システムアーキテクチャ

ＵＳＤＭ要求

ＭＢＤ

ＵＭＬ

レガシー救済

機能安全

解説書作成

ＳＰＬ

ＸＤＤＰ派生開発

emotion

エントロピーを抑え込め！

# ソリューションガイドブック 8

# システムアーキテクチャ構築
## ～全体最適なシステムを設計する～

**最近は、製品機能の大規模・複雑化に伴い、単一のシステムに留まらない、複数システムを相互接続したシステム、いわゆる「システム・オブ・システムズ（System of Systems）」と呼ばれるものが増えてきました。このようなシステムでは、求められる要求や制約が多岐に渡ります。それらを満足するアーキテクチャを構築するためには、適切な抽象度を保ちつつ、常に全体最適を考えることのできる開発手法が必要です。**

## システムアーキテクチャとは

システムとは大小様々な要素で成り立っており、この「要素」と「要素間の関係」を定義したものを構造的なシステムアーキテクチャと呼びます。

システムアーキテクチャの構築にあたっては、ユーザの要求を満たすだけでなく、設計のしやすさやテストのしやすさ、運用・保守まで考慮しなければなりません。また、システムが動作する環境において、さまざまな制約やトレードオフが存在するため、これらへの十分な配慮も大切です。

このように、システムアーキテクチャの構築においては、システムのライフサイクルを見渡し、そのすべてにおいて問題なく進められるよう、全体最適な視点に基づいた設計作業を進めることが求められます。

## システムアーキテクチャ構築プロセス

図1

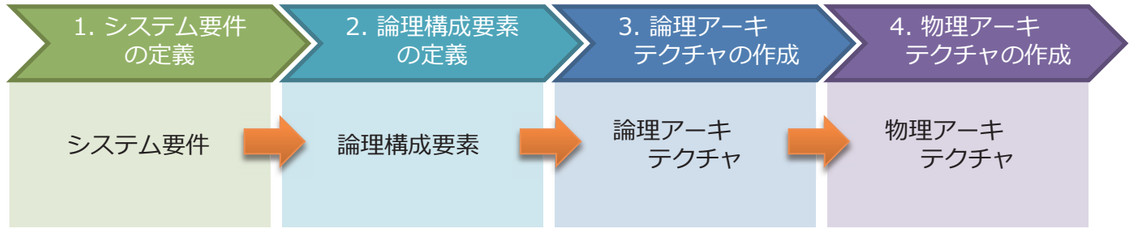



図1は、システムアーキテクチャを構築するための基本的なプロセスを示しています。

「1. システム要件の定義」では、システムに求められる機能要求と非機能要求を明らかにします。全体最適なシステムを設計するには、事業環境や商品戦略なども踏まえたシステムレベルで要求を捉えることが重要です。

「2. 論理構成要素の定義」では、システム要件を実現するために必要な機能を導出していきます。ここで導出された機能がシステムアーキテクチャの構成要素となります。

「3. 論理アーキテクチャの作成」では、導出した構成要素間の論理的な関係を定義します。具体的には、どの要素とどの要素が協調し、そのときのインタフェースは何かを決めていきます。

「4. 物理アーキテクチャの作成」では、論理要素に対する物理要素を定義します。物理要素を定義するには、処理性能や調達コストなどの制約が大きく関係してきます。様々な要因のトレードオフ分析を実施した結果、最終的にシステム要求、及び制約を満たす最適な物理アーキテクチャを定義します。

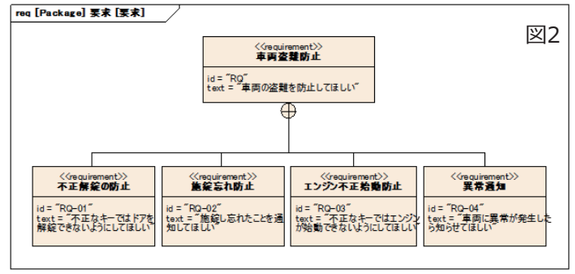
次ページ以降で、これら４つの工程についてより詳細に説明していきます。

## 1. システム要件の定義

図2は、「車両盗難防止システム」の要求をSysMLの要求図で記述したものです。車両のステークホルダーである所有者からの「盗難を防止してほしい」という要求を、さらに細分化された４つの要求「不正解錠の防止」、「施錠忘れ防止」、「エンジン不正始動防止」、「異常通知」によって実現することを示しています。

この４つの要求はすべてシステムに対する「機能要求」です。一方、これら機能の実現をどのように達成すればよいかは「非機能要求」として明確にします。非機能要求を抽出するには、ISO/IEC25010「システム及びソフトウェア品質モデル」などの網羅的なフレームを参照すると効率的です。また、関連する法規や環境規制など、システムが必ず守らなければならないものはシステム制約として整理します。

図2




## 2. 論理構成要素の定義

図3

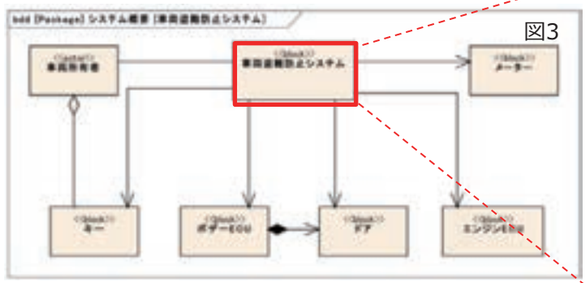



図4

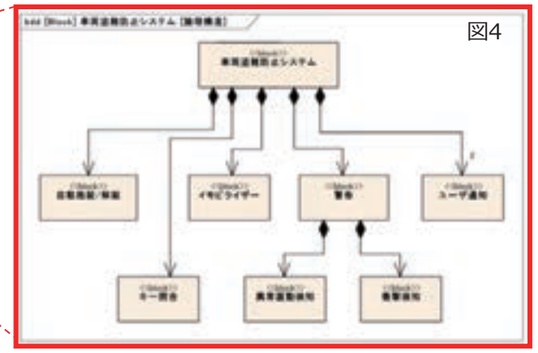



図3は、「車両盗難防止システム」の要求に基づき、システム構成をSysMLのブロック定義図を使って表現したものです。この図からは、このシステムが、ボデー、エンジン制御、メーターといった既に存在する構成要素と関連して成り立っていることが分かります。

図4は、「車両盗難防止システム」の内部の構成要素を表したものです。今回は「キー照合」「自動施錠／解錠」「イモビライザー」などといった機能と、それをユーザに伝える「警告」とそれの詳細である「振動検知」「衝撃検知」、別の通知手段である「ユーザ通知」が構成要素となります。

## コンサルタントが教える成功の秘訣

どのようなシステムにもシステムアーキテクチャは必ず存在します。しかしそれは適切に設計されたものではなく、ソフトウェアとハードウェアを個別に設計した結果を現場の担当者がすり合わせているというケースが少なくないようです。そのように定義されたアーキテクチャでは、継続した拡張や、複数バリエーションへの展開などに困難がつきまといます。

システムアーキテクチャは、それがシステムを作る目的にかなっているか、企業の事業戦略に合致しているかを説明できる構造になっていなければなりません。これは機能個別の設計をしているときには抜け落ちてしまいがちな視点です。常に全体最適の意識を持ちながら、後々まで役に立つシステムアーキテクチャを作り上げましょう。

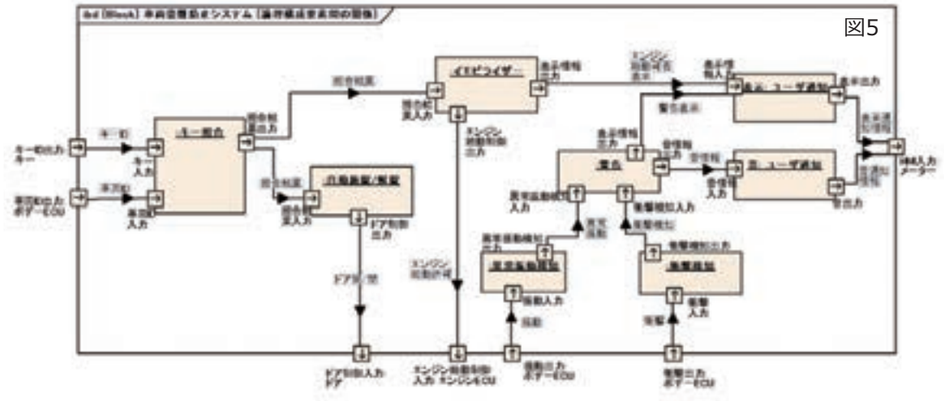
コンサルタント
庄司　順和

## 3. 論理アーキテクチャの作成

システムの構成要素が決まったら、続いて構成要素間の関係を定義します。図5は、構成要素間の関係をSysMLのブロック図で表現しています。定義した構成要素それぞれをブロックとして置き、ブロック間のインタフェースを決めています。

この図からは、「キー照合」の結果を受けて「自動施錠/解錠」、「イモビライザー」が作動し、最終的に表示と音でユーザに通知するという流れが読み取れます。このように、構成要素とそれらの関係を定義することで論理アーキテクチャを作成します。

図5




## 4. 物理アーキテクチャの作成

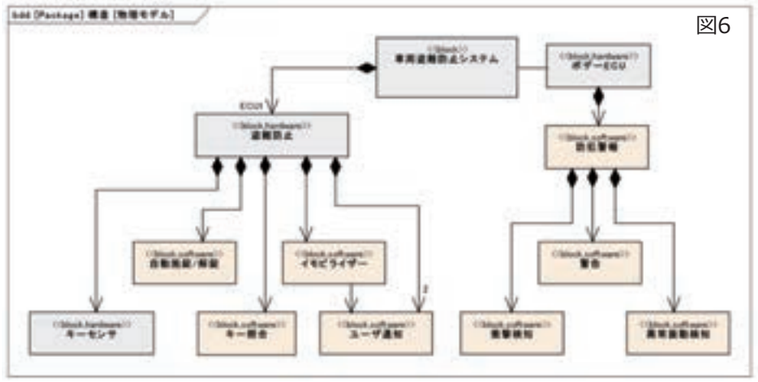
図5で示した論理アーキテクチャに対して、非機能要求や制約に配慮した実際の「モノ」としての構成を物理アーキテクチャと呼びます。物理アーキテクチャの構築には、まずは下表のように、「車両盗難防止システム」の非機能要求を実現するために必要なアーキテクチャ制約を導出し、先に定義した論理アーキテクチャに対しこれを満たすような変更を施していくことになります。

| カテゴリ | | 非機能要求 | アーキテクチャ制約 |
|---|---|---|---|
| 性能面 | 効率性 | キー照合から500ms以内でエンジン始動およびユーザへの通知を完了すること | キー照合から通知までは同じCPUで処理 |
| 開発効率性 | 保守性 | 暗号プログラムを外部からバージョンアップできること | 暗号プログラムの格納方法 |
| ビジネス面 | 調達有無 | 暗号化技術は外部の専門企業から調達 | 暗号プログラムの分離 |
| | 仕向け | 防犯警告は標準搭載、自動施錠・解錠とイモビライザーはオプション | 防犯警告と、自動施錠・解錠およびイモビライザーの分離 |

表のアーキテクチャ制約に対し、さまざまなトレードオフ分析を行い、全体最適な視点で決定した物理アーキテクチャが図6です。

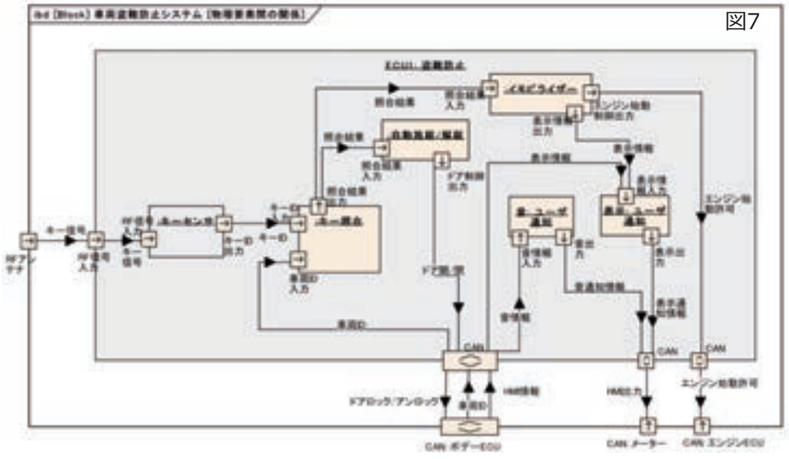
ここでは、ハードウェアとソフトウェアの区別、および、それらがどのように配置されるかを見て取ることができます。今回は、性能面およびビジネス面での制約から、盗難防止とボデーECUの2つのCPUに各構成要素となるブロックが分割配置されたことが分かります。

図6




さらに図7のように、各要素間のつながりや、やり取りされる情報などを表現できる内部ブロック図を活用することで、設計したアーキテクチャの妥当性をモデルを使って早い段階で検証することができます。このような手順を踏むことで、多様で大規模なシステムに対しても、トップダウンかつ全体最適なアーキテクチャを事前に検討することが可能になります。

図7




## システムアーキテクチャ定義言語

車両盗難防止システムの例で示したように、システムアーキテクチャを定義するための言語としてSysMLが挙げられます。SysMLは自由度の高い汎用的なモデリング言語であり、様々なシステムを定義可能です。

SysMLのように自由度の高い汎用的な言語がある一方、自動車システム向けの「EAST-ADL」のようにドメインに特化したシステムアーキテクチャ言語も存在します。EAST-ADLでは、分析レベルや設計レベルといった階層の定義に加え、ISO26262やプロダクトラインなどの概念も取り込んでいます。システムアーキテクチャとしてどこまで定義すれば良いか検討する際に、EAST-ADLのような標準規格を参考にするのも効果的です。

## システムアーキテクトを目指して

システムアーキテクチャは一度構築すればそれで終わりではなく、システムのライフサイクル全般に渡って維持していかなければなりません。また、複数のシステムに対して同じようにシステムアーキテクチャを定義していくことで、機能の統合や相互接続などの全体最適を実現するための検討が進むようになります。

システム開発を行う企業においては、システムアーキテクチャ設計を継続していくシステムアーキテクトの存在が不可欠です。また、システムアーキテクトにとっては、全体最適なシステムを設計することで企業の事業戦略を大きく推進する役割を果たせます。システムアーキテクトを目指し、真に役立つシステム設計を実現させましょう。

# 要求の定義と仕様化

ソフトウェア開発の最上流工程である要求定義で誤りがあると、いくらその後の設計が正しく行われたとしても、やり直しが必要となります。一般的に、不具合の発見が遅れるほど、それを解決するための工数が多く発生すると言われ、上流での問題を早く解決することが、生産性を上げるための重要なポイントと言えます。

## 開発現場には”要求”はなく、”製品スペック”か”制御方法や機能の仕様” しかない

一般的な組込みソフトウェアの開発現場では要求と言われるものはほとんど記述されていません。

あるものと言えば、製品スペックか、タイミングチャートなどの制御の細かな仕様や機能の仕様を記載したものだけで、対応する要求や、仕様のヌケモレの有無は判断できません。その結果、仕様の不備がソフトウェアに不具合として混入してしまいます。

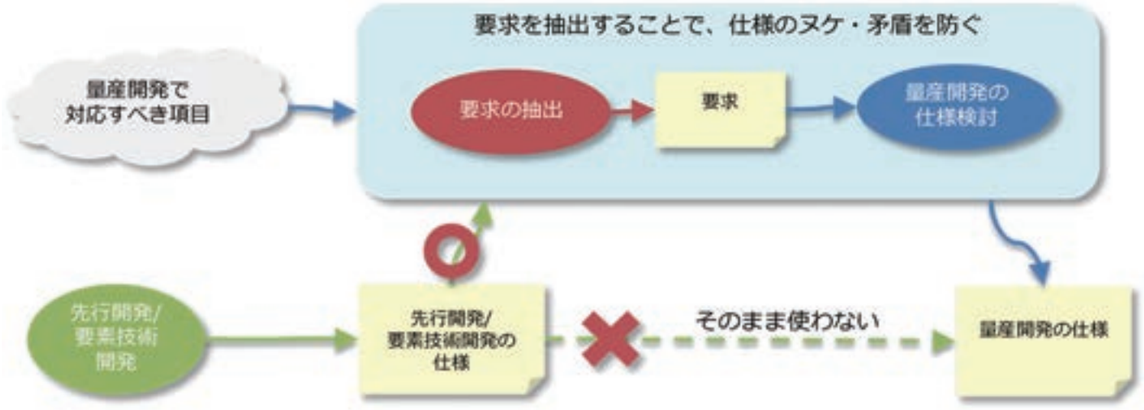
要求が記述されない理由は、量産開発の前に先行開発や要素技術開発が行われることが多く、そこで決まった仕様や成果物を基に量産開発が行われるからです。量産開発でも設計、実装を行いながら仕様を決めていくやり方で開発が進むため、ソフトウェアを統合してテストするまで仕様の矛盾やヌケモレが発見されません。

## 先行開発や要素技術開発の仕様から要求を抽出する

仕様の目的や意図を明確にしたり矛盾やヌケモレを防ぐには、先行開発や要素技術開発の仕様を量産開発でそのまま使わず、量産開発で対応すべき項目も加味した上で要求を抽出・定義し、それらと先行開発や要素技術開発の仕様を基に改めて量産開発の仕様を検討します。設計、実装の前に仕様を決定することで、手戻りを最低限に抑えられます。

ここで、仕様の作成は設計工程の作業だと考える方もいるかもしれませんが、設計は本来なら仕様が決まらなければ開始できないはずです。というのも、仕様とは要求を実現するためのシステムの具体的な動きや制限を表したもので、設計とはその仕様を実現するための構成、その構成要素の役割、構成要素間の相互作用を検討する工程だからです。

また、ソフトウェアコンポーネント設計の工程に入るとコンポーネントごとに担当者や部署が分かれる場合も多く、各担当者が設計を行いながら仕様を決めることになると、ひとつの機能の実現に関係する複数のコンポーネントの担当者間でコミュニケーションのミスが発生しやすくなり、不具合につながってしまいます。このことからも、仕様の作成は設計の前に行うべきと言えます。




## 要求を抽出するにはどうしたらよいか？

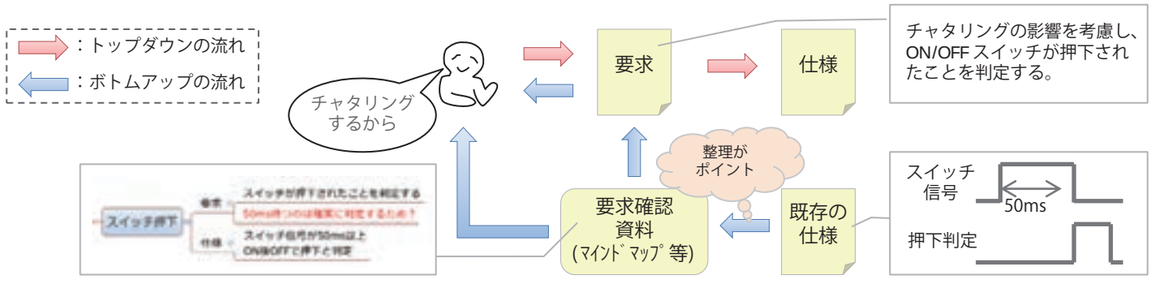
### ■ボトムアップから要求を作り出す

一般的な要求抽出のやり方としては、利害関係者へのインタビューや自社製品や競合他社の製品調査などが挙げられます。このようなやり方は“要求の源泉”からたどっていくという意味でトップダウンと呼ばれます。しかし、この方法では“あるべき姿”を一からあぶり出すことができる反面、今ある仕様を網羅することに対しては不安が残ります。

詳細な仕様がすでにある場合には、ボトムアップからスタートする方法も有効です。詳細な仕様からマインドマップなどの情報を整理する手法を使って仕様の背景にある要求を抽出していきます。

もし、仕様が書かれていない場合は既存資産から仕様を起こす作業を行います。詳しくは「XDDPによる派生開発」の記事をご参照ください。

抽出した要求を基に既存システムや先行開発の担当者にヒアリングをし、抽出した要求が正しいか、他に隠れた要求が無いかを確認していきます。




### ■品質要求の重要性

制御仕様や機能仕様には、システムの機能的な動作に関することや設計制約などは記載されていますが、品質に関する要求は記載されていません。しかし、品質要求はその製品に対して人が持つ印象に大きな影響を与えるため、品質要求を把握することはよい製品を開発する上でとても重要なことです。

品質要求には使いやすさなどの製品のユーザに関係する品質と保守性などの開発者に関係する品質があります。

ユーザに関係する品質はセールスからの要望やユーザクレームの情報等があるため重視されやすいのですが、開発者に関係する品質は見落とされがちです。しかし、開発者に関係する品質要求は、開発の効率や工数に大きな影響を与えるので、良い製品を作ることができる組織を維持するためにはこれらの要求も定義しなければなりません。

## コンサルタントが教える成功の秘訣

私が関与した開発現場では、開発のインプットとして具体的な制御方法について細かく記載したものしかありませんでした。そのため、仕様の妥当性が分からず、今の設計構造にも問題があるのではないかという不安を抱えていました。そこで、細かい制御仕様を抽象化して要求を出し、その要求をブレークダウンする形で仕様を定義するようにすることで、要求を満足するのに十分な仕様であることを把握でき、設計構造を見直すこともできるようになりました。

良い要求仕様を作るためには、“要求”と“仕様”を切り離す必要があります。そのためには、制御仕様を抽象化して要求を得ることが必要です。しかし、長年細かな制御仕様に慣れている開発者の方には抽象化が難しいようです。その分野こそエクスモーションの強みです。ぜひ、お任せください。

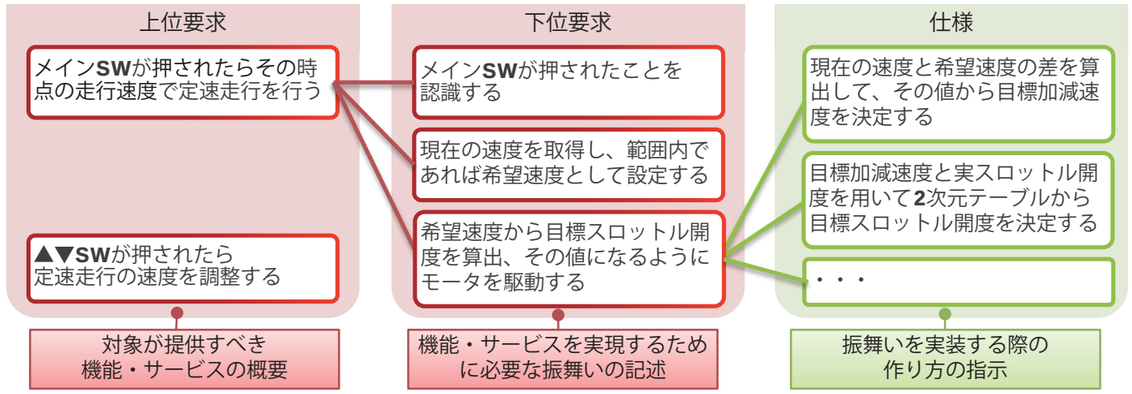
シニアコンサルタント
高橋　久憲

## 要求を仕様化するにはどうしたらいいか？

### ■仕様は開発者への作業指示

仕様とは前述の通り要求を実現するためのシステムの具体的な動きや制限を表したものです。これは開発者への「どのように作るか」という作業指示と言い換えられますので、開発者が設計・実装するのに十分な詳細さで記載する必要があります。

下図はクルーズコントロールシステムの要求と仕様の例です。システムの「目標スロットル開度を算出する」という要求の実現方法として「目標加減速度と実スロットル開度を用いて目標スロットル開度を決定する」という仕様を導出しています。




### ■仕様のヌケモレを防ぐには

要求を基にして仕様を作成しても、ただ思いついた仕様を書くだけではヌケモレは防げません。作成した仕様をレビューしてヌケモレを検出する方法も有効ですが、レビューは多くの工数を必要とするので、これだけでは効率が良くありません。

そこで、仕様を作成する段階でヌケモレを防止する仕組みを導入すれば、最初から品質の高い仕様を作成でき、レビューに必要な工数も削減できます。

その仕組みとしては、時系列分割、構成分割、状態分割、共通分割という思考のフレームワークが有効です。例えば「構成の視点で要求を分割するとどのような仕様が導出できるか」と考えることで、適切な仕様を導出します。これらのフレームワークは仕様を導出する場合だけではなく、上位要求から下位要求を導出する場合にも用いることができます。さらに、このように導出した仕様からテストシナリオを抽出してテスト仕様書を作成することで、テストの段階でもヌケモレを防止できるというメリットもあります。詳細は15ページの「要求仕様書とモデルからテスト仕様書を作成」をご覧ください。

| 観点 | 説明 |
|---|---|
| 構成分割 | 時間的な順序を持たない「機能」や「構成」に着目して仕様を導出する |
| 時系列分割 | 時間軸に沿った動作・処理に着目して仕様を導出する |
| 状態分割 | すでに見えている「状態」に着目して仕様を導出する |
| 共通分割 | 複数の要求や仕様に共通する部分を切り出す |

## 要求と仕様をUSDMで階層化して表現する

要求を抽出したり要求から仕様を導出したりしても、それらを適切にまとめることができなければ、開発に有効な成果物にはなりません。

適切に要求と仕様をまとめ上げる方法のひとつにUSDMという方法があります。USDMは主に自然言語を用いて要求と仕様を階層化して整理するところに特徴があります。派生開発プロセスのXDDPでも変更要求仕様の記述方法として採用されています。

要求と仕様を対応付けることで、その仕様がどの要求を実現するためのものなのか、全ての要求がヌケモレなく仕様まで具現化されているかを確認することができます。

また、USDMでは機能仕様や制御仕様では扱わない保守性などの品質要求も記述することができるため、開発者への設計方法に関する作業指示も要求仕様書に含めることができます。

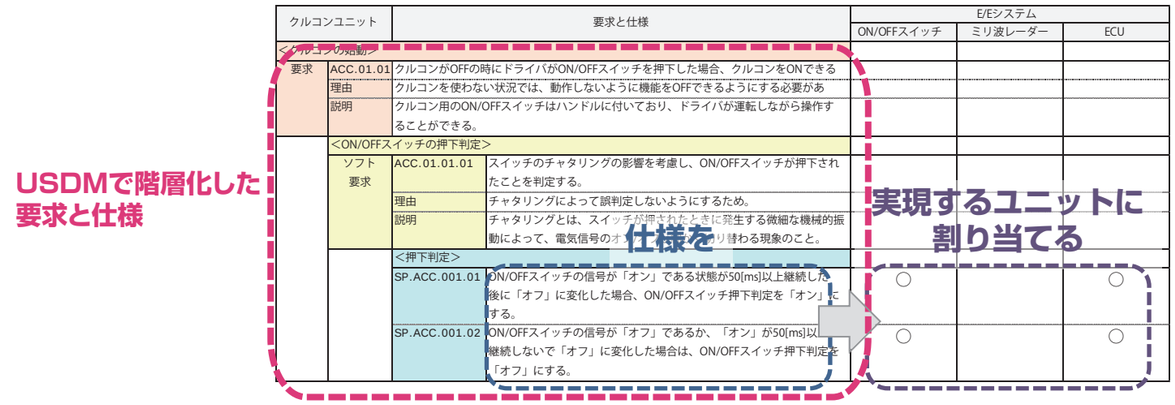
## 要求の定義と仕様化が特に必要なケース

要求の定義と仕様化はソフトウェア開発を行う場合はどのようなケースでも重要な工程ですが、特に必要性の高いケースを３つご紹介します。

まず、開発対象の製品が派生開発である場合です。派生開発では変更管理のプロセスをうまく回すことが大切ですが、USDMを用いて変更要求仕様書を作成することで、変更の実施前に変更内容を明確にでき、間違った方法で変更してしまうことを防げます。

次に、既存ソフトウェアのリファクタリングを行う場合です。実施前と後でソフトウェアの振る舞いが変わっていないことを確認するために元の仕様を明文化する際、USDMを使用できます。

最後に、機能安全に対応する場合です。要求仕様とシステム要素との対応付けが必要なので、要求仕様を明確にする必要があります。そこで、USDMのフォーマットを拡張し、要求仕様とシステム要素の関係を下図のようなトレーサビリティマトリクス（要求仕様システム要素トレーサビリティマトリクス：RSSETM）で示すことで、要求仕様をヌケモレなくシステム要素に割り当てられます。




| クルコンユニット | 要求と仕様 | | E/Eシステム | | |
|---|---|---|---|---|---|
| | | | ON/OFFスイッチ | ミリ波レーダー | ECU |
| <クルコンの挙動> | | | | | |
| 要求 | ACC.01.01 | クルコンがOFFの時にドライバがON/OFFスイッチを押下した場合、クルコンをONできる | | | |
| 理由 | | クルコンを使わない状況では、動作しないように機能をOFFできるようにする必要があ | | | |
| 説明 | | クルコン用のON/OFFスイッチはハンドルに付いており、ドライバが運転しながら操作することができる。 | | | |
| <ON/OFFスイッチの押下判定> | | | | | |
| ソフト要求 | ACC.01.01.01 | スイッチのチャタリングの影響を考慮し、ON/OFFスイッチが押下されたことを判定する。 | | | |
| 理由 | | チャタリングによって誤判定しないようにするため。 | | | |
| 説明 | | チャタリングとは、スイッチが押されたときに発生する微細な機械的振動によって、電気信号のオン・オフが繰り返し入れ替わる現象のこと。 | | | |
| <押下判定> | | | | | |
| | SP.ACC.001.01 | ON/OFFスイッチの信号が「オン」である状態が50[ms]以上継続した後に「オフ」に変化した場合、ON/OFFスイッチ押下判定を「オン」にする。 | ○ | | ○ |
| | SP.ACC.001.02 | ON/OFFスイッチの信号が「オフ」であるか、「オン」が50[ms]以上継続しないで「オフ」に変化した場合は、ON/OFFスイッチ押下判定を「オフ」にする。 | ○ | | ○ |

## 要求と仕様を要求管理ツールで管理する

USDMで整理した要求仕様とその階層関係を要求管理ツールによって管理することもできます。

例えば、要求と仕様に多対多の関係があるなどUSDMでは確認することの難しい複雑な関連が存在する場合には、ツールを利用することによって目的の要求や仕様の関連のみを抽出して関連性を確認することができます。これは、機能安全で要求されるトレーサビリティの確保にも繋がります。

もちろん、使い慣れたExcelで記述されたUSDMの方が閲覧・編集ともに扱いやすく、要求や仕様の可読性も高いと思われます。目的に応じて、これらを使い分けることが重要です。

# ＭＢＤオートモーティブ

**車載ソフトの開発現場は、ハイブリッド化・EV化などのような新たなパワートレインへの転換や、ISO 26262の規格化など、大きな変化を迎えています。**
**これを受け、MATLAB /Simulinkによるモデルベース開発の導入など、生産性と品質を向上させる取組みが進みつつあり、大きな成果を発揮しています。**

## モデルベース開発を導入することで得られるメリットは大きいが…

今、車載ソフトの開発現場では、MATLAB/Simulinkによるモデルベース開発（以下、MBD）が急速に普及しつつあります。その主な理由は、MBDを導入することで、従来開発に比べて生産性と品質へのメリットが得られる点にあります。

例えば、MBDで使用されるツールであるMATLAB/Simulinkには数多くの部品が用意されており、これらをつなぐことで簡単にモデルを作ることができます。作ったモデルはその場ですぐ動きを確認できるため、モデルで記述した制御仕様の検証が容易になり早期の品質確保が可能となります。

更に、作ったモデルをそのまま実車に持ち込んで動作検証を行ったり、モデルからオートコード（自動コード生成）することにより実装工数を大幅に削減するなど、MBDの導入により多くのメリットを享受することができます。

このようにメリットの大きいモデルベース開発ですが、闇雲に取り組むだけでは以下のような問題が発生してしまいます。

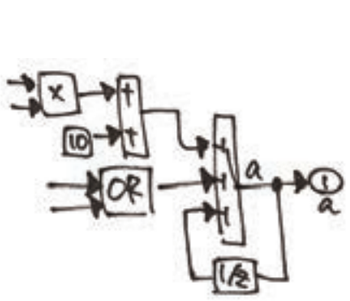
## 「動かすこと」を重視したモデルでは量産開発や以降の派生開発で問題が…

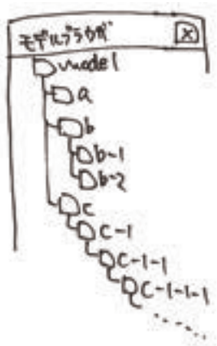
MATLAB/Simulinkで作成するのは実装を前提とした制御モデルです。一般的なV字プロセスに当てはめると「ソフトウェア詳細設計」で作られるモデルに相当します。そのため、それよりも前の段階で混入した問題については何の手当てもできません。

例えば、要求の段階でどんな問題が混入されていようと、それに応じたモデルを作成することしかできません。要求のヌケモレや矛盾、誤解釈があった場合には、それをそのまま引き継いだモデルが作られます。

また、Simulinkモデルは、先行開発においては制御仕様の正しさを確認するために効力を発揮します。しかし「動かすこと」を重視して作られているために、それまでの開発の経緯が残っていなかったり、モデルの構造についてはあまり注意を向けられていないようなケースもあります。

このようなモデルは、量産開発にそれを引き渡してもモデルの意図が汲み取れず理解するのが困難なため、以降の機能追加や派生開発まで考慮すると保守性の面で大きな問題となりえます。



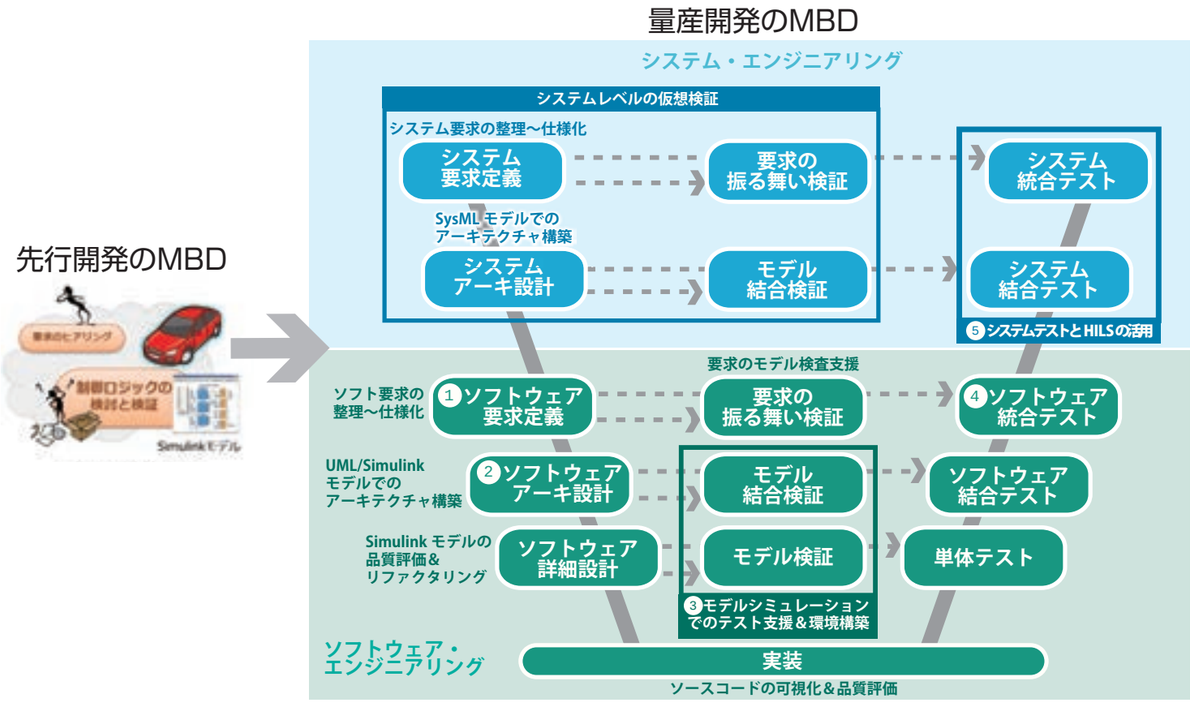

## 品質向上のための「品質開発」を機能開発と別に行う

これまで述べてきたように、これからは単に動くだけでなく、要求を満たして開発者にとって分かりやすいモデルを作ることが重要になってきます。

このような品質の高い MATLAB/Simulink モデルを開発するためには、先行開発で作られたモデルを元に「要求に合致した（＝妥当性のある）」「バグのない（＝動作検証済みの）」品質が担保されたモデルを量産開発向けに作成する必要があります。

そのためには、下図にあるように、機能開発だけに依存したこれまでの開発のやり方を変え、品質向上のための「品質開発」の活動を行うことが有効です。つまり、正しく「仕様化」した要求を量産品質の構造に「洗練」し、仕様の正しさと実装の正しさを「検証」していくことが必要です。

以降、下図の工程①～⑤の詳細を説明します。




## コンサルタントが教える成功の秘訣

ソフトウェアの設計手法には、オブジェクト指向だけでなく構造化設計などの手法も存在します。また、モデリング言語(表記方法)を考えても、一般的なフローチャート/状態遷移表/UML/SysML、ツール独自のSimulink/SCADE、車両制御固有のAUTOSAR/EAST-ADLなど、十分すぎるほど存在します。世間の動向やツールに翻弄され、導入以前より開発効率が低下した経験をお持ちの方もいらっしゃるのではないでしょうか。

開発対象や現場の置かれた状況により、「どの設計手法・モデリング言語を採用すべきか」の共通解は残念ながら存在しませんが、分かりやすくメンテナンスしやすい、バグのないソフトウェアを早く作るという目的は変わりません。

エクスモーションは、お客様の状況や抱える課題を把握し、数多くの選択肢の中から最適なソリューションを提案し、実践しています。

シニアコンサルタント
三輪　有史

## ①「要求の可視化/整頓」が品質開発の最初の一歩

Simulinkモデルの保守性を低下させる一因として「要求」が暗黙知であることがあげられます。この状況で開発されたモデルには「仕様」のみが書かれており、機能要求や品質要求、設計制約などが十分に反映されていません。

これを解決するためには、既存の「制御仕様」を「目的」「要求」「仕様（手段）」に分けて段階的に詳細化しながら形式化することが有効です。詳細は、7ページにある「要求の定義と仕様化」をご覧ください。このようにして作られた「ヌケモレのない要求と仕様」からは、動作検証のためのテストシナリオを作ります。こうすることで、ISO26262で重視される要求と実装のトレーサビリティも満たすことが出来ます。

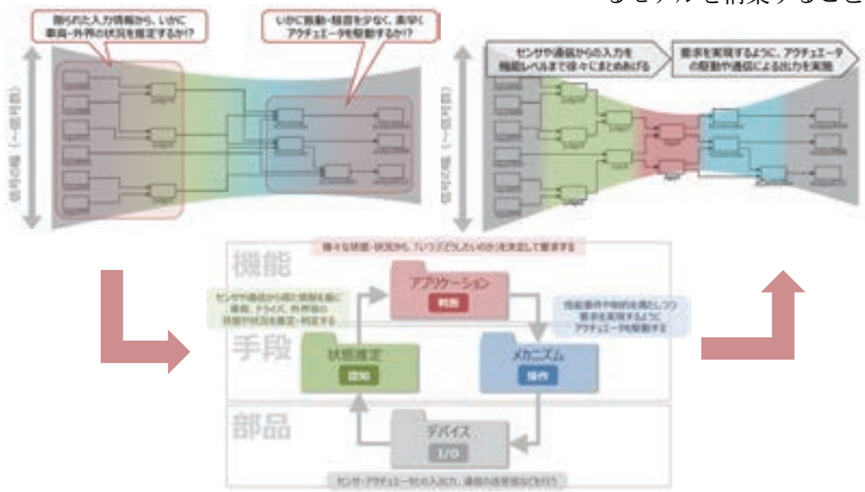
## ②「モデルの整頓」のポイントは「レイヤ」と「ドメイン」による分割

要素技術調査に心血を注ぎ、手段レベルのまま放置されたモデルは、情報をまとめきれず全体的に抽象度が低いため、要素間の入出力関係が複雑で下左図のように処理の複雑化を招きやすくなります。

モデルを整頓する上でうまくアーキテクチャを設計できると、要素間の入出力が簡潔で処理も簡潔になり、各要素の役割も明確で、理解しやすく変更しやすいといったメリットが得られます。

このアーキテクチャ設計のポイントは「レイヤ」と「ドメイン」による分割です。
車両制御ソフトウェアは、抽象度に応じて機能／手段／部品の３つの「レイヤ」に分割できます。

さらにドライバの運転機能に応じて、機能レイヤには「判断」、手段レイヤには「認知」と「操作」、そして部品レイヤには「デバイス」といった「ドメイン」を割り当てることで、互いに疎な構造からなるモデルを構築することが可能になります。




なお、開発対象の規模、最適な設計手法、開発者のスキル等に応じて、他のモデリング言語の併用も有効です。例えば「レイヤ」や「ドメイン」といったレベルでの設計にはUMLの活用が効果的です。

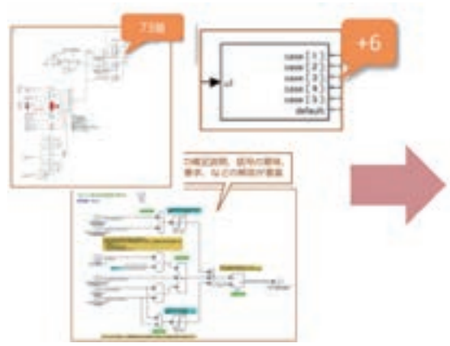
## ②うまく整頓できているか「モデルメトリクス」で確認

アーキテクチャレベルでモデルを整頓した後は、MATLAB/Simulinkモデルを再構築します。再構築がうまくできているかを確認する際に有効になるのがモデルのメトリクスです。基本的な考え方はコードメトリクスと同じで、ソースコードではなくモデルを静的に解析し、定量的な値を測定します。具体的には、サブシステム上のブロック数 、サブシステムの処理の複雑度を計測した経路複雑度、伝播信号の表示率、コメント率などがあります。

メトリクスはソフトウェアの品質と密接な関連性を持つため、メトリクスを評価することで品質上の問題が特定可能になります（次ページ上図）。

このように、開発したモデルに対し定期的にメトリクスを測定することにより、
・品質改善（問題箇所の特定）
・品質のモニタリング
（品質劣化防止、改善効果の可視化）
・品質のベンチマーキング
（品質基準の策定、複数モデルの品質を比較 ）
などを通じて、モデルの品質向上に向けた活動を行うことができます。

弊社では、 MATLAB/Simulinkモデルのメトリクス測定ツールを提供しています。詳細は41ページをご覧ください。




| メトリクス | 品質特性 | | | |
|---|---|---|---|---|
| | 解析性 | 変更性 | 試験性 | 成熟性 |
| サブシステム内のブロック数 | ✓ ブロック数が多すぎると読み解きづらい | | | |
| コメント率 | ✓ コメントがないと読み解きづらい | ✓ コメントがないと変更可否を判断しにくい | | ✓ コメントがないとモデルの妥当性を判断しにくい |
| 経路複雑度 | ✓ 分岐の数が多いと読み解きづらい | | ✓ 分岐の数が多いとテストケースが多くなる | |
| 伝播信号の表示 | | | ✓ 非表示だと接続の妥当性を確認しづらい | ✓ 非表示だと接続ミスに気付きにくい |
| モジュールの結合度 | | ✓ 結合度が高いと変更の影響範囲が広くなる | | |

## ③④「モデルの単体検証」と「統合テスト」で不具合を除去

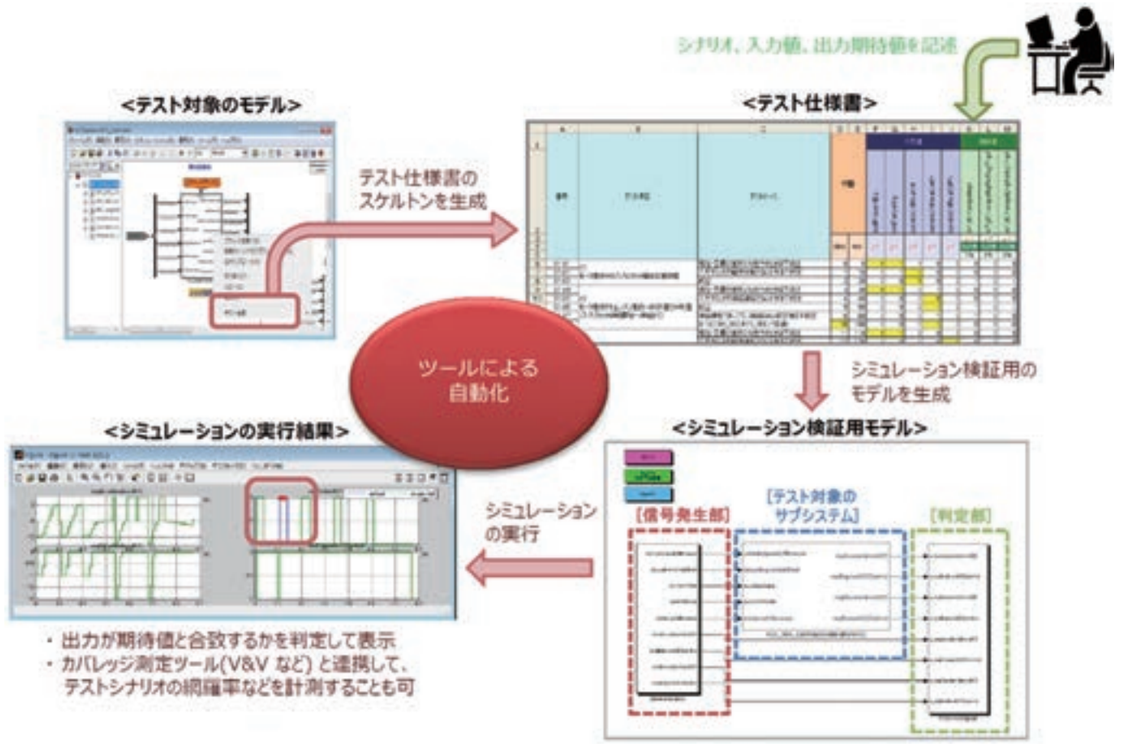
### ■シミュレーション検証の自動化がカギ

モデルを作成したら、その正しさを検証するために「モデルの単体検証」を行います。サブシステム単位でシミュレーション検証をすることにより、仕様の不具合を実装前に除去します。また「統合テスト」ではサブシステム間での協調・連携動作をシミュレーションし、要求を実現できているかどうかシステムレベルで検証します。

ここで気を付けなければならないのは、MBDではツール等を使った検証手段は充実しているのですが、場当たり的にそれらを使っているだけでは十分な成果物が残らないということです。また、テストベクタや報告書をその都度つくるのは非効率的で、ヌケモレも発生しやすくなります。

これらの問題は、MATLAB/Simulinkやカバレッジ計測ツールなどと連携する自動化ツールを導入することで解決できます。テスト仕様書やテストハーネス、テスト結果報告書などの自動生成が可能になることで、再帰テストが楽になり、十分なエビデンスを残すことができます。

次のページから、具体的な実施手順を説明します。



## ③「単体検証」でサブシステム単位でのシミュレーション検証を実施

### ■要求仕様書とモデルからテスト仕様書を作成

まず「単体検証」でサブシステムが仕様通りに動作するか検証します。その準備として、テストの内容を示した成果物となるテスト仕様書を作成します。

初めに、テスト対象となるモデルからテスト仕様書のテンプレートと入出力信号名のリストをExcel形式で自動的に生成します。

次に、テストシナリオごとのテストケースとテスト項目を作成します。テストシナリオは要求の仕様化で作成した「要求仕様書」がベースとなります（19ページの「要求の定義と仕様化」をご覧ください）。テストシナリオ別にシートを分け、テストケースとテスト項目、入力値と出力期待値を定義します。

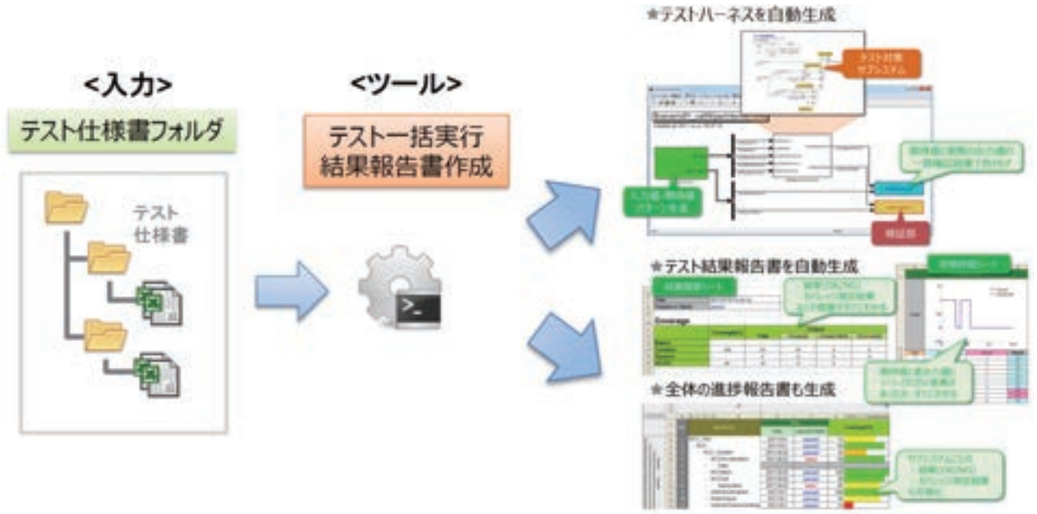
このようにテスト仕様書の入力に要求仕様書を利用することで、ISO26262で重視される要求と実装のトレーサビリティを満たすことができます。


### ■テストの一括実行とテスト結果報告書の自動生成

作成したテスト仕様書からテストハーネス（シミュレーション検証用のモデル）を生成してテストを一括で実行し、その結果をテスト結果報告書として自動生成します。

報告書では、テスト結果をサブシステム単位や要求仕様単位といった観点別に確認することができ、期待値と実出力値が合致しているかどうかを判定して表示します。カバレッジ測定ツールと連携すれば、モデル内やテストシナリオ単位での網羅率を計測し表示することも可能です。

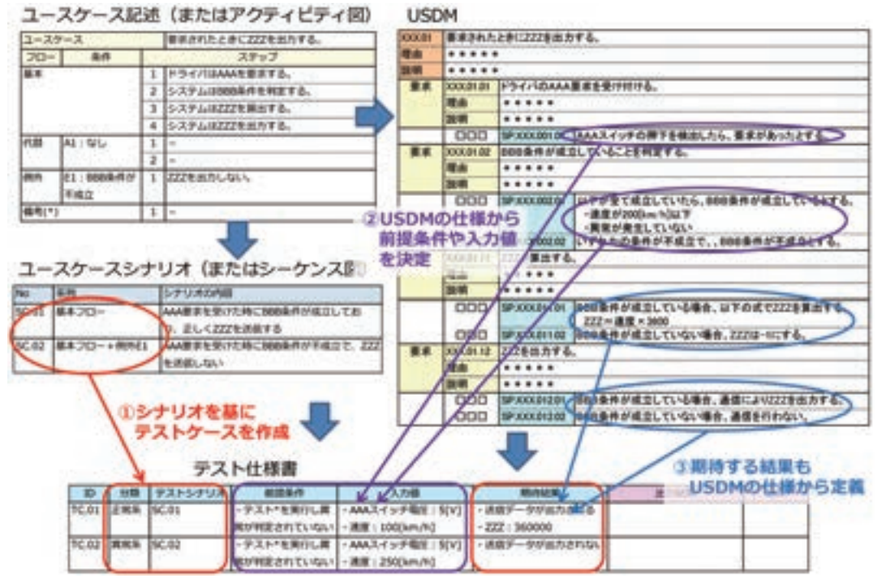
このような自動化によってミスなく効率的なテストが可能となり成果物も確実に残すことができます。




## ④「統合テスト」でサブシステム横断的なシミュレーション検証を実施

「単体検証」によってサブシステム単体での動作を確認した後は、サブシステム間での協調・連携が要求通りに行われているかどうか「統合テスト」を行い検証します。

作成したそれぞれのサブシステムから、他のサブシステムとの協調動作を行っている部分を抽出して結合し、「単体検証」と同様の手順でシミュレーションを行い結果を確認します。テストシナリオはシステム要求レベルとして、必要であればユースケースシナリオなども入力情報とします。

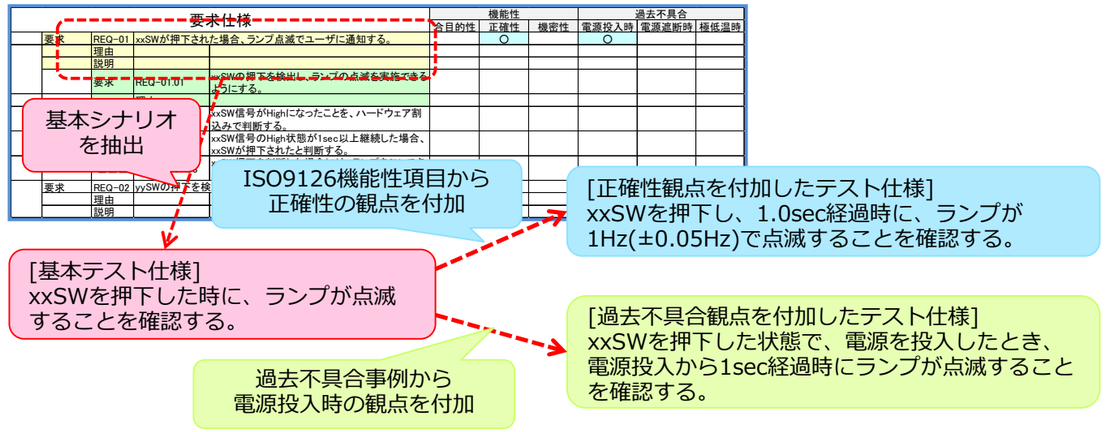



## ⑤システムテストとHILSの活用

### ■テスト観点に基づくテスト仕様の作成

シミュレーション検証の後は、ハードウェアとソフトウェアの結合テストやシステムテストを実施します。ここでのテストの目的は「システム要件が正しく実現されていること」の検証です。

テスト仕様の作成では、まず初めにシステム要件から基本となるテストシナリオを抽出します。このテストシナリオに対し、ISO9126で定義される品質特性や過去不具合事例などから導かれるテスト条件を付加することで、要定義段階では定義しきれない部分まで網羅的にテストを実施することが可能になります。また、このような観点を用いてテスト仕様の設計を行うことは、ISO26262で要求されるテストフェーズでの作業を実現することにもつながります。



## ■HILSの活用

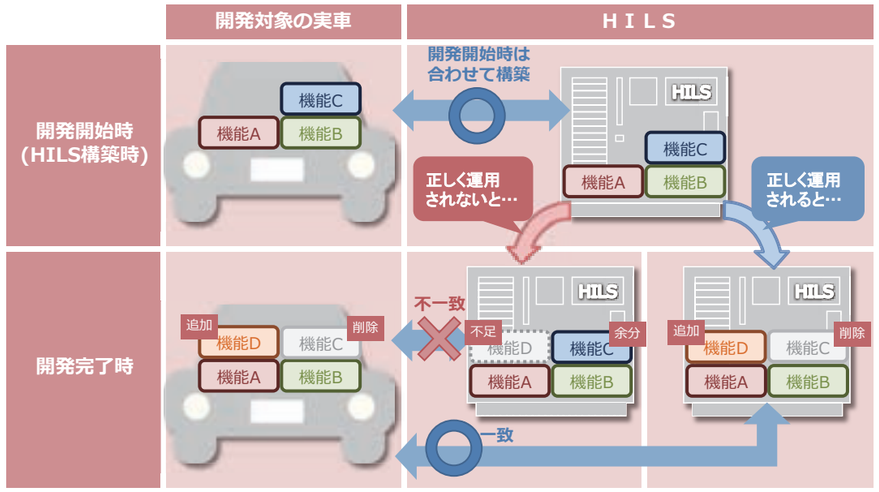
システムテストにおいては、実車両を用いてすべてのテストを実施することが理想ですが、車載ソフトの開発規模が増大している昨今、実車両ですべてのテストを実施することは困難です。

そこで実車両の振る舞いをシミュレーションできるHILSを利用します。HILSでは、さまざまなテスト条件をパラメータとして与えられるため、任意のテストパターンを容易に、かつ網羅的に実施することができます。

HILSの活用には次に説明する「運用プロセス」「自動テスト環境」がポイントになります。

## ■HILSの運用プロセスの確立

前述の通り多くのメリットが期待できるHILSですが、適切な運用ができている開発現場は多くありません。製品開発の過程では、要求や仕様が随時見直されます。しかし、HILSがその変化に追従できなければ、プロジェクト開始当初は機能を果たしていたとしても、いつしか使えないものになってしまいます。




このような開発製品との不整合を引き起こさないようにするには、HILSも製品機能の変更に合わせ、改修を重ねていく必要があります。そのためには、プロセスをつくり、そのプロセスに沿ってHILSを運用していける環境を整えなければなりません。

運用プロセスでは、改修が必要となったときには、改修に掛かる工数を見積もり、リソースを割り当て、システムの設計を行い、改修を加えていきます。このような体制を整えることにより、初めて、製品の開発プロジェクトと歩調を合わせた「使える」HILSが利用できるようになります。

## コンサルタントが教える成功の秘訣

HILSなど検証が上位になるにつれ、システムやコントロールユニットごとの独自の色が出てきます。例えばアナログな制御量が主出力となる制御ユニットと、デジタルな制御モードなどが主出力となる制御ユニットでは、効率的に検証結果を評価するための最適な仕組みは異なります。

またMILSを用いたV字工程の左側での上位検証と、HILSなどを用いたV字工程の右側での上位検証のどちらに重きを置くかについても、システムや開発体制などによってまちまちであると言えます。

エクスモーションではこれまでの様々な現場に対してMBD支援を行った実績を基に、お客様に最適な上流工程の検証プロセスおよび検証システムのご提案をいたします。

コンサルタント
松井　良太

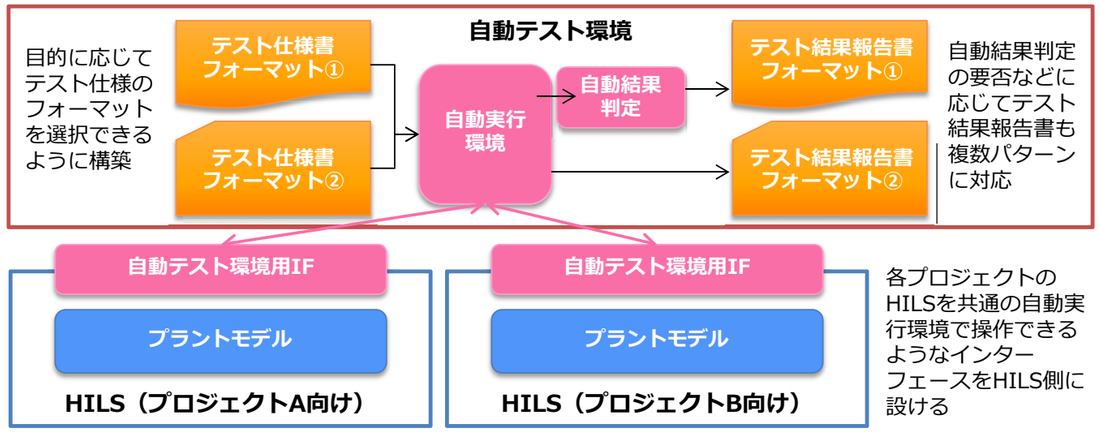
## ■自動テスト環境の構築

HILSにおいても、先に述べた単体検証と同様に自動テスト環境が有用ですが、その構築には工夫が必要になります。

例えば基本のテスト仕様に対してさまざまな車両状態を掛け合わせたテストを実施したい場合、単体検証のようなテスト仕様書ではなく、マトリクス形式でのテスト仕様書が効率的です。また検証結果についても自動での結果判定が適するものとそうでないものがあります。このように、テストの目的や特性に応じたフォーマットに対応できる環境を構築する必要があります。

さらに、プロジェクトごとに自動テスト環境を一から構築することは、コストや運用性の面でデメリットがあり、環境の移植容易性も考慮する必要があります。

このように、HILSの自動テスト環境については、実施したいテストの特性や将来的な展開を考慮して、設計、構築を行うことが肝心です。




## 「品質開発」には開発技術の知見と経験が必要、専門家に託すのも一つの考え方

新たな機能開発で多忙な中、品質確保のために時間や労力を割くことは困難です。また、品質開発では開発技術の知見と経験が必要であるため、自前で実施するには多くの時間を要します。市場で勝ち続ける製品を世に送り出し続けるために、品質開発を専門家に託すというのも一つの考え方と言えます。

## エクスモーションが提供するサービス

| | |
|---|---|
| MBDにおける品質開発のアウトソーシング | 量産用の開発では品質が担保されたモデルを開発しなくてはなりません。しかし、新たな機能開発で多忙な中、品質確保のために時間や労力を割くことは困難です。エクスモーションは、得意とする品質開発のための専門技術を使って、お客さまに代わりモデルの品質を確保します。 |
| Simulinkモデルのシミュレーション検証 | MATLAB/Simulinkやカバレッジ測定ツールなどと連携する自動化ツールを導入し、シミュレーション検証をより楽に開発に取り入れるお手伝いをします。 |
| Simulinkモデルリファクタリング支援 | 保守性が下がったSimukinkモデルの問題点を明らかにし、品質改善を目的としたリファクタリング（設計改善）の計画作成と実施、評価までを一括して請け負います。 |
| HILS活用支援 | 運用プロセスや自動テストシステムの構築など、HILSを効果的かつ継続的に利用できる仕組みを考え、「使える」HILSの環境づくりをご支援を致します。 |

www.exmotion.co.jp/　info@exmotion.co.jp　☎ 03-6722-5067

# モデルベース開発
# UML＋オブジェクト指向

1997年にUMLがOMGで採択されてから、19年が経ちました。組込みソフトウェアの開発現場にも、UMLによるモデル開発が適用され始めましたが、まだまだ定着したとまでは言えないようです。しかし、上流工程を重視するこの技術は、特に、新規でアーキテクチャを構築する時には欠かせないものです。

## 組込みソフトウェアには、丈夫で長持ちするアーキテクチャが必要

現在の組込みソフトウェア開発の現場は、多くの製品バリエーションと短い周期でのバージョンアップに追われています。一度アーキテクチャを作れば、短くとも5年、長ければ20年近くもそのアーキテクチャを使い続けます。

そのため、新規にソフトウェアを構築する時には、いかに丈夫で長持ちするアーキテクチャを構築するか、という点が、重要なポイントとなります。

では、丈夫で長持ちするアーキテクチャとは、どのように作ればよいのでしょうか？

## 抽象化により、丈夫で長持ちするアーキテクチャを

みなさんは、UMLやオブジェクト指向という技術について、どのような印象をお持ちでしょうか？難しいといった声も聞こえてきますが、実は、UMLやオブジェクト指向自体はそれほど難しくありません。それよりもむしろ、難しいのは対象となるシステムの知識を整理することです。

UMLやオブジェクト指向は、その整理したものに対して、整頓するための観点を与えたり、その結果を可視化して、レビューし易くする、といった手助けをしてくれます。

特にオブジェクト指向で強力なのは『抽象化』という概念です。

オブジェクト指向における抽象化とは、物事の大事な部分だけを取り出して、本質的な構造や振舞いをモデル化する、ということです。そうすることにより、時間が変化しても変わらない普遍的なものと、そうでないものが明確となり、時間軸での劣化を防ぐことが可能となります。そのため、長持ちするアーキテクチャを構築するのに向いている、ということが言えます。

## コンサルタントが教える成功の秘訣

オブジェクト指向によるモデル開発では、「抽象化」は非常に有効な概念です。ですが、その抽象化が逆に問題になることもあります。私がこれまで関わったお客様の中には、抽象化されたモデルを、なかなか理解できない方がいらっしゃいました。そのような状況において、抽象化はお客様にとって、むしろ足かせになりかねません。

エクスモーションでは、お客様の目的や理解度に合わせて、どこまで抽象化すればよいか、お客様に確認しながら進めていきます。例えば、「このような変更に対応できるようにしたいので、ここまで拡張性を設けます」といったように、そのような構造になった背景も合わせて解説することも大事です。お客様が納得して、その後の作業もスムーズに取り組めるようなモデルづくりを、エクスモーションがサポートいたします。

シニアコンサルタント
八坂 浩司

## なぜ、開発現場に定着するのが難しいのか？

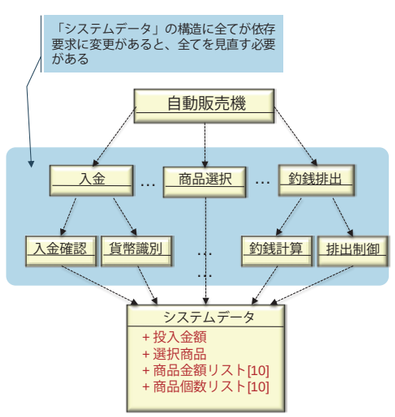
前述にもある通り、UMLが正式に標準化されて19年もの歳月が流れていますが、開発現場に定着しているとは言えません。なぜ、このような状況に陥っているのでしょうか？

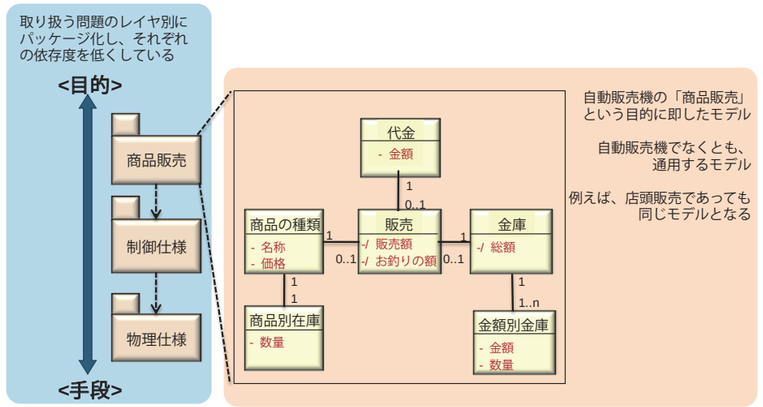
### ■初心者が自己流で超えられない壁

長年、手続き指向で開発を続けていた人が、いざ、オブジェクト指向で開発しようと思っても、なかなかその“手続き的な発想”から切り換えることができません。

例えば、下図の左側のモデルは、“手続き的な発想”のまま作られたクラス図です。一方、右側のモデルは、オブジェクト指向的な発想にのっとって作ったモデルです。この二つのモデルの違いは、時間経過により変わるものと変わらないものを区別して分離しているかどうか、という点です。組込みシステムでは”ハードウェアや制御方法”は変化しやすく、その”システムが達成するミッション”は、あまり変化しません。自動販売機で言うと、“商品販売”という仕事がそれにあたります。

しかし、なかなかここにフォーカスしてモデルを作るのは難しいようです。




### ■良いお手本が現場にない

これは、よい“お手本”が開発現場に存在しないことが一つの原因と言えます。組込みの開発現場は、開発そのものが企業独自のノウハウであり、競争力の源泉となっています。そのため、いくらいいモデルができたとしても、外部には公開しません。その結果、たまたま良いモデルを作れた企業はどんどん良い方向へ、そうでない企業は、モデルから離れていく、といった二極化が起こっているのです。

### ■良いお手本を使って人材を育成

「良いお手本を見て理解を深める」「開発の実践以外にモデルを書く機会を増やす」「繰り返しモデルを書く」ことが、モデリングスキルを向上させるために必要です。そのようにしてスキルのある人材を増やすことが、開発現場への定着につながるのです。

## コンサルタントが教える成功の秘訣

ものごとには、複数の見方があります。例えば、円柱も四角柱も、横から見ればどちらも長方形ですが、上から見ると、円と正方形の全く違う形に見えます。分析とは、ものごとが持っている複数の見え方の、どの側面にフォーカスしてみていくか、ということを決めることです。ものごとをどの側面で見るかによって、その解法は当然違ってきます。「どの側面から見ると、どのような解法になり、それは適切なのか」といったように、複数の観点を比較しながら、最良の分析を見つけていく、そこが難しくも、面白いところです。

エクスモーションのコンサルタントは、まさに、このような分析作業を得意としています。モデリングの良いお手本づくりに、この強みを活用していただければ、時間的・費用的に十分メリットのあるモデルベース開発を実現できると確信しています。

エグゼクティブコンサルタント
井山 幸次

# レガシー救済プロジェクト
## ～悪しき習慣を断ち切れ！～

**近年の組込みシステム開発では、数十年前に構築したシステムを元に製品を構築する『レガシー開発』が主流となっています。しかし、長年にわたり追加と修正をし続けた結果、設計は忘れ去られ、コードは疲弊しています。そして、チームに蔓延するマイナス思考…しかし、そんなプロジェクトにも救済の糸口があります。**

## 悪しき習慣が組織文化となったレガシー開発をどのように救済するか？

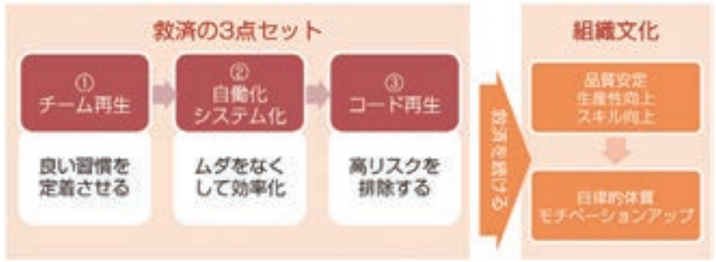
十年以上前に構築されたシステムを元に、製品開発を続けている『レガシー開発』では、大きな問題をかかえているものの、当事者は何とも感じていない、ということがありがちです。それは、問題が常態化し、組織文化として根付いてしまっていることに他なりません。それをどのように救い出すか、ここでその概要について説明します。

### ■救済の3点セット

問題のあるレガシー開発の現場では、品質や生産性の低下が問題として取り上げられますが、その根幹には、開発者のスキルが低いことや、コミュニケーション不足、モチベーションが低いなどの問題が存在しています。それが常態化することで組織文化として定着し、解決はますます困難となります。

まずは「チーム再生」でチーム内の風通しを良くします。これがすべてのベースとなります。その後、「自働化・システム化」を行います。それでようやく「コード再生」に手を付けることができるのです。

組織文化として定着したものを変えるのは容易なことではありません。救済するためにすべきことは、右図にある通り「チーム再生」「自働化・システム化」「コード再生」ですが、全てを一気に進めるのではなく、状況を見ながら、段階的に進める必要があります。




## ①チーム再生は「共有・ルール化・学習」の習慣付け

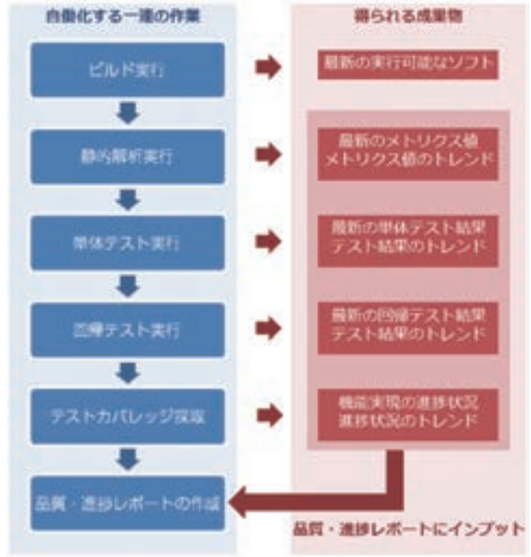
救済の第一段階である「チーム再生」では、生産性と品質向上のための活動のベースを構築するために、右図のような活動をします。

チーム内で主体的にコミュニケーションが取られ、守るべき最低限のルールが守られ、ある程度のスキルアップがはかられると、チーム再生における次のステップである②自働化・システム化の活動に移ることができます。


## ②自働化・システム化は「テスト駆動とCIの導入」で活動を日単位に

### ■テスト駆動開発を導入する

テスト駆動開発の導入は、品質を安定させるための第一歩となります。ここでのポイントは、「ペアプログラミング」「小規模リファクタリング」「レビュー」を織り込みながら実施するという点です。これは、チームのコミュニケーションがベースとして作られ、ルールという共通認識がなければ、成り立たちません。




### ■CIでビルド以降の作業を自働化し共有化する

テスト駆動開発の導入と並行して、ビルド以降の一連の作業（右図）を自働化し、その結果を関係者で即座に共有できるようなシステムを導入します。これは、一般的には「継続的インテグレーション（CI：Continuous Integration）」と呼ばれます。継続的インテグレーションを実現するシステムを導入することで、自働化・システム化し、ムダを排除して、更に品質安定化のベースを作り上げます。

ビルドの結果、得られる情報はシステムを通じて、プロジェクト関係者全員が即座に共有することができます。

それを実現するのが、右図のようなシステムです。

CIのフレームワークであるJenkinsに、単体テスト・回帰テスト・静的解析・カバレッジツールを組込み、ビルドからの一連の作業を実行させます。




## コンサルタントが教える成功の秘訣

多くのレガシーコードは、ハードコーディング、グローバル変数、コメントなし、複雑な依存関係、巨大な関数やコンポーネントなど、様々なアンチパターンから形成されています。このようなコードは、開発費の増大、開発速度のスローダウン、出荷後のバグ多発など、多くの問題を頻発させます。

「そういったコードは、リファクタリングすればいいじゃないか」と聞こえてきそうです。その通り、リファクタリングすればいいのです。ただし、何年にもわたってアンチパターンが積み重なったコードのリファクタリングには、非常に高度な知識、経験と粘り強さが必要です。簡単に説明できる成功の秘訣などありません。それでもあえて言うとすれば、「リファクタリングのプロを探せ」です。

私と一緒に、レガシーコードを蘇らせましょう。

シニアコンサルタント
井上 一郎

## ■品質と進捗を毎日確認する


右図は、Jenkinsのダッシュボードの画面例です。この例では、ビルド履歴、単体テスト・静的解析のトレンドを見ることができます。これらの情報は、ビルドをする度に自動的に更新され、常に最新の情報を得ることができます。

このようにすることで、品質のデグレードや開発の遅れを素早くキャッチし、早い段階で適切な対応を実施することができます。

ここまでくれば、救済が必要だったレガシー開発は、品質と進捗が制御可能な状態となり、これ以上悪い状態に陥ることはありません。誰もが安心してプロジェクトを進めることができます。

## ③コード再生は「問題の可視化」で計画的に行い「回帰テスト」で互換性を保証

## ■コードの可視化で問題を発見

コード再生を開始する時には、“開発者の主観に頼る”ではなく“客観的な指標を使う”必要があります。

右図の「ツリーマップ」は、コードの問題を可視化する「静的解析」の一つの方法です。モジュールのサイズを面積の広さで、複雑度を色で示しています（白→黄→赤の順で複雑度が高くなる）。その他にも様ざまな方法があります。一つの方法ではなく複数の方法で問題をあぶりだしておくことが、その後のプロセスをスムーズに進めるためのポイントです。

これらの情報を元に、コード再生の計画を作成し、要員をアサインして、リファクタリングすることでコードを再生していきます。

## コンサルタントが教える成功の秘訣

品質が低下したレガシーコードの保守に頭を悩ませる開発現場のエンジニアは、みなさまざまな問題認識を持っています。問題を改善していくためには、現場での困りごとを発生させる要因となっているソフトウェア設計上の問題を、客観的事実による裏付けとともに明らかにしなければ、大きな改善効果は望めないのではないでしょうか。

レガシーリファクタリングのサービスでは、ソースコードの調査と現場エンジニアへのヒアリングを通し、設計の問題点を把握することからスタートし、ソフトウェア構造のあるべき姿を設計する支援や、リファクタリングを進めるためのプロセス設計・運用の支援まで、ソースコードの改善をトータルに支援いたします。

レガシーコードの品質にお困りの方は、ぜひ私たちにご相談ください。

スペシャリスト
玉木　淳治

## ■before/afterを保証する回帰テスト

リファクタリングにおいて、何より大事なことは、リファクタリング対象のモージュルの互換性を保証することです。そのためには、回帰テストを実施しなくてはなりません。

回帰テストとは、リファクタリング対象のモジュールに対して、before/afterで振舞いが変わっていないことを確認するためのテストです。

まず、リファクタリングを実施する前のモジュールの振る舞いを「仕様」として確認するためのテストプログラムを作成します。そのテスト結果が、リファクタリング後のモジュールで確認すべき仕様です。

リファクタリング終了後は、同じテストプログラムを通して、テスト結果が一致することを確認します。

下図はGoogle testフレームワークを使ったテストプログラムの例です。テストフレームワークでは、フレームワークが用意している仕組みを使って、単体テストを簡単に作成することができます。下図のように、フレームワークを使うことで、テスト結果まで分かりやすく表示してくれます。

これらの「リファクタリング→回帰テストでの確認」の流れは、CIフレームワークに組み込むことで、一日毎の作業として完結することができます。




長年にわたる流用によりブラックボックス化してしまったレガシーソフト。かつては、副作用が怖くて手を入れることができなかったプロジェクトも多いのではないでしょうか。今回紹介した、テスト駆動開発、CIフレームワーク、テストフレームワークを活用することで、安全で効率的なプロジェクト改善やリファクタリングを行うことができます。皆さんも、これを機に始めてみてはいかがでしょうか。

## エクスモーションが提供するサービス

| | |
|---|---|
| レガシー救済診断 | レガシー資産や開発スタイルの良し悪しを、定量・定性の両面で、経験豊富なコンサルタントが診断し、救済への道筋を提示します。 |
| レガシーレスキュー隊派遣 | レガシー開発を救済するためのプロジェクトを主導していく、通称『レスキュー隊』を派遣します。経験豊富なコンサルタントが、救済プロジェクトを主導していくとともに、ソフトウェア開発技術の知見と経験を併せ持つレスキュー隊が、実際に手を動かして救済を実行します。 |
| ソースコード診断 | メトリクス解析やアーキテクチャ構造解析などによって、コードの品質を多面的に診断し、問題点の抽出と改善へのアドバイスを行います。 |
| リファクタリング請負 | 低レベルから高レベルまで、お客様の状況・目的に応じてリファクタリングを請け負います。 |

www.exmotion.co.jp/　info@exmotion.co.jp　☎ 03-6722-5067

# 機能安全対応

**機能安全規格ISO26262への準拠に向けて、自動車メーカーや部品メーカーでの対応も佳境を迎えています。製品の開発と並行して機能安全規格対応を行うにはどうしたらよいのでしょうか？これまでの弊社の経験をもとに説明します。**

## 機能安全規格ISO26262とは？

安全であるとは、製品が使用される環境において受容できないリスクがない状態のことです。機能安全とは、機能的な工夫を付加することによってこの安全を確保することをいいます。例えば、システムを構成する部品に故障が発生した場合でもそれを検出して危害を回避、軽減するための工夫を導入することで安全な状態を確保できます。

ISO26262は自動車分野向けに策定された機能安全規格で、自動車の電気/電子システムにどのような安全機能を組み込むか、その安全機能を実現するためにどのような手順で開発や管理を行うべきかを体系化したものです。

つまり、ISO26262に準拠することにより、自社の製品が安全面を十分に考慮して開発されたことや、自社に機能安全を考慮した開発が行える能力があることを示すことができます。

## 派生開発をしながら機能安全対応をするポイント

ISO26262で示されているプロセスはトップダウンの流れで説明されており、新規開発の場合にわかりやすい内容となっています。

しかし、実際の開発のほとんどが既存システムをベースとした派生開発であるため、現実的にはシステムへの新たな機能の追加、既存機能の変更、性能向上を行うのと同時に、機能安全規格に対応するためには、既存の仕様で規格の要件を満たせるのか、要件を満たすように改善する必要があるのかを見極めなければなりません。派生開発と規格対応を同時に行うのは非常に苦労するので、まずは既存のどの成果物が規格で求められる成果物となるのかを把握することが重要です。

以降のページでは、ISO26262 Part3とPart4のプロセスと既存システムの仕様との関係、および規格対応時に改善が必要と思われるポイントについて、実例を用いて解説します。

## コンサルタントが教える成功の秘訣

機能安全というと、特別厳密な作り方や基準が求められているように思われる方も多いのではないでしょうか。しかし、実際には、設計根拠の論理的な説明が必要だったり、トレーサビリティをきちんと取ったりなど、一般の開発においても要求されていることが多くあります。このような基本的なことをきちんと実施することは、組織あるいは個人のスキルアップの機会になる、と前向きに捉えて、チャレンジしていただきたいと思っています。

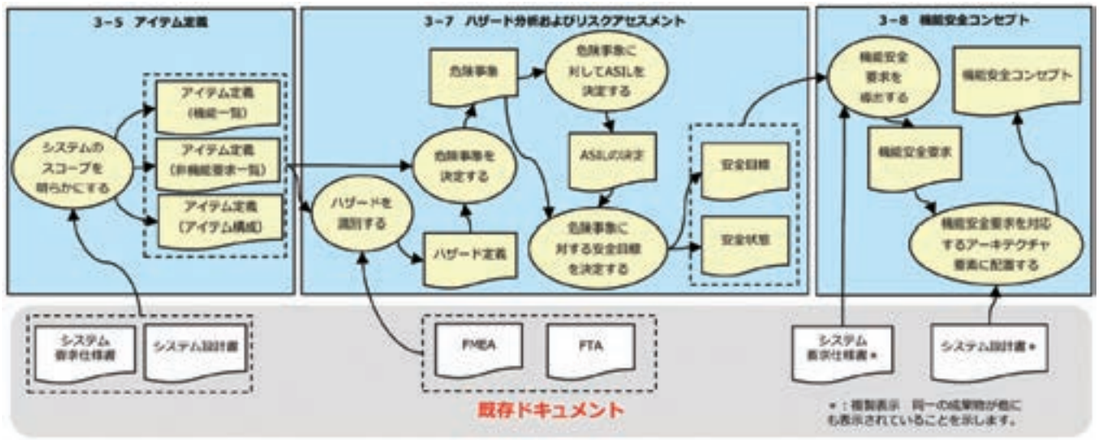
エクスモーションでは、これまでさまざまな技術の導入と支援を実施してきました。その中で、機能安全にも生かせる実践的なノウハウを数多く持っています。また、現場ごとの状況に合わせた提案も得意としております。

ぜひ、私たちにお任せください。

シニアコンサルタント
小坂 優

## Part3：コンセプトフェーズ

コンセプトフェーズでは、アイテム(車両レベルの機能)を定義し、ハザードの識別や発生しうる危害を想定し、安全な状態を維持するための方針を決定します。

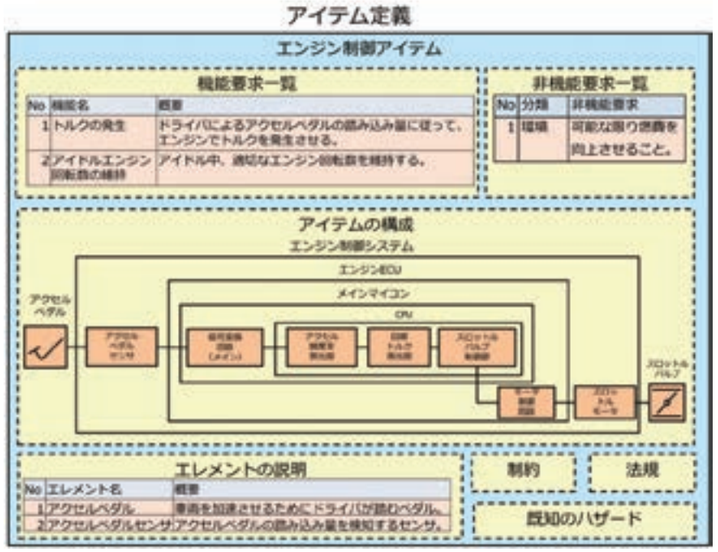



### ■アイテム定義

アイテム定義の工程ではアイテムのスコープを明らかにし、機能や制約、およびアイテムの構成(初期のアーキテクチャ)を定義します。

つまり、既存システムのシステム要求仕様書とシステム設計書をもとにアイテム定義としてまとめれば良いということです。

ただし、アイテム定義では車両レベルの機能を対象としますが、実際には既存システムがECU単位で定義されている場合が多いと思います。その場合には、複数ECUの仕様から必要な部分を抽出し、車両レベルの仕様として定義する必要があります。

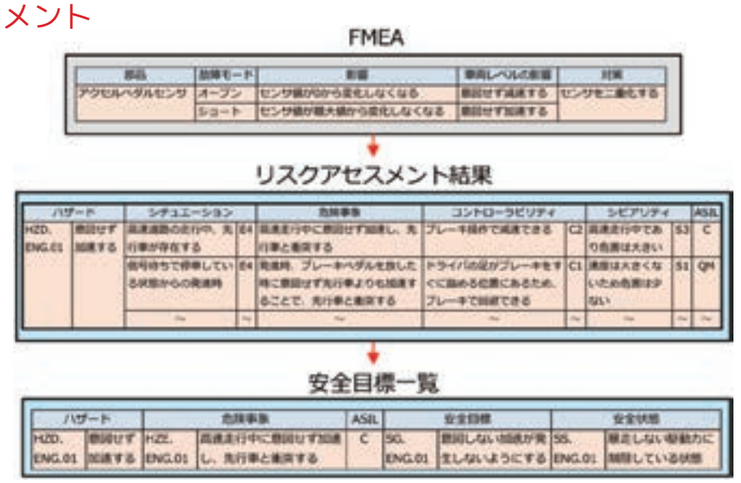



### ■ハザード分析＆リスクアセスメント

ハザード分析＆リスクアセスメントでは危険事象を特定し、回避・軽減するための安全目標を定義します。

ハザードは改めて分析しても良いのですが、既存システムのFTAやFMEAで分析した故障の影響をハザードとして使用できます。そして、そのハザードに対してリスクアセスメントを実施し、ASILを決定します。

ただし、既存システムでは、故障の影響が車両レベルまで分析はされていないことも多く、追加の分析が必要となる場合があります。



## ■機能安全コンセプト

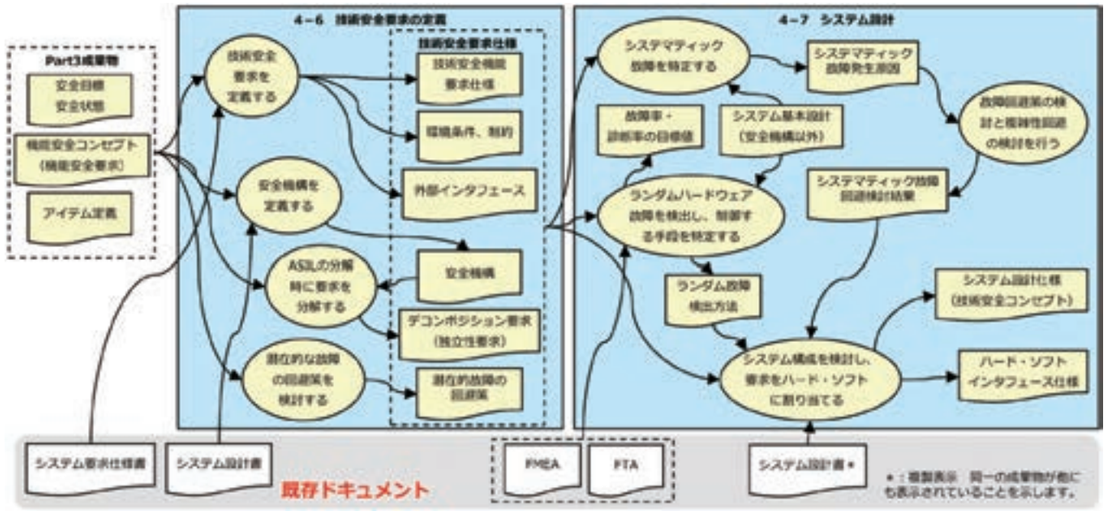
機能安全コンセプトの工程では、安全目標の実現に必要な機能安全要求を導出し、機能安全要求をアイテムのどの構成要素に配置するか定義します。

ISO26262はトップダウンの開発プロセスですが、既存システムのフェールセーフ仕様が安全目標を満たすなら、その仕様からどのように危険事象を回避・軽減するか、どのような警告をするかを整理することにより、ボトムアップのアプローチで機能安全要求を定義することができます。

既存システムのフェールセーフ仕様が安全目標を満たさないのであれば、新たなフォールト検出方法や警告方法などを検討する必要があります。

## Part4：システムレベルの製品開発

システムレベルの開発では、機能安全要求を技術安全要求へと具体化し、機能安全以外の本来の機能と合わせてシステム設計を行います。




## ■技術安全要求の定義

技術安全要求の定義では、実際の物理的なアーキテクチャで機能安全要求を実現することを想定して、機能安全要求を技術的な要求へ具体化します。さらに、技術安全要求をハードウェアで実装するのか、あるいはソフトウェアで実装するのかを特定します。

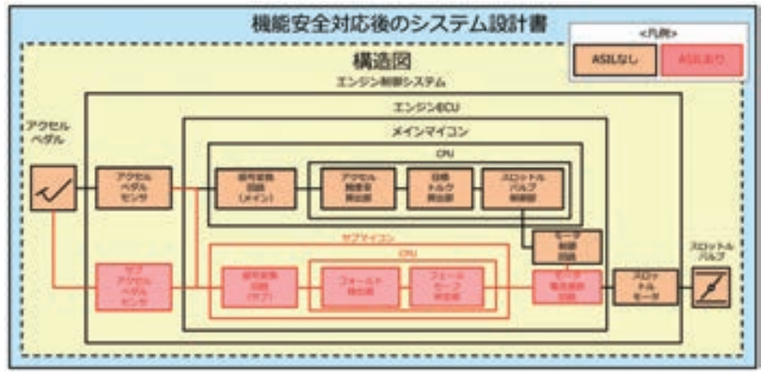
技術安全要求も機能安全要求と同様に、既存システムのフェールセーフ仕様を利用して、ハードウェア・ソフトウェアのどのような機構で安全を確保しているかを整理することにより、技術安全要求を定義することができます。

| 技術安全要求仕様 | | | | 配置するエレメント |
|---|---|---|---|---|
| 機能安全要求 | TSR.ENG.01.01 | | ドライバのアクセルペダルの踏み込み量を正しく取得できなくなるようなフォールトを検出する。 | – |
| | ASIL | | C | |
| | 技術安全要求 | TSR.ENG.01 | 2つのアクセルペダルセンサで踏み込み量を計測し、両者の値を比較することでフォールトを検出する。 | – |
| | | ASIL | C | |
| | | 理由 | アクセルペダルセンサのセンサ値が異常な値になるようなフォールトを検出するため。 | |
| | | 説明 | センサにフォールトが発生すると、信号が入力されなかったり、実際の踏み込み量とは無関係の値が入力されたりする。 | |
| | <H/Wへの要求：アクセルペダル踏み込み量の計測> | | | |
| | サブアクセルペダルセンサの組織 | 技術安全要求 | TSR.ENG.01.01 | システムに、メインのアクセルペダルセンサとは別にサブのアクセルペダルセンサを搭載する。 | ・エンジンECU<br>・サブアクセルセンサ |
| | | | ASIL | C | |
| | メインアクセルペダルセンサによるアクセルペダル踏込量取得 | 技術安全要求 | TSR.ENG.01.02 | メインアクセルペダルセンサで、ドライバによるアクセルペダルの踏み込み量を計測する。 | ・メインアクセルセンサ |
| | | | ASIL | A (C) | |
| | サブアクセルペダルセンサによるアクセルペダル踏込量取得 | 技術安全要求 | TSR.ENG.01.03 | サブアクセルペダルセンサで、ドライバによるアクセルペダルの踏み込み量を計測する。 | ・サブアクセルセンサ |
| | | | ASIL | B (C) | |
| | <H/Wへの要求：デジタルデータへの変換> | | | |
| | メインアクセルペダルセンサ値のデジタルデータ変換 | 技術安全要求 | TSR.ENG.01.11 | メインアクセルペダルセンサの値をデジタルデータに変換する。 | ・信号変換回路（メイン） |
| | | | ASIL | C | |
| | サブアクセルペダルセンサ値のデジタルデータ変換 | 技術安全要求 | TSR.ENG.01.12 | サブアクセルペダルセンサの値をデジタルデータに変換する。 | ・信号変換回路（サブ） |
| | | | ASIL | C | |

## ■システム設計

システム設計では、技術安全要求のシステム構成要素（ハードウェアまたはソフトウェア）への配置を定義します。ここでは単純に技術安全要求を配置するだけでなく、安全機構を本来の機能とは独立して配置することで、ASIL対応に開発コストをかける箇所の局所化を検討します。

今回のサンプルではアクセルペダルセンサを二重化し、センサの故障時にモータ電流を遮断する安全機構を定義しています。更に、本来の機能で故障が起きても安全機構に波及しないよう、安全機構を独立したマイコンに配置し、ASILが求められる箇所をサブマイコンに限定しています。




## トレーサビリティの構築

ISO26262では上流から下流の成果物までトレーサビリティを構築することが求められるので、ツールを用いてアイテム定義からシステム設計、それ以降の下流工程の成果物までを関連付けて管理します。

右図はその一例です。エンジン制御アイテムで定義した成果物内の要素を個別にツールへ登録し関連付けています。システムレベルの製品開発で定義した安全機構であるサブアクセルペダルセンサには Part5のハードウェア設計やPart6のソフトウェア設計で作成した成果物の要素などが関連付けられます。


## 機能安全対応・派生開発をうまく乗り切るために

派生開発で機能安全対応を行うには、全てを一から作成するのではなく、既存システムの成果物を、機能安全プロセスとの関係を把握した上で活用することが重要です。しかし、現場では制御仕様書やソースコードなど下流工程の成果物はあるものの、システム視点での要求やアーキテクチャといった上流工程の成果物が作成・整備されていないことも多いと思います。

各種仕様書がトレース可能な状態で整備されていると、機能安全対応に限らず、派生開発で仕様の追加・変更を正確に、かつ効率的に行えます。

規格で求められているからという理由ではなく、開発の品質を高める目的で取り組み、仕様書が開発組織で共有できる状態を維持していただきたいと思います。

# 既存資産の解説書
## ～埋もれた知識の共有化～

**混沌とした開発現場では、実にさまざまな問題が起こりますが、その多くは『使えるドキュメント』がないことに起因します。開発者は開発プロセスに従って、必要なドキュメントを作りますが、その後、誰も見ぬまま情報が古くなり、『使えないドキュメント』になってしまいます。なぜなのでしょうか？**

## 「使えるドキュメント」がないことが引き起こしている問題

組込みソフトウェアの開発現場では、生産性や品質の問題が日々深刻さを増しています。その原因はいろいろと考えられますが、『使えるドキュメント』がないことが原因の一つであることは、想像に難くありません。ここでは、その典型的な問題が起こっている派生開発を例に、どんな問題があるのかを説明します。

派生開発では、最初の製品を開発した時に、ほとんどの大きな課題を検討し、その結果としてアーキテクチャが構築されます。しかし、そこで残されるのは、検討のために開かれた会議の議事録（あれば上等）と、検討の結果構築されたアーキテクチャとその説明だけとなってしまいます。その後の派生開発においては、最初の製品を開発した時と同じメンバ構成で行われるのは稀で、メンバの多くは入れ替わります。投入された新しいメンバには、検討の結果構築されたアーキテクチャドキュメントや設計ドキュメント（それらはすでにコードと乖離している…）と、ソースコードだけが与えられます。その人達は、与えられたものを元に、現行の仕様を理解しようと試みますが、結果的に理解することは難しく、時間の制約の中で場当たり的な対応を繰り返してしまいます。その結果、いつの間にか保守性は著しく悪化し、生産性の低下につながってしまいます。

以下に『使えるドキュメント』がない開発現場でよく見られる現象をピックアップしてみました。

皆さんの開発現場は、いくつ当てはまりますか？

- ☑ レビューのたびに使い捨ての説明用資料を作っている
- ☑ 日々の開発業務で精一杯で、ドキュメントを作成する工数が割けない
- ☑ ソフトの肥大化・複雑化により、概要を把握しづらくなってきた
- ☑ ドキュメントとして何を残せばいいか良くわからない
- ☑ 作成したドキュメントが、ほとんど参照されない
- ☑ ドキュメントの費用対効果(作成・保守コストvs.役立ち度)が低い
- ☑ 先行開発から量産開発への移行を控えているが、成果物がモデル/コードしか存在せず、引き継ぎできない
- ☑ ドキュメントを作るべき人が作ってくれない
- ☑ ドキュメントのフォーマットや記載レベル、記載範囲がバラバラ

## なぜ「使えるドキュメント」は開発現場にないのか？

では、なぜ「使えるドキュメント」が開発現場にないのでしょうか？その原因は、次の２点に集約されます。

・開発工数・優先順位の問題

・ドキュメントを作成する人のスキルの問題

### ■開発工数・優先順位の問題

近年のような激しい競争が強いられる経営環境においては、いち早く多くの市場を獲得することが最優先の経営課題となっています。そのため、「どこよりも早く製品をリリースする」ために、開発者は何よりも、製品自体の作り込みを優先させられます。

結果として、その時の製品をリリースするための最低限のドキュメントを作るだけで精一杯になります。それには、合意した「結果」だけが書かれ、その根拠や背景については触れられません。つまり、リリース後の派生開発で本当に必要とされる知識はドキュメント化されず、担当者の頭の中に埋もれてしまい、いつしか忘れ去られてしまうのです。

では、ただ時間があれば、「使えるドキュメント」が書けるのでしょうか？

### ■ドキュメントを作成する人のスキルの問題

「使えるドキュメント」を作成するためには、重要な情報を見極め、分かりやすく伝わる形にする必要があります。そのためには、抽象化能力・論理的思考力・図解力・文章力などのスキルが求められます。それらのスキルは、プログラミングや要素開発などの開発スキルとは異なり、一般的には開発者が苦手としている分野です。そのため、たとえ育成したとしても、満足できるレベルまで到達するためには、かなりの期間と費用を必要とします。

このように、「使えるドキュメント」を作成するためには、ここで挙げた２つの問題に取り組む必要がありますが、開発者をそのまま「使えるドキュメント」の作成者にアサインすることは、現実的な解とは言えません。

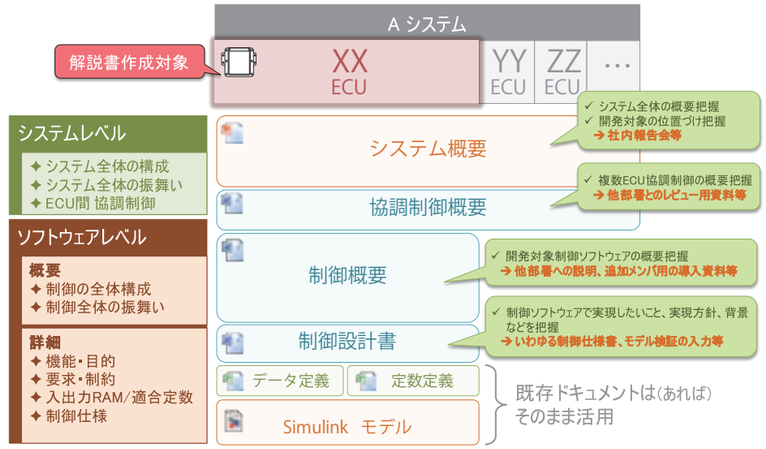
## 「解説書」とは何か？

エクスモーションでは、ここで言う「使えるドキュメント」のことを「解説書」と呼んでいます。「解説書」とは、開発対象にとって重要なポイントを分かりやすく伝えるドキュメントの総称です。

解説書は、その目的や読者により、複数のドキュメントで構成されます。右図はその一例です。

解説書全体として記載する内容は、ユーザーズマニュアルレベルの要求仕様と、その実現手段であるソフトウェア構造やソースコードとを段階的に結びつけるものであり、開発対象によらず共通です。しかし、ドキュメントの構成、各ドキュメントの記載内容や記載方法は、それぞれ異なります。

「解説書」に書かれる内容は、その製品の特徴であったり、背景となっている仕組みの解説が中心となります。それらは、一度作成すれば、長い間参照されるものとなります。そのため、一度作ることに大きな意味があるのです。




## コンサルタントが教える成功の秘訣

現場で必要とされている解説書は、開発対象や現場が置かれている状況などによりまちまちです。そのため、本当に「使える」解説書は、機械的なやり方で導出することは難しく、現場の声を吸い上げて「仕立て」ていく必要があります。つまり、良い解説書を作るためには、お客様の本質的な要望を理解し、開発対象の何が重要かを見極め、それを分かりやすく伝えるスキルが必要となります。

エクスモーションでは、お客様のご要望をお聞きし、それに合った表現方法や進め方をコンサルタントが見極めて、ご提案いたします。また、作成中は、担当者の方に繰り返しレビューして頂きながら、内容の正確さや分かりやすさを高めていきます。そのようなプロセスを経て作り上げる解説書は、どのような開発対象であっても、きっとご満足いただけるものと自負しております。

エグゼクティブコンサルタント
小浜 宗隆

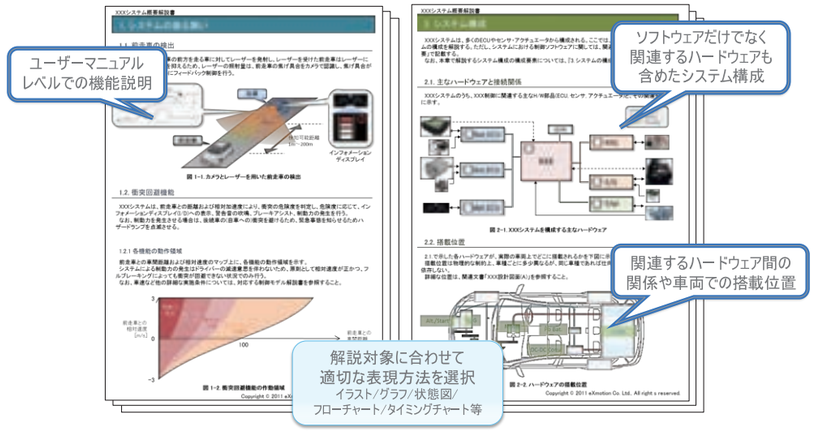
## 「解説書」は“暗黙知”を関係者で共有するためのドキュメント

ここでは、解説書のサンプルを２つ紹介します。一つは、製品全体システムに関わる「システム概要」で、もうひとつは、あるモジュールにフォーカスした「制御設計書」です。

### ■システム概要

下図にある「システム概要」は、ソフト・ハード・企画・品質保証など、さまざまな背景を持つ関係者が読み手になります。そのため、ソフトウェア工学の範疇にある（例えばフローチャートやモデルなどの）アイテムにこだわらず、必要に応じてイラストやグラフなどの表現方法も活用し、読み手への情報の伝わりやすさを第一に考えて記述することが必要となります。また、ユーザーズマニュアルに記載されているような、専門知識を持たない人でも理解できるレベルの内容から、内部の構造であるシステム構成までを、思考の過程を妨げないよう、関連付けを明確にしながら、段階的に詳細化・局所化していきます。こうすることで、ソフトやハードの専門知識を持たない関係者に対しても、深い理解を促すことが可能となります。

『システム概要』※架空のシステムです




### ■制御設計書

右ページにある「制御設計書」は、システムを構成する一部のモジュールを対象範囲とした「解説書」です。このドキュメントの想定読者は、そのモジュールの派生開発の担当者です。

一般的に、派生開発で引き継がれるのは、実際に動かすことのできるコードやモデル、そして機能仕様や要求一覧です。しかし、コードやモデルと機能仕様や要求一覧の間には大きなギャップがあります。

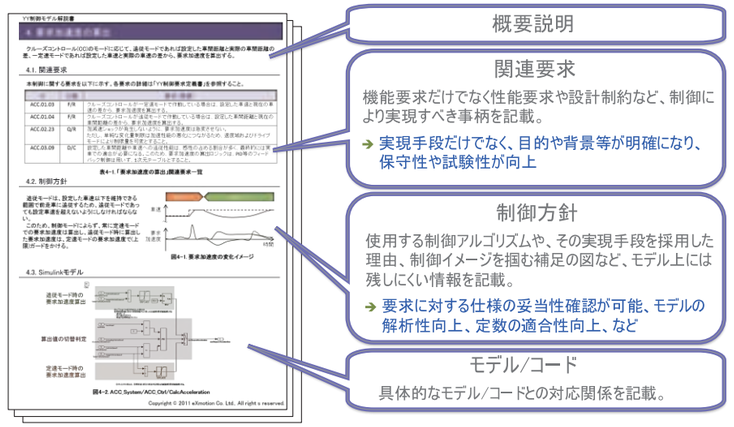
要求や機能は実現手段を選びません。その要求や機能に対して、制約やその他の理由により意思決定がなされ、多くの選択肢の中から一つの実現手段が選ばれます。それが、今のコードやモデルの構造となるのです。しかし、従来の設計書では、その「意思決定がなされた根拠」がまるまる抜け落ちているのです。そのため、引き継いだコードやモデルを、品質よく保守していくには、かなりの「経験」と「想像力」を必要とします。しかし、派生開発は新規開発に比べて“簡単”だと判断され、若手社員や外部社員に任すケースがほとんどです。そのため、ソフトウェアの品質は派生する度に悪くなり、いつの間にか手のつけられない状態になってしまうのです。

ここでは、実際に“架空のシステム”ではありますが、具体例を元に「解説書」を説明しました。これで、一般的な「設計書」と「解説書」の違いがご理解いただけたかと思います。

ところで、「解説書」に書かれている内容は、ドキュメント作成者が考え出したものではありません。それは、最初の開発の時に検討会の資料としてその都度つくられ、担当者のＰＣのＨＤＤに残されたものであったり、検討会の中でホワイトボードに直接かかれた画像として残されたものであったり、ベテラン社員の頭の中にある暗黙知だったりします。その暗黙知を関係者の間で正しく共有することが、ソフトウェアひいては製品開発全体の生産性や品質の向上につながります。



『制御設計書』※架空の制御です




## 誰が「解説書」を作るべきか…？

解説書を具体的に見ていただくと、こんな疑問が出てきたかもしれません。

誰が「解説書」を書くべきか…

解説書に記載している内容は、比較的一般的な知識に基づくものもあれば、高い専門知識に裏付けされたものもあります。そのため、やはり開発者自身が作成するのが適しているのか、というように思われるかもしれません。ただ、前述のように、開発者がドキュメントの作成時間を捻出することは難しいうえ、「使えるドキュメント」の作成スキルと開発スキルは異なります。

そこをカバーできるのが、開発現場主体で活動している、コンサルタントの存在です。

エクスモーションのコンサルタントは、お客様の開発現場で、お客様と一緒になって問題解決にあたります。そのため、組込みソフトウェアの経験が豊富にあり、どんなシステムでも、多少の時間をかければ、ある程度専門性の高い知識であったとしても理解することができます。また、そもそも「抽象化」「整理整頓」はコンサルタントのコアスキルであり、得意分野です。その両方を兼ね備えているからこそ、かゆい所にまで手が届くドキュメントを作ることができるのです。

一方で、開発者は専門家に解説書作成を任せてしまうことで、本来の開発業務に専念することができます。更に、いったん「解説書」が完成すると、最良のドキュメンテーションのお手本を得ることができるのです。お互いが、お互いの得意分野に専念し、協力し合うことで、最高の成果を残す。まさに「餅は餅屋に任せろ」というわけです。

皆さんも、一度、解説書の作成を検討してみてはどうでしょうか？

# プロダクトライン開発

## ～製品群開発を効率化するRIPPLEアプローチ～

**ものがあふれる現代、身の回りにある多くの機器は機能が飽和し、一から製品を開発する機会はほとんどありません。開発現場においても、既存資産を可能な限り活用して新しい製品を作り出すことが、単純に「開発量を減らす＝早期リリースを可能にする」と考えられ、安易なコピーによる開発が横行しています。しかし、そうすることの副次的な問題により、開発現場は一向に楽になることはありません。では、どうすれば良いのでしょうか。**

## 既存資産をコピーすることが引き起こしている問題

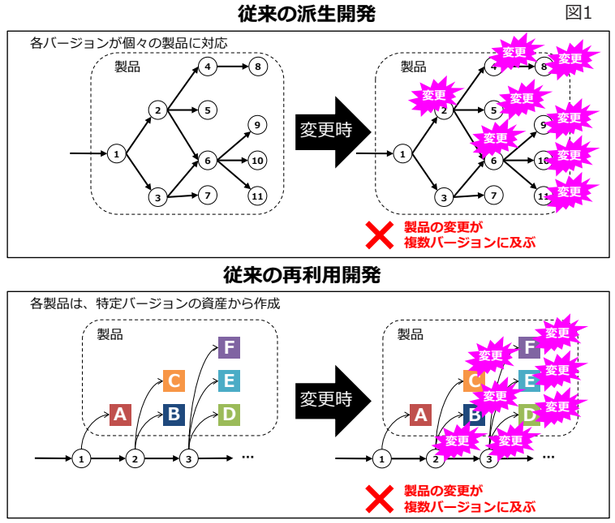
多くの現場では、新製品の開発時に過去に開発した類似製品の資産（＝既存資産）をコピーして、派生開発や再利用開発をスタートします。

図1上は典型的な派生開発を示しています。この方法では製品のコピーを場当たり的に繰り返しており、コピー元の資産は統一されていません。

図1下は典型的な再利用開発を示しています。こちらはコピー元の資産は統一されていますが、各製品はコピー元資産の特定バージョンをコピーしています。

これらの方法は、各製品の資産とコピー元資産の多くの部分を重複してメンテナンスする必要があるため、一つの製品に変更が必要となった時に、同様の変更が複数の製品バージョンに及ぶ可能性があるという問題を抱えています。

図1




## SPL開発とは

上記の問題はSPL(ソフトウェアプロダクトライン)開発によって解決することができます。図2はSPL開発のイメージを示しています。

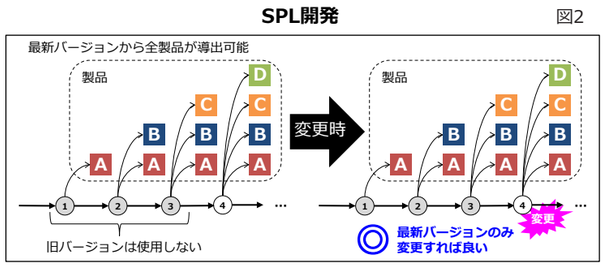
SPL開発では、コピーではなく、統一して管理された資産から各製品を導出する点に大きな特徴があります。

統一して管理された、再利用可能な資産のことをコア資産と呼び、いつでも製品群の全ての製品の導出が可能であるように、定期的にバージョンアップします。そのため、どの製品に変更が必要になっても、変更はコア資産の最新バージョンのみでよくなります。

また、新製品の開発時には、既に開発・テスト済みのコア資産を最大限に活用できるため、新製品固有機能の作り込みのみに専念することができます。

このように変更開発時・新規開発時のいずれにおいても、SPL開発では重複した機能開発を防ぐことができるため、従来に比べて、ムダな開発工数を削減することができます。

図2




## SPL開発への移行の進め方

では、SPL開発に移行するためにはどうすればよいのでしょうか？

一般的には、要求からトップダウン的に、製品群全てを導出可能なコア資産を構築します。しかし、この方法は非常に大掛かりであり、開発現場の負担が大きいため、多くの開発現場では導入に踏み出すことができません。

これに対して、既存資産を最大限に利用することにより、比較的早期にSPL開発に移行することができる方法があります。図3は、そのプロセスを示しています。当社ではこのアプローチのことを、RIPPLE（Rapid/Requirement Integration Production Product Line Evolution）と呼んでいます。

「1. 調査・計画」では、既存資産を解析して製品を統合する順序を決定し、具体的な移行プランを立案します。

「2. 既存資産を統合・差分分析」では、移行プランに従って既存資産の要求やコード/モデルを統合し、統合結果から製品間の差分を明らかにします。

「3. コア資産作成」では、統合した資産から全ての製品を導出できるように、統合した資産のアーキテクチャを再利用しやすい構造に洗練させ、コア資産を作成します。

「4. 製品を導出」では、コア資産から機能を組み合わせる仕組みを構築し、その仕組みを用いて個別の製品を導出します。

以降、1～4の詳細を説明していきます。

図3

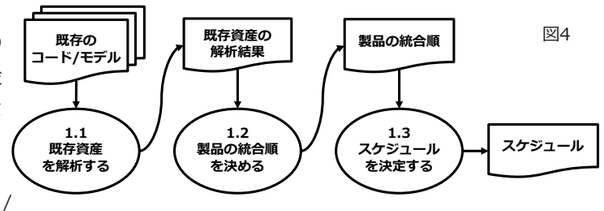

## 1. 調査・計画

SPL開発に移行する際に、既存資産を一度にまとめて統合することは、決して容易ではありません。そのため、事前に既存資産の全体像をしっかり把握し、最適な統合順となる移行計画を立案することが重要になります。図4は移行計画を立案するプロセスを示しています。

「1.1.既存資産を解析する」では、既存資産のコード/モデルなどを解析することによって、製品間の類似度を把握します。

「1.2.製品の統合順を決定する」では、製品間の類似度が高い順に製品の統合順序を決定します。

図4




「1.3.スケジュールを決定する」では、各製品の資産を統合し、コア資産化するまでに要する工数を見積り、全ての製品の資産を統合するまでの詳細なスケジュールの立案を行います。

## コンサルタントが教える成功の秘訣

多くの開発現場では常に開発効率の改善を求められています。これを解決する手段としてSPL開発を取り入れている現場は少なくないようですが、多岐にわたる工学的知識を独力で習得し、正しく実践していくことは大変難しく、期待する効果が得られていないケースも見られます。

エクスモーションでは、これまで様々なSPL開発の導入支援を実施してきました。それら支援を通じて培われた実践的なノウハウにより、現場にとってわかりやすく、具体的なSPL開発の進め方のご提案と実践が可能です。

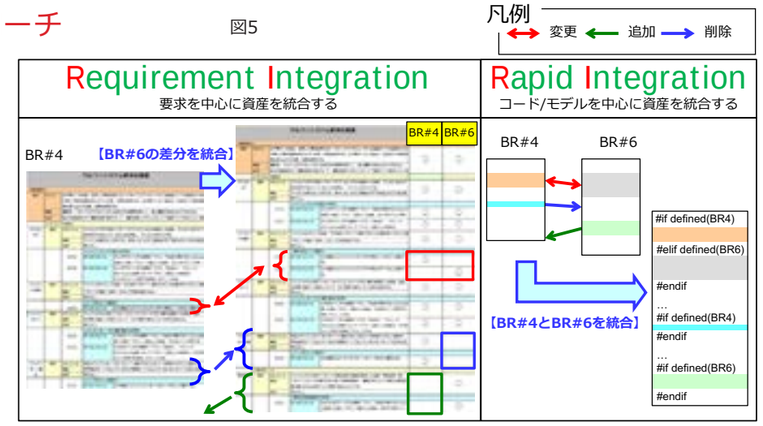
SPL開発の進め方に迷ったら、ぜひ、私たちにご相談ください。皆さんと一緒になって変革を成し遂げることを楽しみにしております。

シニアエンジニア
吉元 崇

## 2. 既存資産を統合・差分分析

### ■RIPPLEの二つの統合アプローチ

図5




SPL開発への移行アプローチであるRIPPLEは、既存資産統合の進め方として、二つのアプローチを用意しています。

一つは要求を中心に資産を統合するRequirement Integrationのアプローチ、もう一つはコード/モデルを中心に資産を統合するRapid Integrationのアプローチです。図5は二つのアプローチの違いを示しています。

要求中心のアプローチの場合、まずは代表製品の既存資産から、要求仕様をUSDM形式で明らかにします。その後、他製品の要求仕様の差分を記述粒度を保ったまま統合し、製品群全体の要求仕様と製品ごとに搭載されている機能を定義します。

USDM形式で記述することによって、製品群全体の機能を漏れなく、正確に把握することができ、加えて製品ごとの機能の差異が明らかになります。

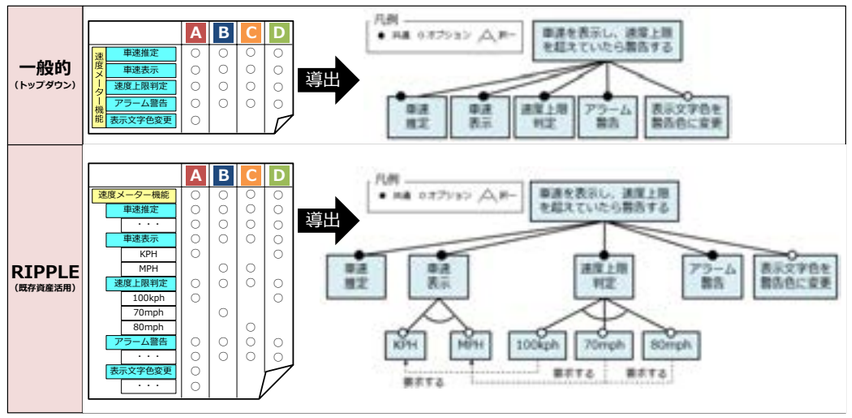
つまりこのアプローチは、既存資産の要求仕様の統合と製品群における搭載機能の違いに関する分析（可変性分析）を同時に、かつ確実に行うことができます。

一方、コード/モデル中心のアプローチの場合は、複数製品間のコード/モデルに共通した処理や異なる処理を、ツールによって自動的に判別し、判別結果を確認しながら差分を統合することで、全製品を導出可能な、包括的なコード/モデルを作成します。この場合、コード/モデルの統合はスピーディーに行えますが、製品群の可変性を分析するためには、製品間のコード/モデルの単純な差異から、製品間の要求の違いを導出する必要があります。

これら二つのアプロ―チは、可変性分析を確実に行う事を重視するか、統合するスピードを重視するかで、使い分けることができます。

### ■一般的なトップダウンのアプローチとの違い

図6




可変性分析における成果物は、全製品に共通・可変な機能をツリー構造で表現し、機能間の依存関係（他機能を要求・排除する関係）を定義した「フィーチャーモデル」と呼ばれるものです。

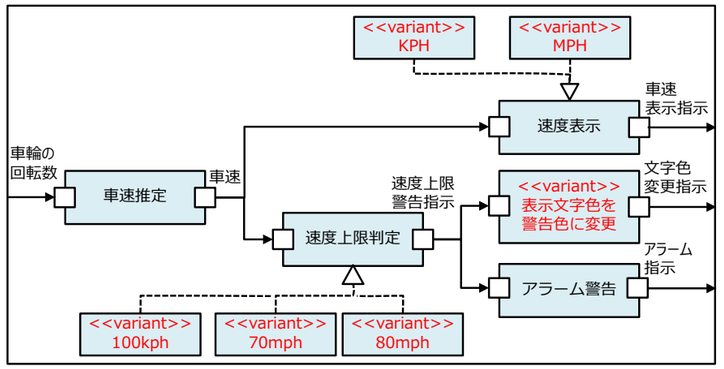
一般的なSPL開発では、フィーチャーモデルを作成する前に、製品群全体の機能を製品企画書等のビジネス要件から抽出し、製品ごとの機能搭載の有無を分析する必要があります（図6上）。

この場合、企画書等からトップダウン的に詳細な機能を、網羅的かつ粒度を統一して見極めることは困難であるため、フィーチャーモデルは粗い粒度となりやすく、実際に開発されるアーキテクチャの実体とかけ離れてしまう恐れがあります。

これに対してRIPPLEアプローチでは、前述のように製品群の特性を、既存資産を元に仕様の粒度まで網羅的に分析するため、より詳細で粒度の統一された、アーキテクチャの実体に近いフィーチャーモデルを定義することができます（図6下）

## 3. コア資産作成

図7




可変性分析が完了し、製品群全体の機能が明らかになったら、製品群で再利用可能な資産である「コア資産」を作り込みます。

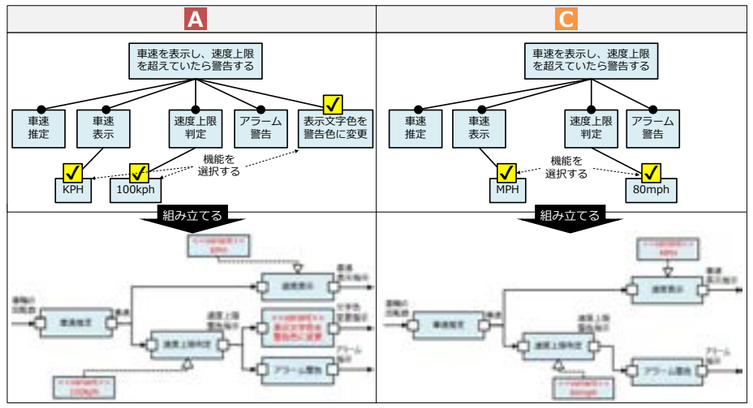
コア資産には、アーキテクチャモデルや設計資料、実装・テスト成果物が含まれます。これらの資産は、フィーチャーモデルの各機能と対応付けて管理します。

図7は、コア資産におけるアーキテクチャの例を示しています。コア資産は全ての製品群の機能を網羅するため、全機能を実現する構成要素を持つ“包括的なアーキテクチャ”となっています。

製品毎に可変となる構成要素は、フィーチャーモデルで定義した機能と対応付け、ステレオタイプに<<variant>>を設定します。

## 4. 製品を導出

図8




コア資産が完成したら、コア資産から各製品を導出するための仕組みを構築し、導出します。

図8の例では、製品AとCで、フィーチャーモデル上で異なる機能を選択し、それに対応した異なる構成要素から成る各製品のアーキテクチャモデルが導出されることを示しています。アーキテクチャモデルだけでなく、ドキュメントやテストなども、その製品ごとに選択した機能に応じたものが導出されます。

このように、SPLを導入することで、製品群の全ての製品の資産を一つのコア資産から導出できるようになります。その結果、重複開発を劇的に減らすことが可能となり、従来に比べて製品開発に要する工数を大きく削減することができます。

# XDDPによる派生開発

**派生開発における失敗やトラブルの原因は、変更仕様のモレやミスによる手戻り、影響範囲の特定が不十分であることによる不具合がほとんどです。それを未然に防ぐのが『XDDPによる派生開発』です。**

## 「XDDP」はソフトウエアの派生開発に特化したプロセスです

現在のソフトウェア開発においては、新規開発はほとんどなく、既存コードに機能追加や変更を行う『派生開発』が多くなっています。

一般的に、変更や追加は仕様レベルでエンジニアに伝えられることが多く、また、納期も短いことから、変更仕様をそのままコードで実現してしまいます。その結果、変更箇所以外の仕様が漏れたり、変更箇所以外の不具合による手戻りの連続により、工数が増大するとともに、エンジニアも疲弊してしまいます。

これまで、ソフトウェア開発技術は欧米を中心として発展し、新規開発を対象としていましたが、派生開発に特化した唯一のプロセスが「XDDP」です。XDDPは変更や追加に必要最低限のドキュメントとプロセスが定義されており、派生開発を効率よく進めることができます。

## 「３つの成果物」により、モレ・ミス・ムダを防止します

XDDPの特徴は『3つの成果物』による、ミス・モレ・ムダの防止です。この3つの成果物を完成させるまでは、絶対に実装をしません。完全なるフロントローディングを行うことにより、手戻りによるムダを排除することで、派生開発の問題を未然に防ぐのです。

### ■USDMで仕様のモレを防ぎます

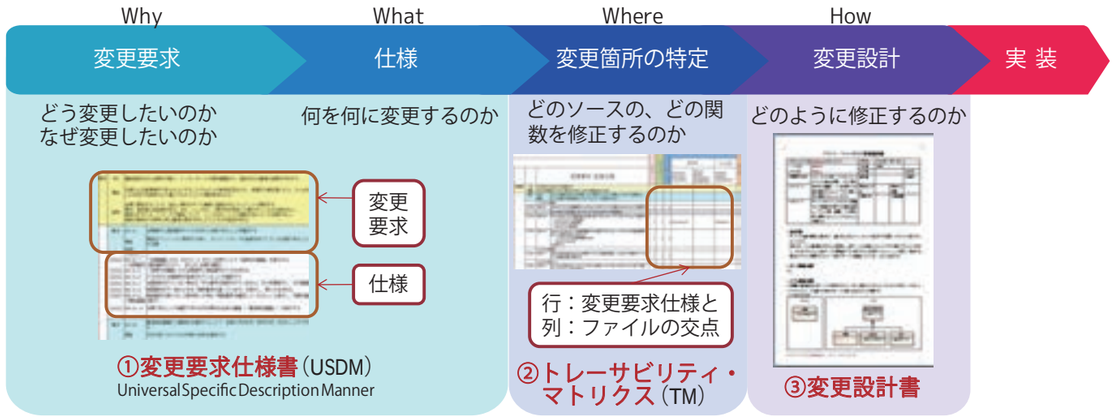
USDMは要求と仕様を形式化するツールです。派生開発における変更要求仕様や追加機能要求仕様をUSDMで記述することにより、要求仕様のヌケモレや矛盾などの問題を見つけやすくなります。

### ■TMで変更のモレを防ぎます

トレーサビリティマトリクス（TM）は、USDMで定義した仕様に対する、コードの対応箇所を示した対応表です。このマトリクスにより、変更仕様に対する影響範囲を把握することができ、変更箇所のヌケモレを防ぐことができます。

### ■変更設計書で､各変更をきちんと設計します

変更設計書は、変更箇所に対してどのような変更をするのかを設計し、明文化したものです。これを有識者との間できちんとレビューすることにより、対応のミスを防ぐことができます。



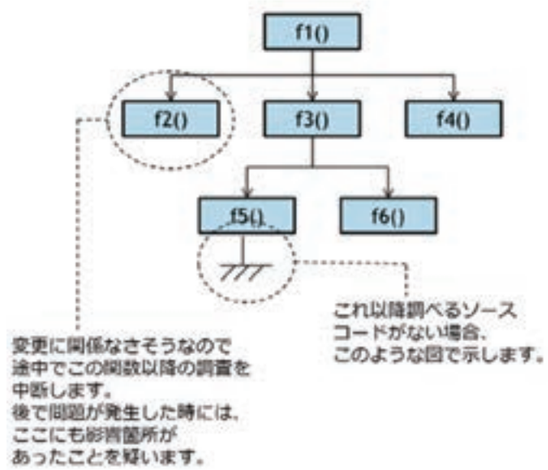

## Why / What： 変更内容の調査と仕様化を行います

XDDPでは、まず最初に変更要求に対する具体的な変更内容をとらえて、変更要求仕様として定義します。変更内容をとらえる方法としては、既存の機能仕様書などの資産から変更すべき内容を探し出す方法の他に、ソースコードから探し出す「スペックアウト」があります。スペックアウトとは理解したことと、拾い出した箇所、変更すべき項目を抽出して資料として残すことです。ここでとらえた変更内容を「要求」「仕様」に振り分けてUSDMに記述することで変更要求仕様として定義します。

### ■ソースコードからも「スペックアウト」して仕様をとらえます

既存資産の要求仕様書や設計書が整備されていれば、変更内容をとらえるのは容易です。しかし、それらがメンテナンスされておらず、ソースコードと乖離している場合もあります。そのような時にはソースコードを元に「スペックアウト」し、変更内容を探し出します。とはいえ、大量のソースコードに対して、やみくもにスペックアウトすると、いくら時間があっても足りません。スペックアウトは、変更箇所にあたりをつけてから実施します。

スペックアウトで、ソースコードを理解しながら「図面化」し、理解した箇所をチェックして進めていきます。図面化は構造と振る舞いに着目し、開発対象の特徴に合わせて構造図などの最適な図面を選択します。右図は構造図の実施例です。




### ■とらえた仕様はUSDMでモレなく正確に仕様化します

スペックアウトで抽出したものも含めて、ここまででとらえた変更内容を元に、変更要求仕様書をUSDMを使って作成します。USDMを作成する上で重要な点は要求と仕様を分けて書くことです。そうすることで、要求と仕様の関係が明確になり、仕様のヌケモレや認識違いを防ぐことができます。その他、Before/Afterで記述する、変更の範囲・要求の理由を記述する、等のポイントがあります（詳細は7ページの「要求の定義と仕様化」をご覧ください）。

また、USDMで作成した変更要求仕様書は設計者に対する「作業指示」としても使えます。USDMでは、仕様のモレや誤解を防ぐために、仕様を具体的に、検証可能なレベルで記載します。そうすることで、設計者に対して「厳密な変更の指示」が可能になります。変更要求仕様書が適切に作成されており、仕様通り変更作業を行えば、どんなスキルレベルの設計者であっても同じ結果になるはずです。

## コンサルタントが教える成功の秘訣

XDDPを導入する上で多くの人がつまずくのが、最初の一歩となる『変更要求仕様書（USDM）』のところです。USDMでは、要求を階層化しグルーピングしていくのですが、どんな粒度や視点で階層化やグルーピングするのか、初心者の方はずいぶん悩まれるようです。

要求の階層化は、「スペックアウト」で正確に仕様を抽出し、その仕様から構成や時系列の視点で要求を定義、整理します。ここで、スペックアウトを行うには設計技術が必要となります。この仕様化技術と設計技術のコツさえ掴んでしまえば、後はスムーズに進むのですが、この段階でつまずくと、XDDPの導入はうまくいきません。

エクスモーションでは、仕様化技術、および、設計技術をベースにXDDPの導入を支援するサービスを行っております。

エグゼクティブコンサルタント
斎藤　賢一

## Where：変更仕様に対する変更箇所を特定します

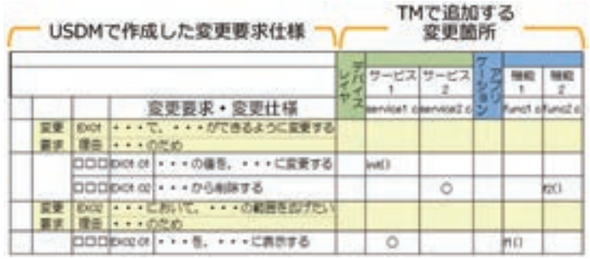
TMはUSDMで定義した仕様に対し、変更点がどのモジュールに存在するかを示します。そのため、変更に対する影響範囲を確認することができます。

また、前のプロセスで作成した「変更要求仕様書」と、この後に作成する「変更設計書」とをつなぐ役割を果たします。

・変更するのは本当にその関数だけで良いのか、他にも変更箇所がないか
・変更することで、他の関数への影響がないか
・別の変更仕様で、同じ関数が変更されないか

といったことを具体的に確認します。

変更に対する影響箇所を調べていくと、凝集度の低いモジュールや、結合度が高いモジュール構造が見つかる場合があります。このような箇所は変更の確認が難しいだけでなく、変更による二次障害が発生しやすくなります。その際には、「保守性の向上」を変更要求としてUSDMに追加します。

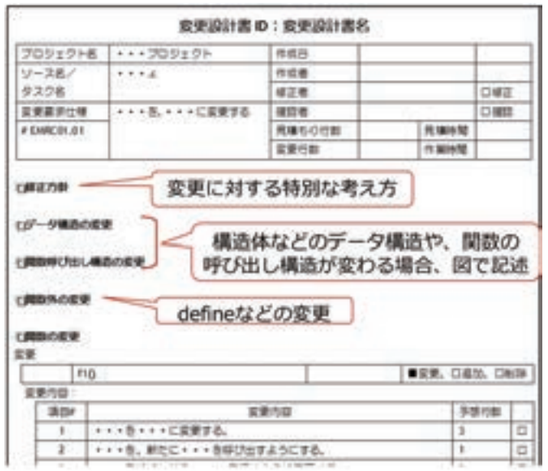



## How：どのように変更するかを決定します

変更箇所の特定ができたら、それぞれの変更箇所について、どのように変更するかを決定し、変更設計書を作成します。ソースコードではなく「文章」で記述することにより、変更方法の正しさをレビューできると共に、変更方法を確実に残すことができます。変更内容の詳細度は、実際に変更作業を行う担当者のスキルに応じて臨機応変に決定します。

変更内容を記述する中で、関数の引数や戻り値の変更に気づくことがあります。このような変更は他の関数へ影響を与えるので、TMに戻って影響範囲を確認をすることが必要です。

変更要求仕様書と変更設計書を作成し、レビューを実施することで変更内容が確定した後は、ソースコードを一斉に変更します。




## 開発を支援する様々なこと

XDDPでは必要最低限のプロセスとドキュメントを定めています。しかし、品質を保ち、継続的な開発を進めていくためには「レビューの実施」「適切な見積もりと計画」「要件の管理」「公式ドキュメントの維持」「開発プロセスの更新」などの、開発を支援する様々なことも必要になります。

### ■レビュー

一般的な派生開発では、担当者があたりを付けてコードを変更するという、思い込みによる変更が行われています。そのため、問題が多く発生しています。この問題を解決する効果的な方法はレビューしかありません。XDDPでは3つの成果物を用いることで、変更箇所、影響範囲、変更方法等のレビューを実施することができます。

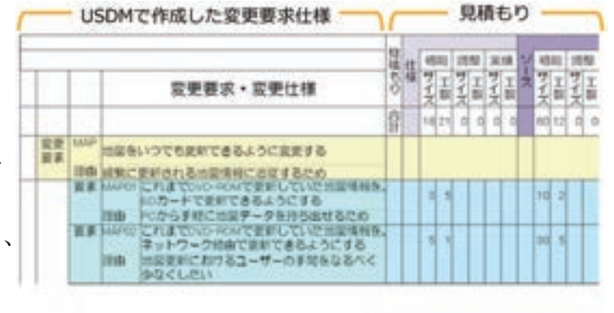
### ■公式ドキュメントの維持

一般的な派生開発では、新規開発の流れに沿ってドキュメントを完成させながら変更の作業が進められます。したがって、変更仕様の改訂が頻繁に起こるような場合は、ドキュメントの変更回数も増えていきます。しかし、限られた時間の中ではコードの修正が優先され、ドキュメントの更新がなされないまま、開発が進められてしまいます。

XDDPで作成するドキュメントは全て既存資産との「差分」です。テストで変更の正しさを確認し、問題がないと判断した時点で、公式な設計文書へ「差分」をマージします。そのため公式文書の変更を１回で済ませることができます。

### ■見積もりと計画

限られた期間で慌てずにXDDPに取り組むためには、サイズに基づいた見積もりが不可欠です。XDDPでは見積もりをするタイミングと、各タイミングにおいて何に着目して見積もりするかを定めています。またUSDMにおいて、Before/Afterを記述することで見えてきた「変更の量」や「難易度」も、見積もりに活用します。




### ■モレを逃さない要件追跡リスト

暫定の仕様や、一度要求仕様が確定した後に生じた変更は、全て「要件追跡リスト」に登録します。この時、変更作業担当者とは別の人を追跡担当者として確保します。追跡担当者はXDDPのプロセスに沿って作業を進めるよう、変更作業担当者へ促します。また実現性が危ぶまれている変更要求仕様などもリストに登録し、適切に設計・実装されたかを追跡します。このようにすることで、忘れがちな追加や変更への対応も逃さずに進めることができます。


## XDDP導入における「よくある問題」

XDDPを導入する際によくある問題として、まず、現状の問題を明らかにせずにXDDPに飛びついてしまい、効果が出ないといった問題があります。XDDP導入にあたっては、現状のプロセスの問題点を明らかにし、XDDPによる改善をイメージできるようになってから導入する必要があります。

次に、スペックアウトが十分できていないまま、あいまいな状態でUSDMを記述することがあります。このような状態では変更要求仕様としては不十分な状態のため、実装でモレや勘違いに気づき、手戻りとなります。

また、実際にXDDPを実践してみると、要求や仕様をどう分割していいかわからず、途方に暮れることがあります。仕方なく、思いついた要求と仕様のみを記述し、先に進むことで、後になってから仕様モレに気付くことがあります。

XDDPは、比較的早い段階ではっきりとした効果が得られる技術です。しかし、誤った適用や正しくない導入により、失敗してしまうケースもあります。それを未然に防ぐためには、経験豊富な専門家に依頼することも、一つの方法と言えます。