面に着目してトップダウンでモデルを作ったが、クラスにふさわしい状態を見つけられない」という理由をあげる人もいるのではないのでしょうか。

ここでは、「構造の側面に着目してトップダウンから始める」という従来の常識をくつがえす「振舞いの側面に着目してボトムアップから始める」モデリングアプローチを、自動販売機のモデリングの「利用者がコインを投入してから商品を受け取るまで」を例に、紹介したいと思います。

本題に入る前に、トップダウンやボトムアップの前提となっているアーキテクチャのモデルをご紹介しておきましょう。表1は、動的な側面を取り扱うモデルの階層構造を示しています。Application levelがトップであり、Device levelがボトムとなります。

Step1.入力系Device levelのモデルを作る

動作レベル	内容
Application level	ユーザ定義の動作シーケンス Use Caseレベルの動作
Action level	シーケンス内の個別動作 戦略的動作
Behavior level	複数センサ入力を利用した実時間閉ループ制御、自律動作など
Device level	デバイス制御、保安動作 ハードウェアの抽象化

表1 動的な側面を取り扱うモデルの階層構造⁽¹⁾

「利用者がコインを投入してから商品を受け取るまで」に関係する入力系デバイスには、投入コインを検出するセンサがあります。投入コインセンサは、常に待機中で、コインが投入されたときに、コイン種類を検出します(図1)。

Step2.入力系Behavior levelのモデルを作る

Device levelのモデルから、抽象度を一段階上げましょう。Behavior levelは、Device levelで得られた複数の情報を活用します。入力系Device levelで得られた複数の「コイン検出」を使った振舞いとしては、コインの種類ごとの投入コイン数に關係する振舞いがあります。一般的に自動販売機では、コインの種類ごとに投入最大枚数を設定しています。そこで、コインの種類ごとに、何枚投入されたかを覚えておき、最大枚数に達した場合には、それ以上の種類のコインを受け付けられないような振舞いをする必要があります(図2)。

図2 コイン毎の受け付け

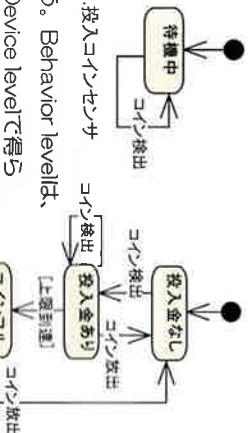


図1 投入コインセンサ

Step3.出力系Device levelのモデルを作る

「利用者がコインを投入してから商品を受け取るまで」に関わる出力系のデバイスは、商品ボタンについている「ランンプ」のみです。ランンプは単純に「点灯・消灯」の状態を持ちます(図3)。

Step4.出力系Behavior levelのモデルを作る

一段抽象度を上げて、Behavior levelで考えると、その「ランンプ」が「点灯・消灯」することの意味である「ランンプがついているボタンが押せるかどうか」という振舞いが考えられます。これは、「商品選択ボタンの振舞いと考えることができます。商品ボタン」には、「押せない」と「押せる」状態があり、「押せる」時にはランンプが点灯します(図4)。また、「商品ボタン」には、「ランンプ」以外に「スイッチ」が付いており、入力側の役割も果たします。

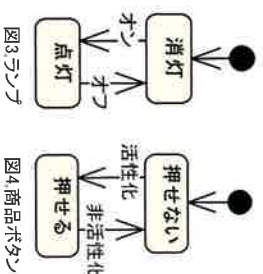


図3 ランンプ

「点灯・消灯」することの意味である「ランンプがついているボタンが押せるかどうか」という振舞いが考えられます。これは、「商品選択ボタンの振舞いと考えることができます。商品ボタン」には、「押せない」と「押せる」状態があり、「押せる」時にはランンプが点灯します(図4)。また、「商品ボタン」には、「ランンプ」以外に「スイッチ」が付いており、入力側の役割も果たします。

Step5.Action levelのモデルを作る

では、Behavior levelのモデルから、さらに一段抽象度を上げてAction levelの振舞いについて考えましょう。この部分は、入力系Behavior levelで得た情報を受け取って、出力系Behavior levelの情報を作り出す部分となります。その振舞いは、機能要求から採り出すことができます。

インポートされたコインの種類ごとの枚数を使って、利用者が購入のために投入しているお金の合計金額を割り出すことができます。この情報を使って、現在の投入金で購入可能な商品を利用者に教えます(押下可能なボタンを教える)。例えば、120円と150円の商品を取り扱っている自動販売機の場合、投入金が120円以上になったタイミンで120円の商品が選択可能となり、150円以上になったタイミンで150円の商品が選択可能となります(図5)。販売可能な状態になると、商品ボタンを押せるようになります。

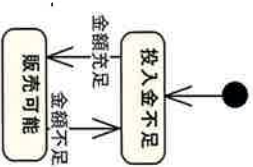


図5 商品銘柄ごとの販売可能状態

今回の範囲では、最上位のレベルであるApplication levelのものはありません。まとめ

このように、ボトムアップで振舞いに着目して情報を整理整頓していくと、それなりの状態を抽出できます。特に、Device /Behavior levelであれば、わりと簡単にできると思います。

図6に示している自動販売機のパッケージ構成では、商品販売以外のパッケージの振舞いを、このやり方で見つけることができます。しかし、上位レイヤのモデルも全てこのやり方の延長上で作れるか、というところ...なかなかそう簡単にはいきません。このあたりにつきましては、機会をあらためてじっくりと説明したいと思います。次回をお楽しみに。

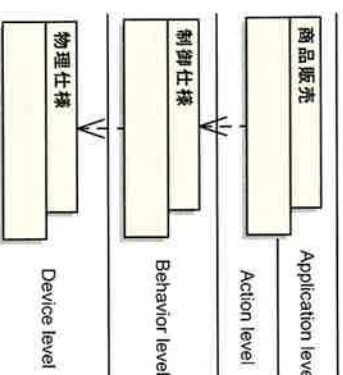


図6 自動販売機のパッケージ図とLevelとの対応
テクノロジカルガイド2009 P14より引用。
<http://www.exmotion.co.jp/laboratory/2009/05/2009.html>

UML/オブジェクト指向モデル作成支援

弊社では、お客様の状況に合わせたモデル作成支援を行っています。ご質問、ご要望がございましたら、こちらまで。 info@exmotion.co.jp

Simulinkモデルの品質を改善しよう！

Simulinkモデルを使った開発では、制御ロジックの再利用率が高く/バリエーションが多い製品の場合、ひとつのモデルを維持修正しながら開発が行われることが多くなります。しかし、このような開発が繰り返されると、徐々にSimulinkモデルの品質が劣化していき、開発に悪影響を及ぼすようになります。これを防ぐには、Simulinkモデルの品質を改善する活動を継続的に行うことが重要になってきます。

では、どうすればSimulinkモデルの品質を改善することができるのでしょうか？

品質の改善を行うには、まず、Simulinkモデルに混入した問題点を発見すること、そしてその問題に適切な対策を行うことが必要です。

ここでは、具体的に二つの改善手法をご紹介します。一つはメトリクスを用いることで品質を改善する方法、もう一つはモデルのクローンを検出することで品質を改善する方法です。

1. メトリクスによる品質改善

メトリクスとは、対象のものを様々な視点から定量的に計測した情報のことです。Simulinkモデルからも得ることができます。メトリクスにはSimulinkモデルの様々な特性が表れるため、これを評価することでモデルに問題があるかどうかを検討することができます。

ここでは、メトリクスを用いて品質を改善する方法のイメージをつかんでいただくために、二つの例を示しながら解説していきます。

例1) ブロック数から規模が適切でないサブシステムを検出し、改善する

この例では、サブシステムに含まれるブロック数を調べることで、規模の適切さを評価します。

サブシステムの規模が大きすぎる場合、そこには多くの処理が詰め込まれており、理解や修正がしにくくなっていることが考えられます。

図1のSimulinkモデルの各サブシステムに対し、ブロック数を計測した結果を示したのが図2です。このデータを見ると、Cというサブシステムのブロック数が他と比べて多い、つまり、規模が大きく、多くの処理を行っている可能性があることが分かります。

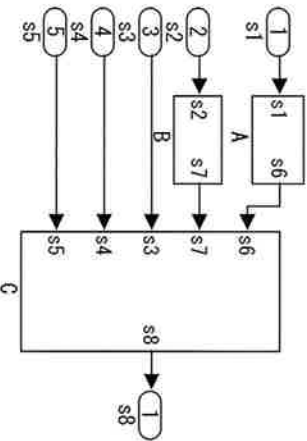


図1. 評価対象のSimulinkモデル

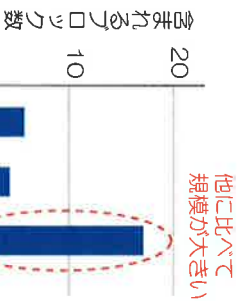


図2. サブシステムごとのブロック数

そこで、Cの内部処理がどうなっているか確認してみます（図3）。このモデルを解析すると、ある数式を計算する“計算処理A”、別の数式を計算する“計算処理B”、条件によってそれらの計算結果のどちらを採用するかを判定する“判定処理”という三つの処理を行っていることが分かりました。三つの処理が詰め込まれた結果、ブロック数が多くなっていたわけです。

それそれぞれの処理を別のサブシステムにする

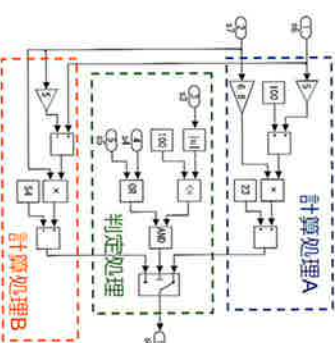


図3. Cの内部処理のモデル

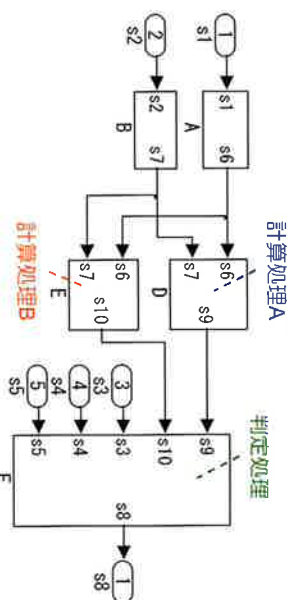


図4. 改善後のモデル

サブシステムCの問題点が分かったので、それを改善してみよう。この例では複数の処理がひとつのサブシステムに詰め込まれていることが問題なので、これらの処理を別のサブシステムとして分離します（図4）。

この改善によって各サブシステムの規模が適切になりました。そして、それぞれが単一の処理だけを行うようになり（凝集度が向上）、そのサブシステムで何の処理をしているかが明確になりました。

例2) 経路複雑度から処理が複雑なサブシステムを検出し、改善する

この例では、サブシステムの経路複雑度からそのサブシステムの処理の複雑さを評価します。経路複雑度とはSwitchブロックやIfブロックの数、Multipoint Switchブロックの入力数といった分岐の数を元に計測されるメトリクスで、分岐が多くなると処理が実行されるパスが多くなることから、処理の複雑さを測る指標として用いられます。

サブシステムの処理が複雑だと、その処理の内容が理解できなかったり、修正するにもどこから手をつけてよいかわからなかったりと、そのモデルを保守することがとても難しくなります。



図5. サブシステムごとの経路複雑度

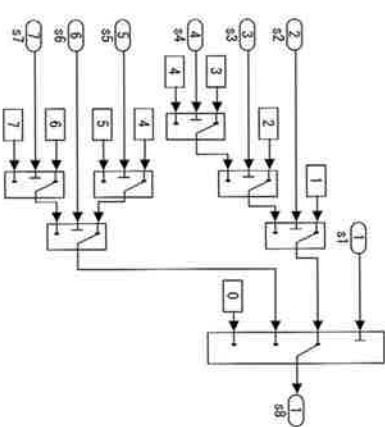


図6. Jの内部処理のモデル

ここで、あるSimulinkモデルの各サブシステムの経路複雑度を計測した結果を見てみましょう（図5）。このデータを見るとJというサブシステムの経路複雑度が高い値を示していることが分かります。そこで、Jの内部処理を見てみます（図6）。Multiport SwitchブロックやSwitchブロックが多く、どのような条件でどのような判定を行っているか分かりにくくなっています。

このように分岐が多く複雑な処理を改善するには、ある分岐のまとまりごとにサブシステムに分離することが効果的です。Jの場合、Multiport Switchブロックへの入力となっている部分をそれぞれサブシステムにしてみます。その結果、図7のようなモデルとなり、このサブシステムの処理の複雑さを軽減することができました。

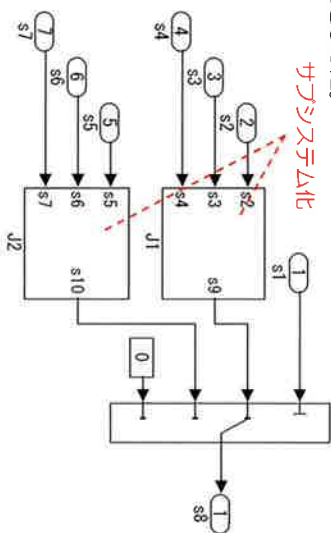


図7. 改善後のJのモデル

このようにサブシステムの処理の複雑さを下げると、分かりやすさが向上するだけでなく、動作の検証がしやすくなるというメリットがあります。

二つの例で示したように、メモリクスの用いることでSimulinkモデルの問題点を発見することが容易になり、改善を効率的に行えるようになります。

2. クローン検出による品質改善

Simulinkモデルにおけるクローンは、コピー&ペーストなどにより他の箇所にも同じブロックを組み合わせた同一の処理が存在することを言います。コピー&ペーストすることで開発効率を向上するように考えがちですが、モデルに不具合があった場合に、コピーしたモデルを全て修正しなければならず作業量が增大しますし、修正漏れの危険性もあります。そのためクローン化されたモデルを早期に発見して対策を行う必要があるわけです。

クローン化されたモデルを改善する例として、あるサブシステムK（図8）とサブシステムL（図9）の内部処理を見てみます。これらのモデルをよく見てみると、破線で囲った部分はブロックが同じ組み合わせで結ばれており、同じ処理を行っていることが分かります。この部分がクローンです。クローンの部分は入力信号とConstantブロックの値以外は同じですので、それらを入力に持つサブシステムを作成し、そこにこの処理を移動させてみましょう（図10）。このサブシステムを例えばライブラリブロックとして定義し、Kとしてそのブロックを利用するように変更すれば（図11）、仮にその処理を変更する場合でもライブラリを修正するだけで済みます。

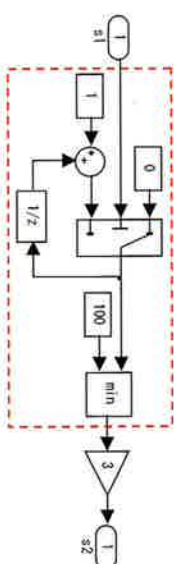


図8. Kの内部処理のモデル

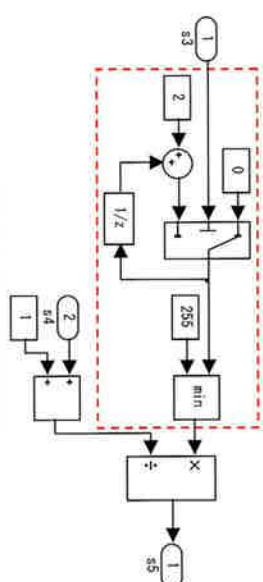


図9. Lの内部処理のモデル

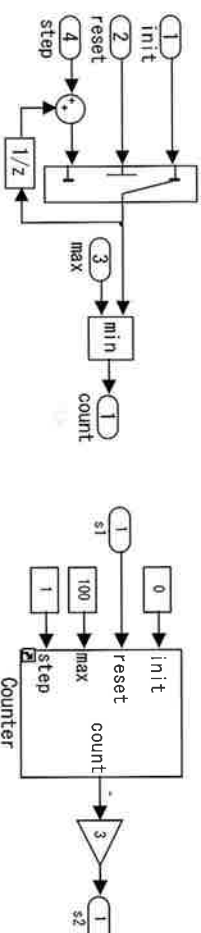


図10. 共通処理をライブラリ化

図11. ライブラリを使うようにKを変更（Lも同様）

このように、モデルのクローンを発見し、共通処理をライブラリ化することができればモデルの保守性が向上しますし、ライブラリが充実していくことで開発効率の向上も期待できます。また、ライブラリが様々なところで利用されることでその信頼性も向上します。しかし、全てのクローンを目標で見つけることは困難です。ツールを用いて自動的にクローンを検出することが現実的です。

以上のような手法を用いることでSimulinkモデルを改善することができます。また、現状ではモデルの品質に問題がない場合でも、メモリクスの測定やクローンの検出を定期的に行うことで、品質が劣化した時にすぐに気付くことができ、開発に悪影響を及ぼす前に対策することが出来ます。

まずはSimulinkモデルの品質の現状を知るために、メモリクスの測定とクローンの検出を実践することをお勧めします。

Simulinkモデル品質診断ツール

弊社では、Simulinkモデルのメモリクスの測定やクローンの検出を行える品質診断ツールを開発中です（年内発売予定）。このツールをお使いいただくと、この記事に記載したような品質改善も容易に行うことが可能です。

ご興味のある方は、ぜひ以下のメールアドレスまでお問い合わせください。

info@exmotion.co.jp

モデル検査の使い方

ソフトウェアには静的側面と動的側面があります。静的側面はモジュールやクラス、関数等で構成されデータ構造と手続き、役割といった視点で構築されます。一方、動的側面はスレッドや割り込みで構成され並列・並行性と時間制約、振る舞いの視点で構築されます。モデル検査技術は振る舞いをモデル化し、検証するための技術と言えます。

組み込みソフトウェアは、何らかの処理中でもイベントに反応しなくてはなりません。しかし、それらの振舞いを事前に精査することなく、実装されることが往々にしてあります。その結果、不具合が起これたり、性能を満たすことができなかったりなどの問題が起これります。モデル検査技術を上手に利用することで、作れこみの段階でそのような問題を事前になくすることが出来ます。

モデル検査本来の使い方は作ったモデルの性質を調べるプロパティ検証ですが、合成・分解・比較・変換を使うことでモデルを作る作業を直接支援できるようになります。一度振る舞いをモデル化してしまえば、要求工程から実装工程まで検証しながら振る舞いを数理的に扱うことができるだけでなく、最終的なタスクは様々で得ることができるようになります。



工程	工程における一般的な作業	モデル検査技術による支援	プロパティ検証
要求分析	シナリオ検証、環境モデル作成 動作要求定義 外界との関係定義 システム動作を定義する	個別動作から全体動作の合成 プログラム可能な記述へ変換 振る舞いの階層構造の発見 時間制約の取込	テストロジック* ライブラリ**
	静的構造との対応付け システム動作をオブジェクト毎 の動作に分割して内部動作を定義	オブジェクト動作の導出と検証 オブジェクト間の状態不整合の検出 不足するシナリオの検出 システム動作の合成 インタフェース検証	テストロジック* ライブラリ** 応答性検証
	オブジェクト指向モデリング 構造化 モデリング	システム動作のトップダウン分解 による内部動作を定義する	
アーキテクチャ設計	内部動作をタスクや割り込み ハンドラに割り当てる 優先度、動機構造を定義する	タスク分割と優先度付検証 不足する同期構造の検出 テストロジック、状態不整合の検出と 解消 テストライブラリ検証	テストロジック* ライブラリ** 応答性検証
タスク設計			

*テストロジック: 出口のない状態 **ライブラリ: 出口のないルーチン

動作モデルの作り方はツールに依存しますが、プロセス代数ベースのツールのTSA, PAT, FDR では動作式を書くことがモデル作成に相当します。動作式は、XXする→OOする→△△する...という形になります。動作の順番を含んでいるものであれば何でも動作モデルにすることが出来ます。

ソフトウェア開発の各工程では、さまざまな振る舞いの表現がありますが、一般的に良く使われる「文章」「フローチャート」「UMLの状態マシン図」「UMLのシーケンス図」からは、次のような動作モデルが作られます。そして、動作モデルにすることで合成・分解が可能になります。合成の際に矛盾があればテストロジック、分解の際に無理があればインタフェース定義失敗が発生します。既にUML等の設計資産がある場合にはツールを使って動作モデルに変換することも出来ます。動作モデルにすることでシーケンス図と状態図の内容に矛盾がないかの検証ができるようになります。

振る舞いの表現	動作モデル
文章 aする...bする cする...dする	$S1=(a \rightarrow b \rightarrow \text{END}),$ $S2=(c \rightarrow d \rightarrow \text{END}).$ 合成できるので一文ずつのモデル化で良い
フローチャート 	$S1=(p1 \rightarrow c \rightarrow C),$ $C=(p2 \rightarrow \text{END} p3 \rightarrow p2 \rightarrow \text{END}).$ 「または」演算子
UMLステートマシン図 	$S1 = (\text{init} \rightarrow S1),$ $S1 = (\text{act1} \rightarrow S2 \mid \text{act3} \rightarrow S1),$ $S2 = (\text{act4} \rightarrow S1 \mid \text{act2} \rightarrow S2).$ S1式はS1状態からact1を実行してS2に移る。またはact3を実行してS1に戻る。という動作を表している。
UMLシーケンス図 	Aは、aを送信する。次にbを受信する。 (b)という動作を表している。 $A = (a \rightarrow b \rightarrow \text{END}),$ $B = (a \rightarrow c \rightarrow b \rightarrow \text{END}),$ $C = (c \rightarrow \text{END}).$ 合成演算子 $Ma = (a \rightarrow a \rightarrow \text{END}),$ $Mc = (c \rightarrow c \rightarrow \text{END}),$ $Mb = (b \rightarrow b \rightarrow \text{END}),$ $\parallel \text{Seq1} = (A \parallel B \parallel C \parallel Ma \parallel Mc \parallel Mb).$

要求工程の例としてジュースの自動販売機の動作モデルを作成して要求の実現可能性を検証してみます。作成した動作モデルは動作仕様として設計工程で使用します。まず、コインを1種類として自販機に関わる外部動作と制約条件を調べます。

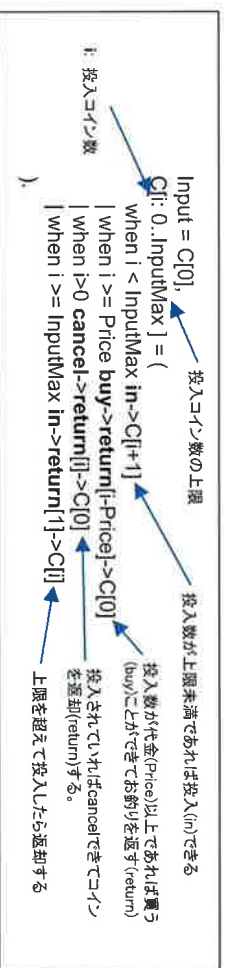
動作	動作名	動作内容・制約条件	備考
コインを入れる	in	入れたコインの数を数える 入れられるコインの数には上限がある 入れた金額が代金以上であれば買うことができる	上限を超えたら返却する
ジュースを買う	buy	代金以上のお金は返却する 代金は金庫に保管する 金庫には容量がある 在庫がなければ買えない	保管できれば買えない
買うのをやめる	cancel	入れた金額のコインを返却する	

これらの動作制約を満たすためには、入れたコインの数、金庫の中のコインの数、在庫数が必要なので、この三つを扱う変数(Input, Safe, Stock)を導入します。

変数(動作式)	管理する状態	関連する外部動作	追加した動作
Input	入れたコインの数	in, buy, cancel	return
Safe	金庫のコインの数	buy(saveとして)	save/out
Stock	在庫数	buy	なし

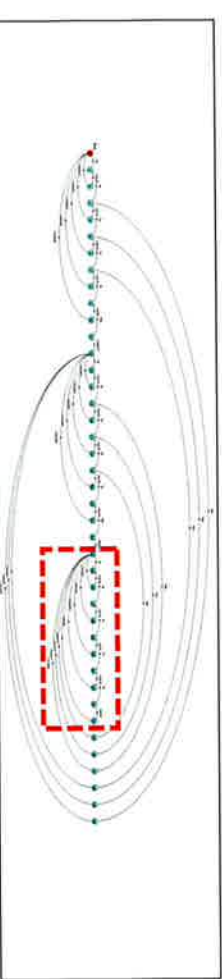
変数に影響を与える動作の関係を記述すると動作式になります。この過程で動作を完結するために必要な動作が明確になるので追加します。組み込みシステムはバッチ的な処理をするのではなく動き続ける特性があるため、初期状態に復帰することで動作が完結します。復帰するためには逆動作等が必要になります。上記の表では、in動作の逆動作としてreturnを追加しました。またbuy動作はSafeにとっては金庫に保管するsave動作であり、その逆動作としてoutを追加しました。

Input動作式を以下に示します。ここでCはInputのローカル状態でC[i]によってi個のコインが投入された状態であることを表します。動作式は動作制約を一つずつ表現することで作成します。たとえば、投入枚数が上限(InputMax)未満の時はin動作をすることでC[i+1]状態からC[i+1]状態に遷移します。C[i+1]は取り得る状態、すなわち変数値の範囲を定義しています。投入枚数がジュースの代金(PPrice)以上であればbuy動作をすることができて、その際には余ったコインを返却してC[i]状態に遷移します。



この様に投入コイン数に影響を与える動作をまとめてInput動作式を作成します。この際に、動作間に矛盾がある場合にはコンパイルする際にエラーとなります。エラーが無ければその変数に関する状態やシンクが生成されます。Safe, Stockについても同様にして動作式を作成します。完成したら三つを合成して自販機全体の動作モデルを生成します。合成の結果次のような状態やシンクが生成されシステムとして動作可能であることが分かります。また、動作モデルが完成したのでプロバタイ検証をすることが可能になります。

プロバタイ検証の結果、全体動作モデルにはinとcancelしかできない無限ループが存在することが分かりました。それは下図の赤枠の部分ですが、このループは在庫切れでbuyが実行できないためにinとcancelのみの出口の無いループです。この状況は、現実の世界でも起こり得る状況なのでモデル上の問題ではありません。メンテナンス関係の動作モデルを作成することで解消されます。



工学的アプローチによるリファクタリング

派生開発や保守開発が中心の組み込みソフトウェアは、度重なる仕様変更・機能追加により、その品質が劣化しがちです。ソフトウェアの品質劣化は、放置すると加速度的に進行していきます。瞬間に大規模な再構築を余儀なくされるような破綻した状態に陥ってしまいます。ソフトウェアの品質劣化を防止し、その寿命をのばすためにはソフトウェアを定期的にメンテナンスすることが必要です。

リファクタリングは、ソフトウェアの定期的なメンテナンスに適した、手軽に取り組むことができる品質改善の手段ですが、目についたところからアホボックに手をつけていても品質向上の効果はあからない、あるいは、効果があがってもそれはすぐにサチってしまいます。品質管理の分野では「勘・経験・度胸 (KKD)」だけに頼らず、事実に基づいて行動することが重要であるといわれていますが、これはソフトウェア品質改善においても同様です。

限られた納期とコストの中で、著実に効果をあげながら継続的に品質改善を行うためには、客観的な事実に基づいて問題箇所と要因を発見し、効果的な打ち手を講じることが必要です。

弊社では、これを行うための手法として、ソースコードのリバースエンジニアリングとメトリクス分析を使ったリファクタリングの工学的アプローチを推奨しています。以降では、そのステップを追いつながら、内容を紹介していきます。

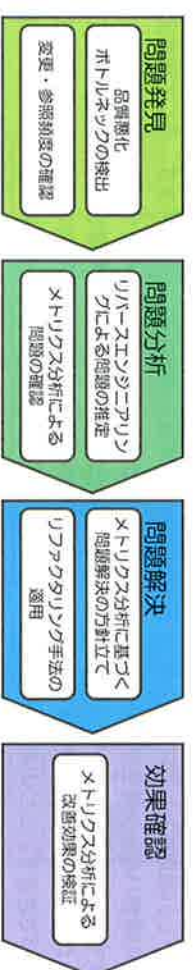


図1：工学的アプローチ

ステップ1. 問題発見

リファクタリングによる品質改善を始めるにあたって最初に行うべきことは、その対象を適切に定めることです。対象とするソフトウェアの「どこが」・「どのように」悪いのかを客観的に評価して改善対象を選定します。

バレーの法則 (20:80の法則) という経験則がありますが、ソフトウェアの場合も、全体で見ると少数の要因・要素によって品質問題は引き起こされています。そのような要因・要素を特定するには、不具合の発生件数や保守工数との相関を調べるのが効果的ですが、十分にデータを持ち合わせていない場合には、ソースコードを解析して得られるメトリクスを用いても推定することができます。

図2はあるソフトウェアについて関数の経路複雑度をヒストグラムで示したものです。基準値10を超える複雑な関数は全体の20%です。処理の複雑さを低減したいのであれば、これらの関数をリファクタリングするのが効果的であることがわかります。

また、リファクタリングの対象を定める場合には、その変更・参照頻度を確認しておくことも大切です。いくら複雑で保守性が悪くても、ほとんど修正する機会がない箇所はリファクタリングを行う意味はありません。忙しい開発現場では、だれも読まないコードをキレイにしている時間とお金の余裕はありません。

ステップ2. 問題分析

リファクタリングの対象が定まったら、次はその対象をより詳細に分析します。この過程ではリバースエンジニアリングやメトリクス測定によって、ソフトウェアの『形や重さ』を明らかにすることが有効です。図3はC++で書かれたソフトウェアの一部をリバースエンジニアリングして作成したクラス図、図4、図5はそれらのクラスに関するWMC、LCOMというメトリクスを測ったものです。

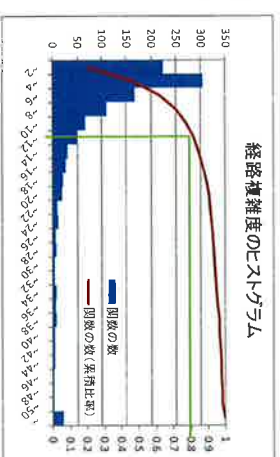
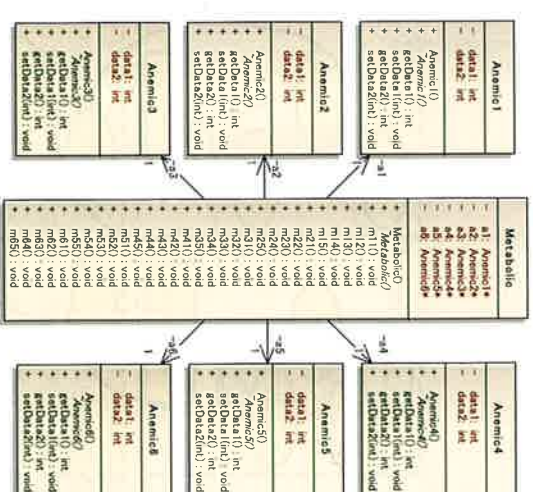


図2：経路複雑度のヒストグラム



また、LOOMというメトリックは、クラス内のメンバ変数の関連性の強さにより凝集性の欠如を指標化したものです(図6)が、“巨大な”クラスはこの値が大きく(つまり凝集度が低く)分割の余地があることがわかります。

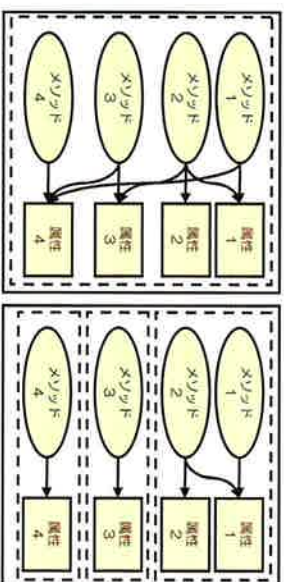


図6：LOOMの定義

以上の分析から、このソフトウェアの設計上の問題は次のようなものと言えるそうです。

「クラスの分割が不適切なため、凝集度が低く、規模が肥大化したクラスに複雑さが集中している。その結果、クラスの責務と関係によって機能を把握しづらく保守が困難になっている。」

ここまで分析できれば、「巨大な」クラスを分割して集中した責務を分散させる」というリファクタリングの方針を立てることが出来ます。

ステップ3. 問題解決

“巨大な”クラスはどのように分割すればよいのでしょうか。分割が悪いと、クラス内の複雑さは減少させることができて、副作用としてクラス間の関係の複雑さを生じさせてしまいます。したがって、クラスの分割は、分割されたクラスの凝集度が高く、クラス間の結合度が小さくなるようにしなければなりません。

ここでは、再びLOOMの考え方をもとにクラスの分割を検討してみます。データを中心に関連性の強いメンバ変数を集め、メンバ変数間の呼び出し関係が疎な部分を境界にしてクラスを分割すれば、高凝集・疎結合なクラス構造を実現できるはずです。

“巨大な”クラスのメンバ変数と属性のコールグラフ(図7)を調べると、関連するクラスごとにアクセスするメンバ変数が限られており、これらのメンバ変数を関連するクラスに移していくことで集中していた責務を分散させることがわかります。

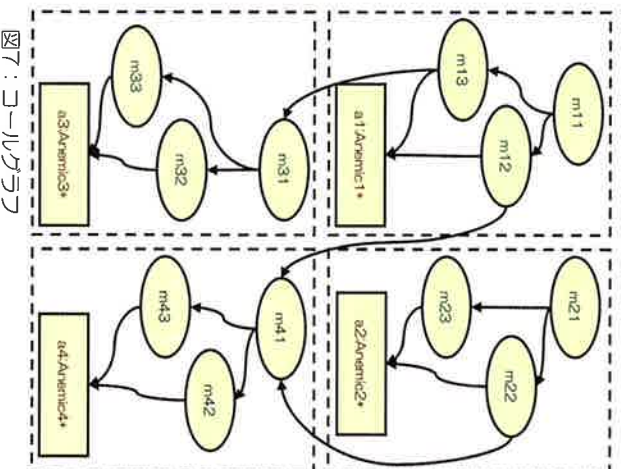


図7：コールグラフ

ここまで方針が定まれば、実際にソフトウェアの構造やメンバ変数の仕様を設計することができそれにしたがって、コードを修正していくことができます。コードを修正していくステップは、リファクタリング手法として書籍などで広く紹介されていますので、ここでは詳しく触れませんが、この例の場合「メンバ変数の移動」という手法を用いて簡単にリファクタリングができそうです。

ステップ4. 効果確認

修正後のテストを含めてリファクタリングが完了したら、その効果を確認します。問題を発見するときに使ったリバースエンジニアリングやメトリクスを使って、設計上の問題が解消できているか、新たな問題(副作用)を生じさせていないかを確認します。

リファクタリング後のソースコードに対応するクラス図とメトリクス(WMC・LOOM)を図8・図9・図10に示しました。凝集度が欠如した肥大化した巨大なクラスのメンバ変数が分散することによって、保守性が高まったことが確認できると思います。

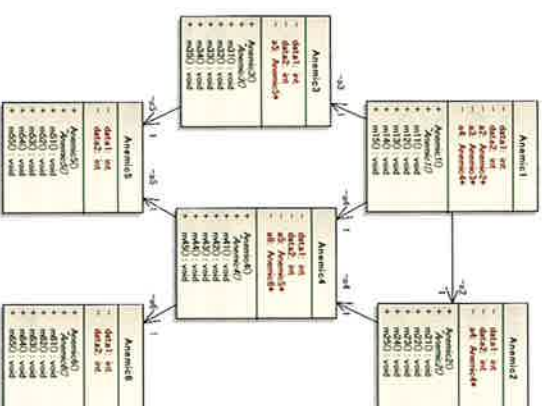


図8：クラス図

以上、リバースエンジニアリングやメトリクスを使って、客観的な事実に基づいてリファクタリングを進める手法について紹介してきました。図式表現や数値を使って、問題を推測したり、その裏付けをとりながら進めるのが工学的アプローチです。品質改善を行う上で、勘や経験は大切ですが、それのみに頼ることなく事実に基づいたアプローチをとることで、より効果的で着実な改善を実現できると思います。

コード品質診断ツール [exAuto]

『exAuto』はソースコードのリバースエンジニアリングやメトリクスの分析によりソフトウェアの品質診断を実現するツールです。『exAuto』を使うことで、ここで説明したリファクタリングをより容易に実践することが出来ます。ご興味のある方は以下をご覧ください。
<http://www.exmotion.co.jp/service/tool-exauto.html>

また、弊社では、ソースコード品質診断サービスも行っています。経験と技術の豊富なコンサルタントが課題の抽出や評価、アドバイスをまで行うサービスです。ご興味のある方は以下までお問い合わせください。
info@exmotion.co.jp

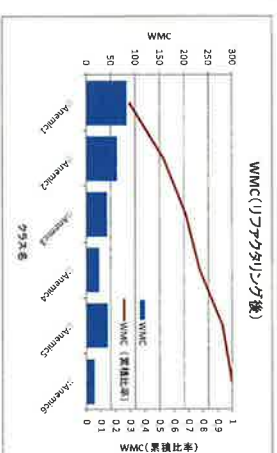


図9：WMC

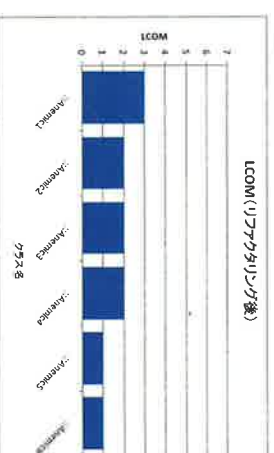


図10：LOOM

「リファクタリング」という言葉が世の中に出始めてから10年が経ち、さまざまな状況でリファクタリングという言葉が使われています。一般的な「リファクタリング」は、常日頃からプログラムを整理し、仕様変更にも対応できるプログラムにするという行為です。リファクタリングの手順は、まず既存コードの動作を保証するテストを作成し、その後コードの修正を行います。開発者が普段から修正しているコードにまた手を入れるという感覚であるため、「気軽」に日常的に行うプロセスです。

一方、実際の開発現場で良くある状況として、開発者が一から書いたコードだけでなく、多くの人が引き継ぎ、修正を重ねたコードもリファクタリング対象になる場合があります。また、構造が複雑になったため、複数の関数やクラスを修正する場合もあります。そのような場合も「テストを作成してからコードを整理する」で進めて安全でしょうか。筆者は、このようなリファクタリングで次のような問題が発生するのを、しばしば目にしてきました。

コードの理解不足による予期せぬミス

リファクタリングの対象となるコードは解析性が悪いことが多いのですが、それが他人によって書かれたコードである場合には、なおさら理解が困難です。過去の変更要求や修正依頼に対応するために、さまざまな試行錯誤が繰り返されたコードの場合には、いくら内容を理解して、慎重にリファクタリングを行っても開発者個人の目には限界があり、思わぬミスが生じてしまう場合があります。

影響範囲の確認不足による予期せぬミス

リファクタリングで行われるテストは、通常ユニットテスト（関数やクラスの単体テスト）ですが、複数の関数・クラスを修正するようなリファクタリングになると、単体テストではうまくいきません。機能レベルのテスト（機能テスト）を行い、関数間の呼び出し関係が正しく動作するかを検証する必要があります。複数の関数やクラスをリファクタリングしたときに、修正した全ての関数を通して機能テストが実行できていれば問題はありますが、機能テストでは、あらゆるテストケースを事前に用意するには大変な労力がかかるため、想定されるテストケースのみとなることが多いのが現実です。その結果、開発者個人のテストでは発見できない不具合が入り込んだまま、ソフトウェアが統合され、統合後の実機でのテストで「他の関数への影響」や「予期せぬタイミソングエラーの発生」などに気づくこととなります。その結果、トライアンプトエラーによる非効率なコード修正が行われます。

このような問題は、一般的なリファクタリングの「テストを作成してからコードを整理する」というプロセスでは十分ではありません。経年劣化したコードやある程度の規模のリファクタリングは、「コードの整理」ではなく、「コードの変更」と捉え、変更に含まれた開発プロセスが必要となります。変更に含まれた開発プロセスとは、成果物を定義し、プロジェクトチームでそれをレビューすることです。これにより、開発者個人では発見しにくい仕様、設計・テストのミスに早期に気付くことができたり、複数の開発者間での認識の齟齬を埋めることができます。結果として、作業の混乱や不具合の混入を防ぐことができ、変更を安全かつ効率的に進められるようになります。

このように、一般的なリファクタリングの「気軽」に行うプロセスに対し、プロジェクトチームで行う変更開発には「厳格」なプロセスが必要といえます。

チーム開発におけるリファクタリングのプロセス

変更による予期せぬミスを効率よく防ぐには、開発者個人がコードを変更する前に、次の3つについてプロジェクトチームでコンセンサスをとりながら進めることが必要です。

- 概要レベルの変更内容が仕様化され、それが妥当であること
 - 仕様の段階で、他のコードへの影響が追跡できること
 - 概要レベルの仕様に対し、どのように設計やコードを変更するかが示され、それが妥当であること
- 変更内容が仕様化された状態で、変更の影響の追跡と変更設計をレビューで確認することにより、変更によるミスを防ぐことができます。これらの成果物を確認した後にはコードを変更することで、テストの段階では仕様通りの動作することをチェックするだけになります。

しかし、ソフトウェアを新規に開発する場合のプロセス・成果物の定義が世に多く存在するのにに対し、ソフトウェアを変更するときのプロセス・成果物はあまり見かけません。その中で株式会社ソラテムクリエイツの清水吉男氏が提案された「XDDP (eXtreme Derivative Development Process)」は派生開発のためのプロセスであり、リファクタリングのプロセスを設計するときのヒントになります。リファクタリングはソフトウェアの機能こそ変えないものの、その設計を変更する作業だからです。

XDDPに関する詳細は、清水氏の書籍^[1]を読んでいただくこととして、本章では「XDDPの概要」と「リファクタリングへのXDDPの適用」について紹介します。

XDDPの概要

XDDPは、派生開発を行う場合に必要なプロセスと成果物の連鎖で構成されていますが、その特徴のひとつに「変更要求仕様書」・「TMT (Traceability Matrix)」・「変更設計書」という3点セットの成果物（図 1）があります。変更する箇所の情報を3点セットの成果物に記述することで、各担当者が予定している変更内容や変更箇所、具体的な変更方法をお互いに知ることができます。これを使って、コーディング前に仕様と設計の確認作業を徹底することで、もし、ミスなどによる手戻りのムダを撲滅します。



図 1 : XDDPの3点セット

書籍[1] 「『派生開発』を成功させるプロセス改善の技術と強要」（清水吉男著、技術評論社刊）

RIPPLE ～ 既存ソフトウェアを活用したソフトウェア開発の実現

皆様はソフトウェアプラットフォーム (SPL) についてどのようなイメージをお持ちですか？

我々は、お客様から以下のようなご意見をよく伺います。

- ・ SPL に取り組みたいが、ハードルが高そう
- ・ コア資産が重要なのはわかるが、ソフトウェアの再構築をやっている暇はない
- ・ 組織やプロセスまで変えるのは、リスクが大きすぎる

確かに、SPL は総合的な取り組みを必要とするため、ハードルが高く感じますし、その全てを一度に実践することは極めて困難です。しかし、皆様の企業・組織にとって重要な部分から、段階的に進めていくこともできるのです。特に近年では、既存のソフトウェアから再利用資産化できそうなものを抽出し、製品開発時に新たな要件等を加えて段階的に資産化していく方法での成功事例が多く報告されています。

弊社では、この手法を更に詳細化した“RIPPLEアプローチ”を推奨しています。本アプローチでは、以下の3つのステップを経て、既存のソフトウェア開発をPL開発へと移行させていただきます。

1. 既存資産の統合による、資産の一元化 (Rapid Integration)
 2. 一元化した資産の可変性分析と整理による、製品導出 (Production) の仕組み作り
- RIPPLEアプローチの作業フローは、以下の図で表わされます。

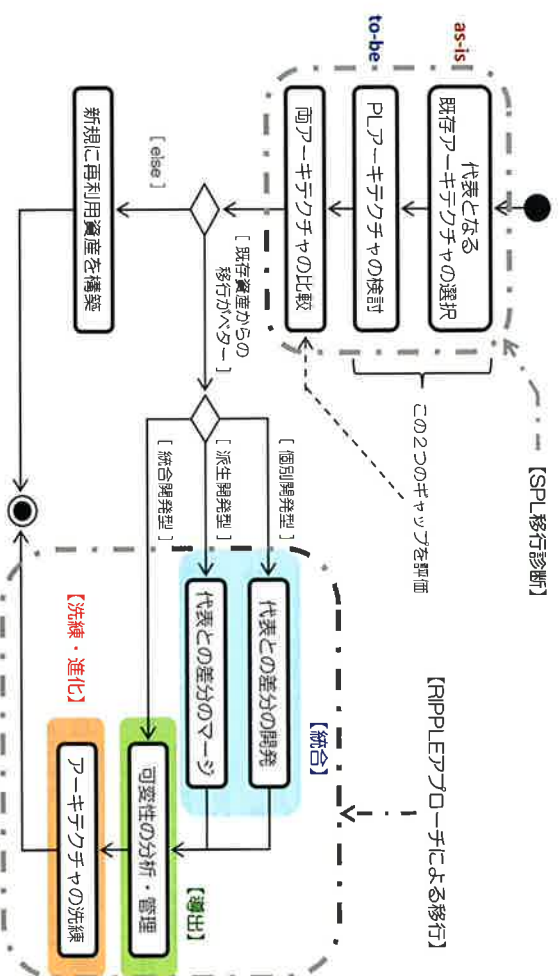


図1：RIPPLEアプローチ手順

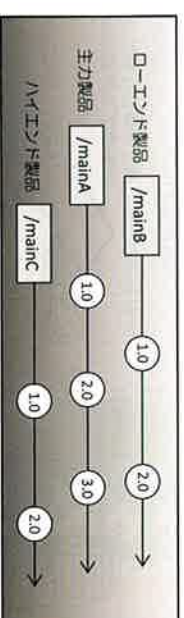
ステップ1：資産の統合 (Rapid Integration)

最初のステップでは、既存資産を1つに統合していきます。まず、皆様が現在開発している製品ファミリの資産（主としてソースコード）が、どのように構成管理されているかを把握することから始めます。具体的には、以下の3つのパターン（およびその組み合わせ）のいずれに該当するかを調べます。

【個別開発】

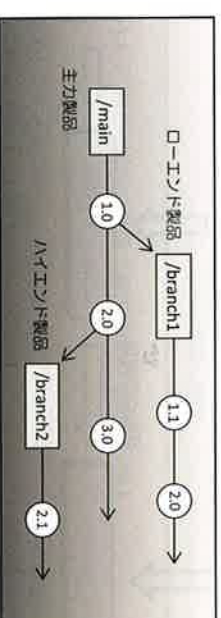
- ・ 同系列の製品を、部署（事業部）毎に個別に開発している

- ・ 製品間での流用は行われていない



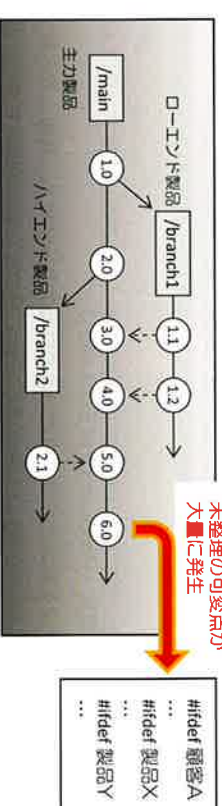
【派生開発】

- ・ 同系列の新製品の開発時に、既存製品のコードを複製して、追加＆変更しながら開発している
- ・ 複製したコードは独自に進化し、既存のコードと統合されることはない



【統合開発】

- ・ 同系列の製品に対して、共通のコードリポジトリが存在している
- ・ 新製品を開発する際、リポジトリからコードを複製して、追加＆変更しながら開発している
- ・ 複製したコードでの追加＆変更はリポジトリへと反映されるが、製品間の差異を分析・管理するまでには至っていない



個別開発 or 派生開発の場合は、このステップを通じて1つの資産へと変換していきます。これにより、統合開発と同じ状態へ到達することになります。統合開発の場合は、本ステップはスキップします。

ステップ2：製品導出の仕組み作り (Production)

次に、統合された資産の中に存在する可変点（製品毎に異なる点）について、それらが生じる要因（可変性）を特定していきます。通常、可変性が整理されていない状態では、本来同じ要因であるにも関わらず、異なるものとして実装されていることがほとんどです。これらを整理することで、可変性モデル（フーチャモデル等）が作成できます。

可変性の整理ができたなら、資産中の可変点をこの可変性に置き換えていきます。これにより、製品開発時には可変性を決定（解決）していくことで、自動的に再利用資産から製品資産を導出することができるようになります。

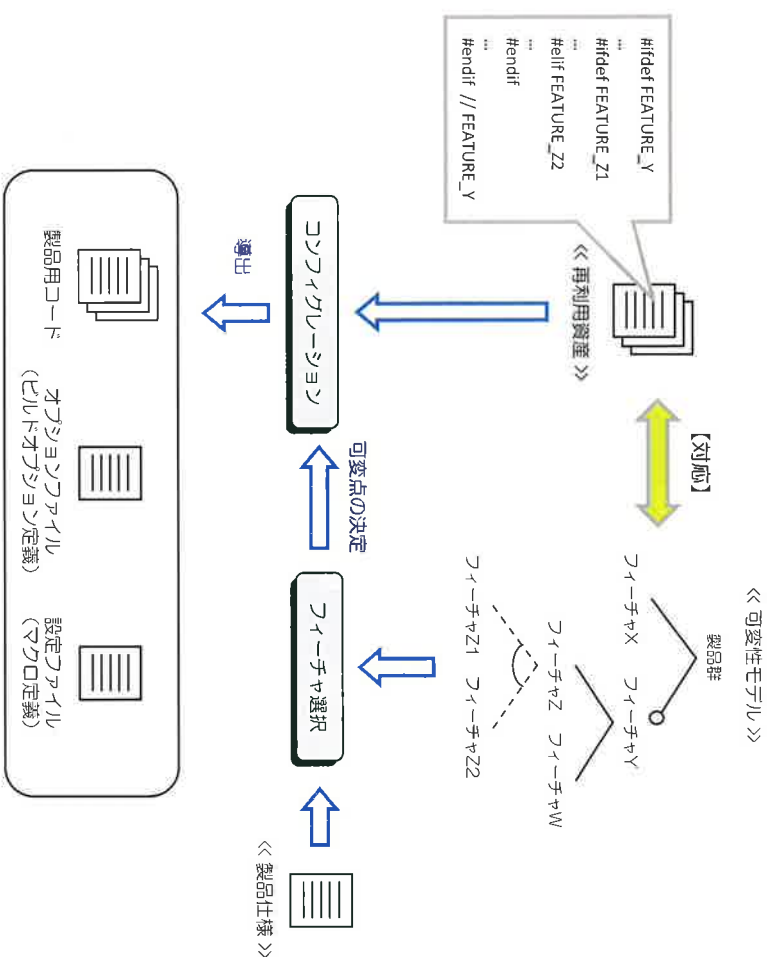


図2：可変点の決定による製品導出

ステップ3：資産の洗練・進化 (Product Line Evolution)

PL開発は、一度再利用資産を揃えたら終わりというわけではありません。対象とする製品ファミリーの寿命が尽きるまで、開発は延々と続いていくため、これに合わせて再利用資産も進化・洗練していかなければなりません。これが、本アプローチの最後のステップに相当します。

製品開発時には、新規要件への対応や仕様変更など、様々な要求が発生します。これら全てに対し、製品開発前に再利用資産として整備できる場合もあれば、工数や納期的に困難な場合もあります。

後者の場合は、一度製品として開発し、その後に再利用資産へフードバックしながら、1つの統合された資産を維持管理していくことになります。これにより、当該ファミリーに属する全ての製品は、1つの資産から導出可能な状態を維持していきます。

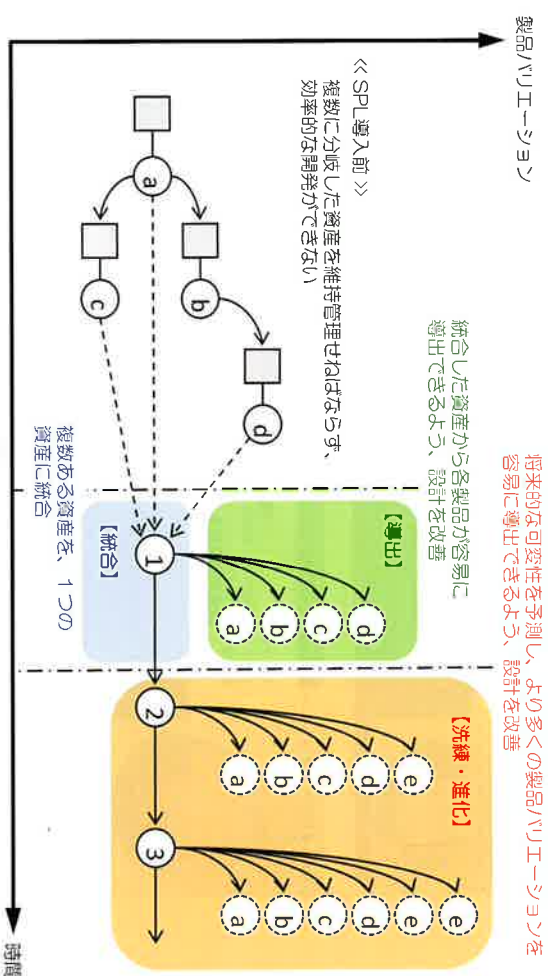


図3：RIPPLEアプローチの全体像（派生開発からの移行例）

この3つのステップを通じて、既存のソフトウェアをベースに当該製品ファミリーのソフトウェアアーキテクチャ (PLA) が構築され、再利用コンポーネントが整備されていきます。即ち、PL開発の技術的側面をカバーすることができるようになります。

ここまで来れば、PLAに沿った開発プロセスやメンバの役割・組織体制のあり方について、具体的なイメージを持つて議論することができるようになります。

SPLの実施に当たり、プロセスや組織体制を最初に変える例も見受けられますが、この場合は経営層およびユーザに確固たる意志と権限がないと上手くいきません。それよりも、技術的基盤を先に構築し、SPLによる製品開発の効率化が具体的にイメージできる状況にすることで、より多くのメンバーを容易に巻き込むことが可能になります。プロセスや組織体制の見直しは、それから始めても遅くはありません。

SPLの導入についてお悩みの方は、本アプローチを採られてみてはいかがでしょうか？

SPL 導入支援

弊社では、ここで説明したRIPPLEアプローチによるSPLの導入支援だけでなく、導入のための事前診断や、プロダクトアーキテクチャ形成のための実践的なSPLトレーニングもサービスとして提供しています。ご興味のある方は、以下のURLで詳細な内容をご覧ください。

<http://www.exmotion.co.jp/service/spi-solution.html>

コンサルティングの価値とは

のっけからブルーな話で恐縮ですが、今、エクスマーシオンは、組み込みソフトウェアの将来を真剣に心配しています。とにかく納期最優先の「動いているから OK」的なソフトウェアで本当に良いのか？莫大なテストによって品質は担保されているものの、この非効率な開発のままでは本当に大丈夫なのか？直近の対応に目を奪われ、問題をどんどん先送りするこの姿勢は、どこかの国の累積した赤字国債を想定せずにはいられません。

確かに表面的には何とかなっているかもしれませんが、無理して頑張っている弊害は、至る所に歪みをもたらします。こんがらがったソースコードを見て育った新人達には、それ以上の設計を期待するのは困難です。し、オシホの理解困難な設計は、忙しくて猫の手も借りたい開発のピーク時に、新メンバーの追加を拒む参入障壁となって立ち塞がります。結局は、急がば回れと言われるように、できるところから変えていかないと、至る所でじわじわと進んでいく負の連鎖を、止めることはできません。

前置きが長くなってしまいましたが、コンサルティングとは、まさしくこの負の連鎖を止めるための「外部の力」に他なりません。今回のテクニカルガイドでは、まずはこの「外部の力」の威力を、組み込みソフトウェア開発の現場にいるみなさんに、純粋に知っていただきたいと考えました。冒頭からの特集は、私たちがコンサルティングを通じて実際にお手伝いしてきたお客様カルテの公開です。これを見ていただければ、コンサルティングの具体的なイメージをつかんでいただけると思います。

もちろん、特集はそれだけではありません。裏面から始まる特集では、機能安全や形式手法、ソフトウェアロジックライオンなどといった技術的キーワードが溢れている現状を踏まえ、これら開発技術の体系化と、エクスマーシオンがキーとして考えている個々の技術についての解説を盛り込んであります。開発者に限らず、管理職や経営層といった方々も含め、この内容をきちんと把握していただければ、巷に溢れる開発技術を正しく俯瞰できるものと確信します。

このところ、風当たりが強い日本の製造業ですが、こらでもう一度エンジニアの魂に火を点けて、グローバル市場に打って出ようではありませんか。そして、その際には、ぜひ私たちエクスマーシオンも「外部の力」として、お手伝いさせていただければ幸いです。

2010年5月 株式会社エクスマーシオン 社員一同



exmotion Technical Guide 2010



改善パターン編

- 03 開発現場の改善パターン
- 05 コンサルティングカルテ

About us

コンサルティング紹介＆会社案内

技術編

- 01 技術マニア
- 03 ボトムアップから始める状態モデリング
- 05 Simulink モデルの品質を改善しよう
- 09 モデル検査の使い方
- 13 機能安全における障害解析と要求管理の統合化
- 17 工学的アプローチによるリファクタリング
- 21 XDDP によるリファクタリングの実践プロセス
- 25 RIPPLE 既存ソフトウェアを活用したプログラマライオン開発の実例

本文書に掲載された文章、画像、及びその他のコンテンツを許可なく複製、転載、引用する事を禁じます。
Copyright (c) 2010 exmotion Co., Ltd. All rights reserved.
MATLAB 及び Simulink は The MathWorks, Inc. の登録商標です。
そのほか記載されている会社名、システム名、製品名は一概に各社の商標または登録商標です。
なお、本文および図表中、「TM」、「®」は明記しておりません。

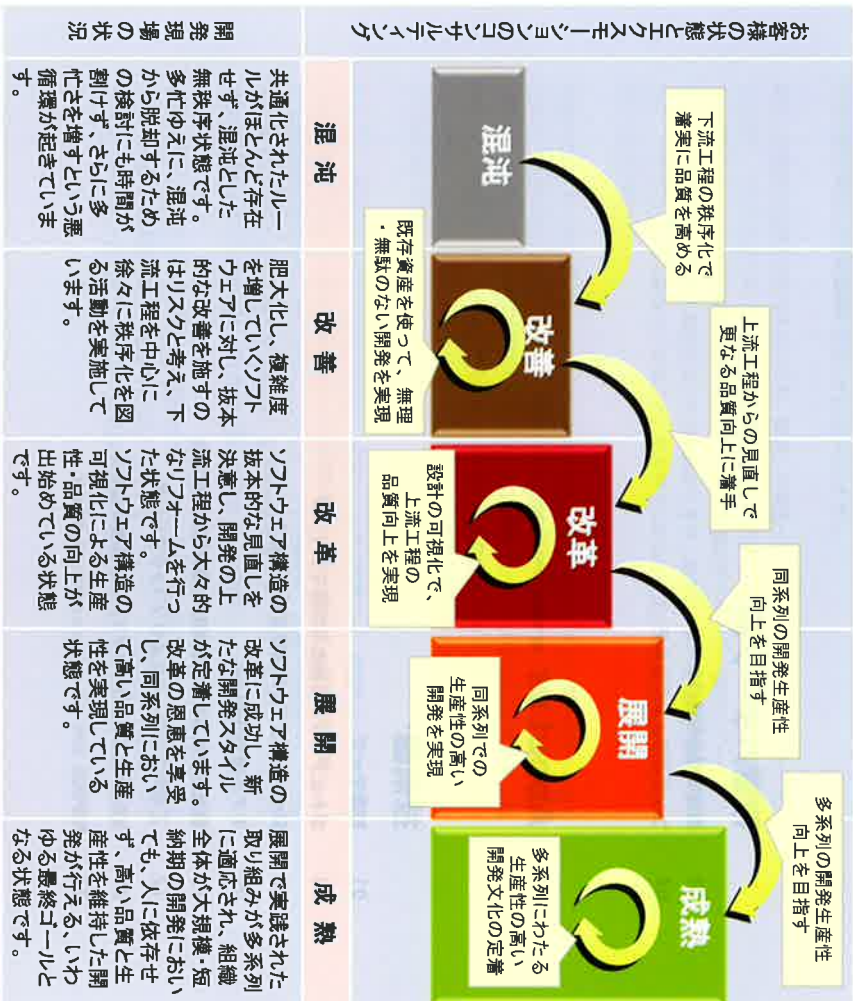
CONTENTS



ソフトウェアの開発力を高めるにはどうすればよいのでしょうか？

世の中には、さまざまな状況の開発現場があります。ソースコード中心の開発は一般的ですが、モデルを使った開発に取りくんでいたり、中には、ソフトウェア部品を活用した効率的な開発を始めている現場もあります。

エクスモーションでは、組織の開発力は5段階の状態に分けられると考えており、各状態にふさわしい改善方法と、有効な技術を定義しています。エクスモーションのコンサルティングは、この定義に則りながら、下の線の矢印で示しているような、「上の状態に移行する」あるいは「状態の中で活動を強化する」ためのお手伝いをすることに他なりません。

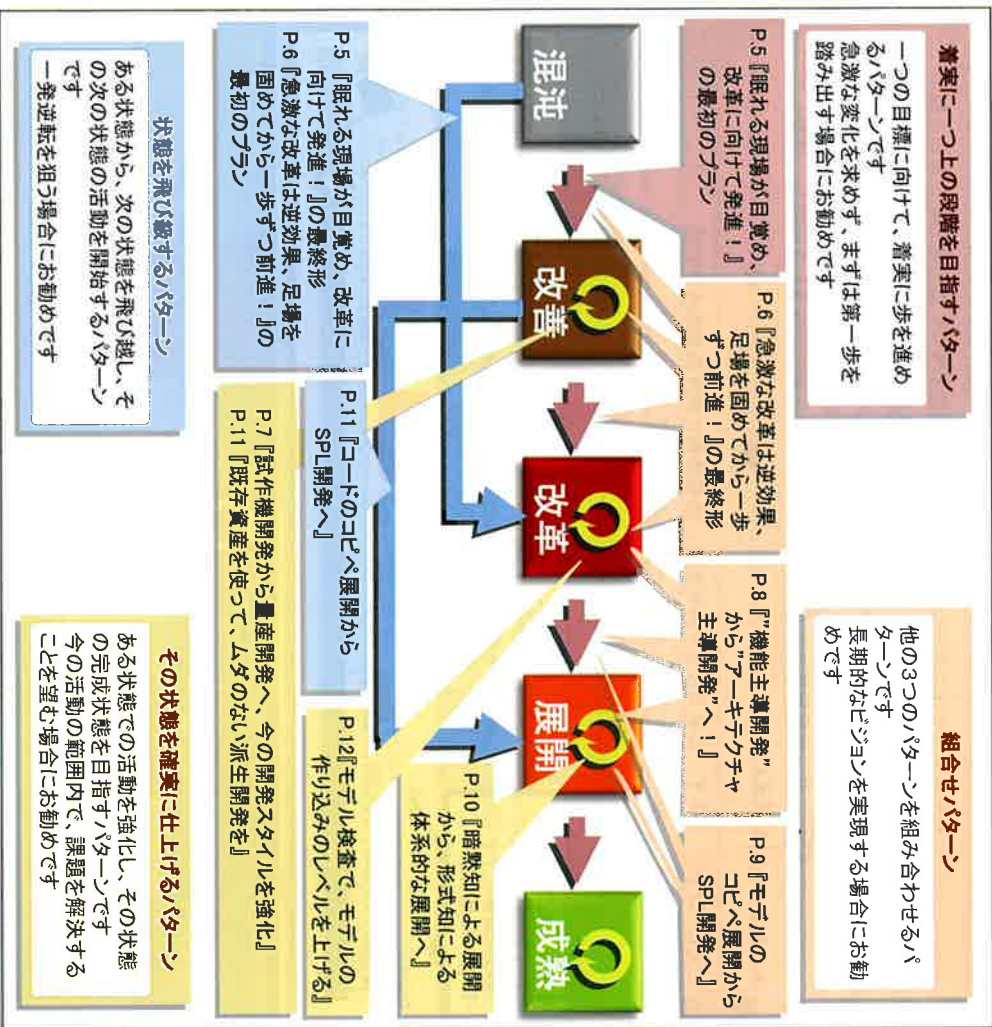


「スタート」と「ゴール」とその間の「道すじ」、改善パターンはいらいます

開発力を高めるための最初の一步は、その開発現場がどんな状況にあるかを把握することです。これにより、改善活動のスタート位置が決まります。次は、最終的にどのような状態を希望しているのかを把握します。これがゴール位置となります。さいごに、スタート位置からゴール位置にたどりつくまでに、何をどのような順序でやっていくかの道すじを決めます。これが改善活動のシナリオになります。シナリオが異なれば、同じスタートとゴールであっても、違う道すじになります。

エクスモーションが手がけてきた、さまざまなコンサルティングを振り返ると、この改善シナリオは、以下の図にあるように、大きく4つのパターンに分類できることがわかりました。

次のページからは、各パターンごとに、実際のコンサルティングでの活動内容を明記した『コンサルティング図』を通じて、実際の改善活動の様子やコンサルティングのイメージを紹介します。



眠れる現場が目覚め、改革に向けて発進！

✓ 症状

信頼できるソフトウェア成果物はソースコードだけです。開発の方法は個人任せで、みんな自分勝手なやり方をしています。ベテランは、現状に問題があると気付いていますが手が出せず、若手はこの状況に染まってしまっています。

✓ スタート

開始時の状態は『混沌』です。
下流中心の問題解決により、成功の実感を得やすい『改善』を目指します。

✓ 道すじ

まずは、ソースコードのクリーニングから始めました。不要な変数や関数の削除、コンパイルを厳密にすることで噴出したブリーニングへの対応etc。混沌とした状況が改善である、ということを開発メンバーは改めて気付いたようです。

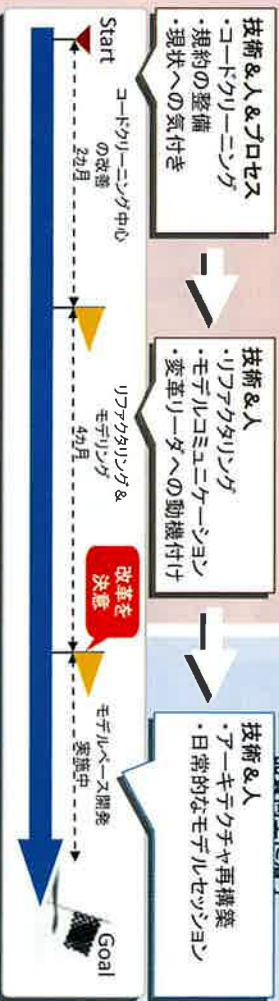
その後、リファクタリングを実施しますが、設計は各自の頭の中にしかなかったため、モデルをコミュニケーションツールとして活用することにしました。自分の頭の中にあることが可視化・整理されていく過程を見ることが、モデルングに対する動機づけにもつながりました。

その結果、今のペースでリファクタリングしては間に合わない、と判断したメンバーが再構築を決議。今では、メンバー全員で『改革』を目指し、活動しています。

✓ ポイント

まず、出来ることから始めて、改善活動に対する自信をつける
リーダーが変革に向けて一歩踏み出す勇気を持つ

下流工程の秩序化で着実に品質を高める



上流工程からの見直しで品質向上に着手



急激な改革は逆効果、足場を固めてから一歩ずつ前進！

✓ 症状

テキストベースの設計仕様書を使って開発していますが、一部の関係者からモデルでの開発を要望されています。しかし、モデルベース開発の経験がなく、何から手をつけて良いかわからない状況でした。

✓ スタート

開始時の状態は『混沌』です。
一気にモデルベースで開発する『改革』を目指します。

✓ 道すじ

まずは、ある一部分をモデルに置き換える活動を始めました。しかし、仕様を理解するために設計仕様書を参考にするものの、理解するのが大変な上に、設計仕様書自体の管理もなされてませんでした。そのため、記載に抜け・漏れ・矛盾がある状態でした。ようやくモデルを作って関係者に説明しても、一部の関係者からは、強い抵抗を受けてしまいました。

その結果、急激な改革は無理と判断し、モデルベースの開発はいったん休止しました。そして、現状の設計仕様書の改善に注力、すなわち足場固めから始めることにしました。その成果として、小さいけれど効果がわかりやすい改善結果を示して、徐々にまわりの理解が得られるようになってきました。

その後、可能な部分からモデルに置き換える改革活動もできるようになりました。

✓ ポイント

無理な改革は逆効果、一歩一歩の前進が実を結ぶ

上流工程からの見直しで品質向上に着手



下流工程の秩序化で着実に品質を高める



試作機開発から量産開発へ、今の開発スタイルを強化！

✓ 症状

試作機開発のフェーズも終盤に近付き、量産開発に向けて準備を始める段階です。しかし、開発意図の多くは開発メンバーの頭の中…。また、量産に入ってから頻繁な仕様変更が考えられ、このままのやり方で品質を確保しつつ上げられるか不安です。

✓ スタート

開始時の状態は『改善』です。一部、モデルを使った開発も行われてはいるものの、まだまだ実装レベルの内容となっており、設計を可視化するまでには至っていません。

✓ ゴール

量産開発になってからも、開発現場が混乱しないように、プロセス改善を中心とした『改善』の完成形を目指します。

✓ 道すじ

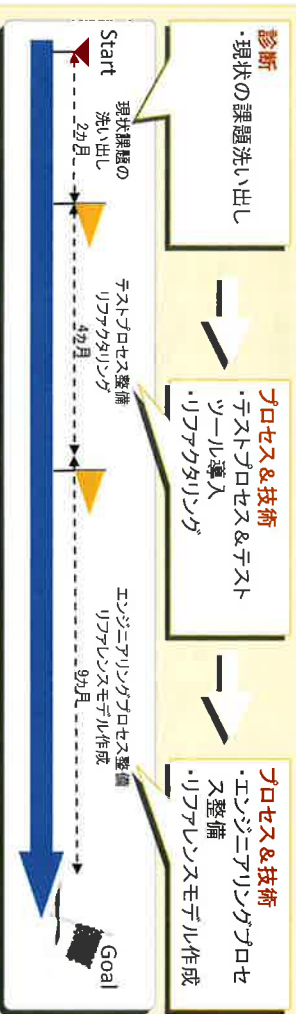
量産開発に移行するとしても、これまで通り、新しい機能の開発や性能向上など、試行錯誤が続きます。とはいえ、市場品質を満足させていく必要もあります。そのため、まずはこれまでの機敏な開発スタイルを残しつつ、市場品質の確保が可能となるよう、品質面でのプロセス強化を行います。

それと同時に、より上流からの設計を目指し、現在の実装レベルのモデルのリファクタリングと、モデルの抽象度を上げていく活動も行います。

✓ ポイント

今の開発スタイルを残しながら、必要な箇所を強化する
中長期的な改善にも、少しずつ手をつける

既存資産を使って無理・無駄のない開発を実現



“機能主導開発” から “アーキテクチャ主導開発” へ！

✓ 症状

C++言語を使っていますが、モデルはほとんど書かれていません。また、担当者に機能を割り振って開発を進めているため、複数の人がクラスを共有しています。そのため、各自の考えで関連や属性が追加されてしまい、開発当初に作られたアーキテクチャはいつの間にか崩れてしまいました。

✓ スタート

開始時の状態は『改革』です。しかし、上流からの再構築にチャレンジした結果、要求定義はうまくいきませんが、ソフトウェアの設計実装に課題があります。

✓ ゴール

『改革』にじじまらず、さらに『展開』までを目指します。アーキテクチャを再構築し、同系列・多機種展開を簡単にできるように目指します。

✓ 道すじ

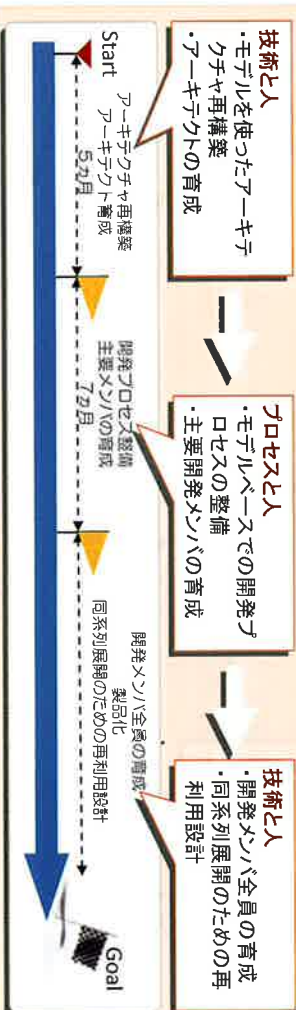
まずは、アーキテクチャチームを中心に、アーキテクチャの再構築と、OJTを中心としたアーキテクチャ育成を実施します。それが軌道にのった道に、開発プロセスの整備に取り組みます。ある程度プロセスが確立したら、アーキテクチャが中心になって、一般の開発メンバーの育成を行います。

まず1機種新しいアーキテクチャにて開発し、その後再利用資産として整理していきます。

✓ ポイント

アーキテクチャ再構築は、まずはアーキテクチャの育成から

設計工程の可視化で上流工程の品質向上を実現し、同系列での高い生産性を実現する



モデルのコード展開からSPL開発へ

✓ 症状

以前、UMLモデルを使った開発に取り組み、上流工程からの品質向上を実現しました。しかし、シリーズ展開する際に、モデルとコードをコピーして使い回したために、思ったように開発生産性が上がりません。シリーズ内で類似コードが多く存在し、アーキテクチャにもいつの間にかほころびが見え始めています。

✓ スタート

開始時の状態は『改革』が終わった状態です。

✓ ゴール

『展開』の完成形を目指します。同一系列の多機種にわたり、部品を使った効率的な製品開発を目指します。

✓ 道すじ

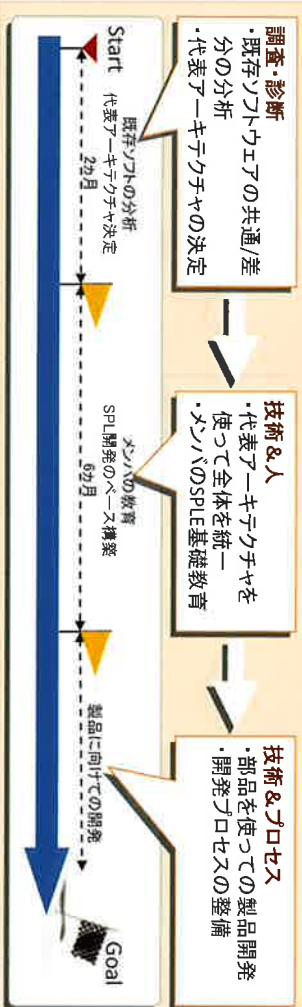
まずは、現状のソフトウェアの状況がどうなっているかを調査します。機種間の差異部分がどれくらいあり、共通部分がどれくらいあるのか？代表アーキテクチャはどの機種とすればよいのか、などです。

代表アーキテクチャが決まれば、他の機種の差分を加えて、一つに統一します。そこまでできたら、今度は部品部分を決め、そこから複数の製品が導出できるよう、設計と実装の整理・整頓を行います。(RIPPLEアプローチ)

✓ ポイント

現状分析に基づき、効果的な手段を選ぶ

同系列での生産性の高い開発を実現



“暗黙知による展開” から “形式知による体系的な展開”へ

✓ 症状

ある成功プロジェクトでのやり方を、組織横断的に展開しています。しかし、展開する技術ノウハウが担当者の中にとり、効率的に展開できていたとは言えません。また、その技術の効果についても曖昧な状況です。

✓ スタート

開始時の状態は『展開』です。

✓ ゴール

ある組織の中で確実に水平展開できる『展開』の状態の完成形を目指します。

✓ 道すじ

まずは、展開する対象である技術を形式化します。具体的には、担当者の中にある情報を外に出し、その情報を、使う側の人の立場で体系化します。

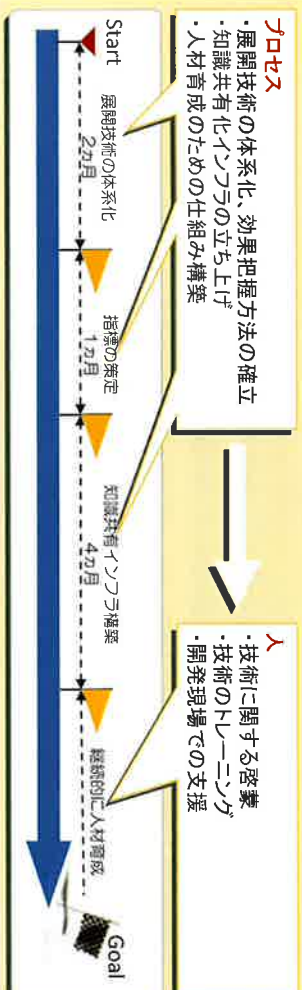
次に、技術の効果を把握するための指標を決めます。効果的な指標を得るためには、実際のプロジェクトでの計測を行うなどの活動が必要で。

その次に、知識を共有化するためのインフラを構築します。最後に、それらの成果を活用して展開活動を継続的に実施します。

✓ ポイント

横断組織の担当者を『開発者の延長』の状態から『展開のプロ』に意識転換・動機づけ
展開する技術を共有資産として位置付ける

これまでの蓄積を、同系列へ適用



ここまで紹介してきたパターンの他に、エクサモーションとしてお勧めする3つの改善パターンをご紹介します。

まずは、『改善』がスタートの状態となる2つのパターンを紹介しましょう。

コードのコード展開からSPL開発へ

ソースコードベースでのSPL開発です。既存ソフトウェアのアーキテクチャの品質状態によっては、再構築をしないで、SPL開発に移行することができます。

こんな場合にお勧め

- 製品をシリーズ開発している
- 製品市場は、価格競争・リソース競争
- アーキテクチャはそんなに悪くないと思う
- コピー&ペーストが日常的で、似たようなコードが氾濫している
- 新しい技術を取り入れるのに、さほど抵抗はない

改善パターンの特徴

- ✓SPL開発に移行し、製品ファミリーの開発を効率化します
- ✓現在のアーキテクチャを活用し、大幅なアーキテクチャの再構築はしません
- ✓既存資産をできるだけ使います



※SPL開発の関連記事はTechnical - 25 にあります

典型的な進め方

- 1.対象とする製品シリーズのコードを全て調査し、共通点と相違点を分析します
- 2.代表的なアーキテクチャを決めて、コードを統合します
- 3.統合したコードから、複数製品の開発ができるように、設計しなおします

既存資産を使って、ムダのない派生開発を

SPL開発ではないですが、XDDPという技術を使って、派生開発を実現します。手戻りができるだけしないようなプロセスにより、ムダを作らないようにします。

こんな場合にお勧め

- 既存コードの保守開発がメインである
- 製品市場は、価格競争・リソース競争
- アーキテクチャはそんなに悪くないと思う
- 導入コストはできるだけ抑えたい
- プロセスを重視して改善を進めたい

典型的な進め方

- 1.ソースコードを調査し、設計仕様を文書として起こす
- 2.追加・変更の要求は様と、コードとの対応関係、追加・修正のやり方をきちんと詰める
- 3.詰めたやり方でコードを変更する。



※XDDPの関連記事はTechnical - 21 にあります

次にお勧めする改善パターンは、現在モデルベースの開発をしている人にお勧めです。

モデル検査で、モデルの作り込みのレベルを上げる

モデルの動的側面でのモデル検査により、モデルの段階での作り込みのレベルを上げることができます。そのために必要な技術は、形式手法です。

こんな場合にお勧め

- モデルで開発している
- モデルにヌケ・モレ・矛盾があり、事前に何とかしたい
- できるだけ、上流の作りこみ時での完成度を高めたい

改善パターンの特徴

- ✓形式手法の考え方に基づきます
- ✓要求工程から設計工程まで、モデル検査を実施することが可能です
- ✓ツールを使って、ある程度自動化した検査ができます



※モデル検査の関連記事はTechnical - 9 にあります

いかがでしたか？

実際のコンサルティング活動に基づくコンサルティングカルテを6件、お勧め改善パターンを3件ご紹介いたしました。みなさんの状況にピッタリのものがありましたか？

エクサモーションでは、『診断』サービスでお客様の現場の今の状況を明らかにし、お話を伺いしただうえで、お勧め改善パターンを提示します。そして、技術・人・プロセスの全ての面でサポートする『現場支援』、実践力を強化する『人材育成』を通じて、現場でお客様と一緒に、その次の段階によっていきます。（お問い合わせは info@exmotion.co.jp まで）





Mika Yoshimura
エグゼクティブコンサルタント
芳村 美紀

技術導入と人材育成を同時に行うことにより、皆様によるお客様の長期的な発展をご支援いたします

日本語 英語 中国語



Kazunori Yamachi
シニアコンサルタント
山内 和幸

単なる技術の適用を超えた、真の意味での課題解決を私たちがエクスモーションのコンサルタントが支援します

日本語 中国語 英語



Muneaki Kohama
シニアコンサルタント
小浜 宗隆

コンサルタントが現場に身を投じ、実務者・当事者の視点を得ることなく、お客様と共に課題解決に当たります

日本語 中国語 英語



Toshinori Fujikura
シニアコンサルタント
藤倉 俊幸

検証しながら作るアプローチで、正しいプログラムを、正しく厳密に作るお手伝いをいたします

日本語 中国語 英語



Ikuoaki Matsumae
エグゼクティブコンサルタント
渡辺 博之

「一スコード中心の開発」から「モデル中心の開発」へと舵を切るためのお手伝いを開発者のスキルアップの段階から支援いたします

日本語 中国語 英語



Koji Iiyama
エグゼクティブコンサルタント
井山 幸次

「しなければならぬ」のではなく、開発者が自ら「取り組みたい」ような技術導入と、同時に動機づけをご支援します。

日本語 中国語



Kenichi Saito
シニアコンサルタント
斎藤 賢一

モデリソング技術を中心としたソフトウェアプロセス改善により、上流工程中心の開発スタイルへの移行を支援いたします

日本語 中国語 英語



Shuichi Horiguchi
シニアコンサルタント
堀口 修一

ソフトウェア開発現場での「クラウド図」を大切にした現場改善活動を支援いたします

日本語 中国語 英語



Yutaka Mitani
コンサルタント
三輪 有史

問題を整理説明し、本質に沿った制御を構築することで、仕様変更や機能展開に強い安定した制御を作ります

日本語 中国語 英語



Shigeru Kosaka
エグゼクティブコンサルタント
小坂 優

みなさんの開発現場にある「むやう」と「すまじ」に、エクスモーションは幅広いシステム開発を現場から支援いたします

日本語 中国語 英語



Junji Yamaki
スペシャリスト
玉木 淳治

ソフトウェアの品質を多面的・定量的に測定して可視化するツールを提供し、現場での品質改善活動を推進します

日本語 中国語 英語



Hisonori Takahashi
エグゼクティブコンサルタント
高橋 久憲

適切なモデリソング技法をご提案することで、お客様のコミュニケーションを円滑にご支援いたします

日本語 中国語 英語



Tomoyuki Fukunaga
スペシャリスト
福岡 呂之

お客様の開発領域のモデリソングを効率化するツールを開発することで、幅広いソフトウェアのモデリ開発を支援していきます

日本語 中国語 英語

COMPANY PROFILE

会社概要

社名 株式会社エクスモーション (英字名称: exmotion Co., Ltd.)

代表取締役社長 長尾 章
専務取締役 渡辺 博之
常務取締役 芳村 美紀

主な事業内容 組込みソフトウェア開発力の向上につながる各種支援事業

ソフトウェアインフラ事業
プロジェクト診断 (問題発見と改善策の作成・提案)
現場支援 (改善策の実行に必要な技術支援)
一スコード中心の開発 / Simulinkモデルの作成・改良 / UMLモデルの作成・改良 / ソフトウェアプロジェクト管理に向けた体系的支援 など

教育・人材育成事業

現場で活用できるスキル習得のための実践的コースの提供
一スコード開発スキル / UMLモデリングスキル / ソフトウェアプロジェクト管理の実践スキル など

ツール開発事業
一スコード診断ツール「exauto」
UMLからSimulinkへのモデル変換ツール「mtrp」など

TEL 03-6722-5067
FAX 03-6722-5057
URL <http://www.exmotion.co.jp>
〒108-0014 東京都港区芝5-33-7 徳栄本館ビル8F (受付 9F)
所在地