モデルベースシステムズエンジニアリング

導入の手引き

2013 年 8 月

独立行政法人情報処理推進機構
Information-technology Promotion Agency, Japan

序文

モデルベースシステムズエンジニアリング（Model-Based Systems Engineering、以下 MBSE と略す）は欧米では既に、航空・宇宙、防衛・軍事、鉄道、自動車など、複雑で高い品質を求められる製品やサービスに適用されています。

しかし、平成 24 年度初頭の日本では、実験的な導入を数例ほど確認できるだけでした。IPA/SEC 内に設置されていた統合システムモデリング技術 WG で議論の末に、MBSE が有効であるということが浸透していないから、日本では普及しないという結論に達しました。

そこで本手引きを企画して、WG 内で議論を重ね、本手引きを作成する運びとなりました。それから 1 年ほどで、日本でも一部導入が始まりました。本格的に導入を検討している組織も増えてきました。おそらく、自動車、スマートシステム、鉄道などの分野で盛んに導入されることと思われます。

更に導入を加速するためには、起爆剤が必要であると考えています。本手引きが日本の開発現場に MBSE を導入する起爆剤となれるのであれば幸甚です。

2013 年 8 月

目次

1. 「はじめに」

この手引きでは、モノづくり、コトづくりを推進する業界の皆様に、多岐にわたる開発を成功に導くために有効なモデルベースシステムズエンジニアリング（Model-Based Systems Engineering、 以下 MBSE と略す）をご紹介いたします。MBSE 導入の狙いは、「製品やサービスなどのシステムの開発を成功に導く」ことです。そして、MBSE は、開発の複雑さに対応するための技法であり、開発プロセスでもあります。そこで、本章では、皆様の組織や開発部門へ MBSE を導入することによって、どのような効果をもたらし、そこに革新を起こすことが可能となるかを示します。更に MBSE を導入する際の標準的な手順を示します。

1.1. MBSE 導入の狙い

日本のシステム開発には様々な課題があると言われています。以下に代表的なシステム開発における現状の課題を列挙します。

- システムの複雑化に伴う課題

  複雑化に伴い、全体像を把握することが困難になり、システム全体に不整合がおき、開発効率が低下する

- システムの派生やシリーズ化に伴う課題

  あるシステムをベースにした派生開発や、製品やサービスのシリーズ化に伴い、製品系列ごとの成果物や、開発工程の管理が複雑化し、困難になる

- 要求の不適切な分析／定義に伴う課題

  要求分析が網羅的ではなく、定義方法が曖昧であるため、要求の抜け漏れ重複のみならず、誤解も発生し、効率的に開発ができなくなる

- 要求変更への対応に伴う課題

  要求変更の発生に対して、要求のトレースができず、システムアーキテクチャ自体の変更が余儀なくされる

- 大きな手戻りの発生に関する課題

  テスト工程に入って、大きな問題が明らかになり、意図しなかった根本的な手戻りが発生することで大きな損失を被る

・ドキュメントの標準化に関する課題

ドキュメントが標準化されておらず、運用や保守時に役に立たない

・関係者間のシステム開発に対する意識の統一や均質化に関する課題

関係者間の意識にばらつきがあると、勘違いや思い違いによる不整合が発生し、品質や開発効率の低下に繋がる

上記のような課題に対し、これまで、日本で製品やサービスなどのシステムを開発する企業の多くは、全社的と言うよりは、部課のレベル、局所最適化で対応してきました。しかし、近年の大規模化、複雑化するシステムの開発においては、課題が局所最適で扱える限界を超えてしまっているとの声も聞かれます。

MBSE は、「製品やサービスなどのシステムの開発を成功に導く」ことを目的として、システム開発の全体最適を図るための技法、そのためのプロセスを定義しています。システムの全体最適を行うためには、高い抽象度を持ち、システム全体を見渡す鳥瞰的な視点と、それを基に、システムを分解、詳細化し、開発効率を上げる分析的視点が必要となります。

ですから、これまで単独ソフトウェアや、ハードウェアモジュールの開発によく見られる、「出荷直前の手直し」という局所最適開発の立場からすれば、MBSE の理解が困難であったり、考え方が受け入れ難かったりすることもあるようです。

本章では、これ以降、MBSE を理解することの助けになるよう、様々な角度からシステム開発の課題とそれに対する解について考えてみましょう。

システム開発の第一歩は、利害関係者の曖昧な要求を元に、これから開発しようとする製品やサービスに対する要求を明確にするところから始まります。そして、その際には、具体的なハードウェアやソフトウェアのコンポーネントや技術そのものを要求として挙げるのではなく、まずは、製品やサービスを見通せるように、できるだけ抽象度を上げてシステムに対する要求を考えることが重要となります。複数の工学分野にまたがる技術や要素から構成される製品やサービスを提供するには、その開発の段階で、多岐にわたる分野の関係者間で様々なやり取りが行われます。円滑にやり取りを行うためには、それら関係者間でお互いに何を考えているのかを共有することが重要です。人は頭の中でものごとを考える際には、図的（ビジュアル）な何かを頭の中に思い浮かべて考

えているはずです。開発関係者にとって、その図あるいは絵をどのように外に出し、共有するのかが、重要なポイントとなります。

私たちの身の周りにある製品やサービスは、以前にも増して、多くの機能を有しており、機械、電気などのハードウェアに加えてソフトウェアやデータが構成要素として複雑に関係しています。こうした製品やサービスを一つのシステムとして捉え、そのシステムをどのように成功裏に開発するかは、それを開発する企業にとって、大変に重要なものとなっています。また、昔のように製品やサービスが単体で成り立つのではなく、他の製品やサービスと関連性をもって所望の機能をもつことがあり、また、それによって更に価値を生むことが多くなっています。このように異なる複数のシステムが互いに複雑な関係をもつシステムは、System of Systems（以下、SoS と略す）と呼ばれます。例えば、SoS の中でのある一つのシステムを開発する際には、他のシステムとの接続性を考慮する必要があります。

製品やサービスに求められる機能が多様となり、高度になったことで、システムにおけるソフトウェアへの依存度が高くなりました。家電製品や自動車、スマートフォンなどマイコンや CPU（中央処理ユニット）で制御されるシステムが増え、メカ、エレキ、ソフトが複雑に関係性を持つため、その開発過程では、様々な複数の分野にまたがる関係者が開発に携わることになります。更に、その開発は、ある企業 1 社内のみで行われるのではなく、分業化が進んでいるために、国内外の複数の会社が関係する場合も少なくありません。こうした国際的な分業化が進む中で、いかに効率良く、要求される品質、コスト、時間に見合った製品やサービスを開発するか、ということも企業に強く必要とされています。そのためには、生産性の高い効率的なプロセス管理も必要となります。複数の分野にまたがる複数のチームがお互いにそのプロセスを理解しながらシステム開発を推進するためにも、従来の文書に基づくアプローチに代わるものが必要とされています。

冒頭で述べたように、MBSE 導入の狙いは、

「製品やサービスなどのシステムの開発を成功に導く」

ということです。MBSE の技法やプロセスには以下のような特徴があります。

- 複数の分野にまたがる関係者が互いに図を見ながらコミュニケーションをとることができる。

- 開発の初期の段階で要求の明確化を図的に行うことにより関係者間の理解が進み、開発途中での意図しない手戻りの発生を抑えることができる。
- 要求のトレーサビリティが確保されるので、要求の変更へ適切な対応を行うことができる。
- 図的に表現されたシステムモデルを再利用できることから、製品やサービスのシリーズ化、ファミリー化を容易にする。

そして、MBSE の技法やプロセスが持つこれらの特徴こそが、上記のような多くの課題の解決に有効なのです[1]。以下の節では、MBSE を理解し、有効活用するため、特に重要なトピックスについて、更に踏み込んでみましょう。

1.2. 開発プロセスを可視化すること

製品やサービスなどのシステムの開発プロセスは、ソフトウェアやメカ、エレキなどの領域をまたがったコンカレント開発のプロセスです。これを可視化することは、ソフトウェアや、ハードウェア単独の開発に比べて、混乱を避けるという観点から、極めて重要です。システムを構成するサブシステムやコンポーネントの開発を任されたエンジニアが、他のサブシステムやコンポーネントに関する情報を一切知らずに、任された部分のみの開発をすることは不可能です。元の要求からのトレーサビリティはどのように関係しているのか、機能としてどのように他の部分と関係しているのか、構造的にどのように他の部分と接続されているのか、他の部分とパラメトリックにどのような関係を持っているのか、などこうしたことを明確に把握することは、任された部分の開発のために重要です。そして、他の部分の開発状況を知ることもまた、重要となります。

IEEE 1220[2]によるシステムズエンジニアリングプロセスの基本的な定義は、図 1 のようにまとめられています。

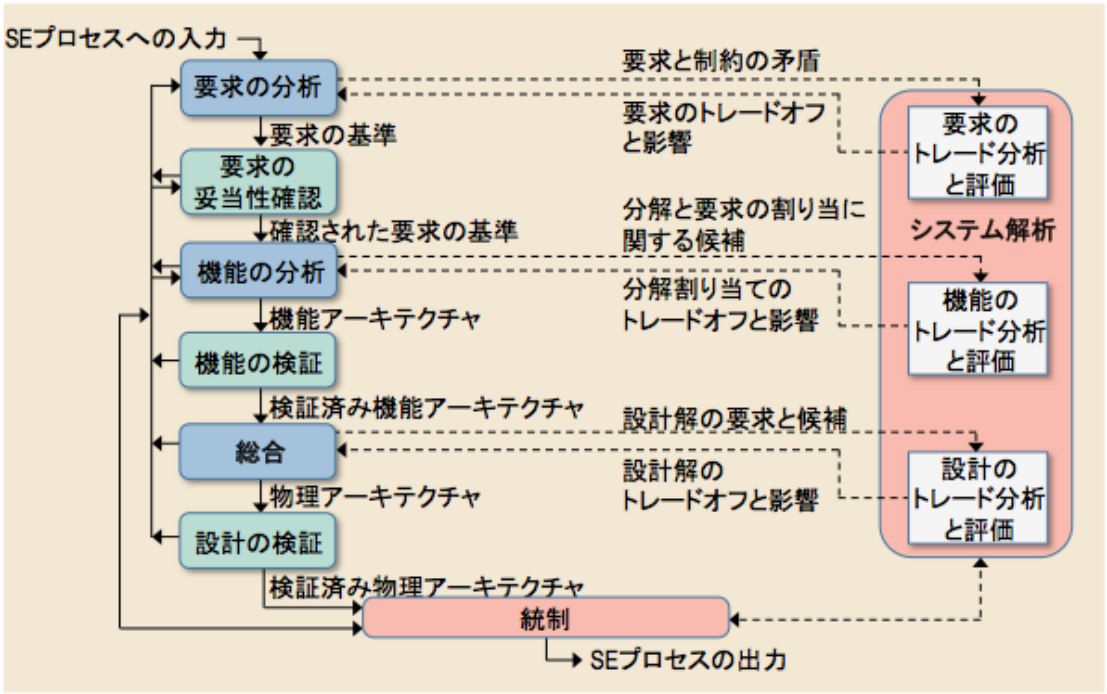

図　1　IEEE1220 システムズエンジニアリングプロセス

1) 要求の分析

システムズエンジニアリングプロセスへの入力を受けて、要求と制約の矛盾を考慮し、要求のトレード分析と評価を行う。

2) 要求の妥当性確認

要求の基準について妥当性確認をとる。

3) 機能の分析

確認された要求の基準に沿って、システムの分解と要求の割り当てを行い、幾つかの候補の中で、機能のトレード分析と評価を行う。そして、機能アーキテクチャを決定する。

4) 機能の検証

機能アーキテクチャを検証し、物理アーキテクチャを決定するための総合的な設計に進む。

5) 総合

機能アーキテクチャを満たす設計解の候補から、トレード分析と評価を行い、物理アーキテクチャを決定する。

6) 設計の検証、統制

設計された物理アーキテクチャを検証し、システム解析との統制をとって、システムズエンジニアリングプロセスの成果物とする。

ここで注意が必要なことは、これらシステムズエンジニアリングのプロセスは、ソフトウェア開発のためのウォーターフォールプロセスのようにトップダウンで一方的に進むのではなく、繰り返し実行される可能性があるということです。また、繰り返すといっても、それは、主として開発リスク低減に効果があると言われるスパイラル開発のように、設計と実装を繰り返すのではなく、システム分解の異なるレベル（システム、ハードウェアやソフトウェアサブシステム、ソフトウェアシステム内のソフトウェアコンポーネントなど）ごとに分析、設計を繰り返すのです。そして、その繰り返しは、同一レベルで行われることも、分解度の高いレベルでのプロセスから、分解度の低いレベルでのプロセスにフィードバックがかかるという繰り返しもあります。ただし、もちろん、繰り返しが何度も生じるのは、開発コストの増大や出荷への影響が大きいため、意図しない繰り返しはできるだけ避ける必要があります。

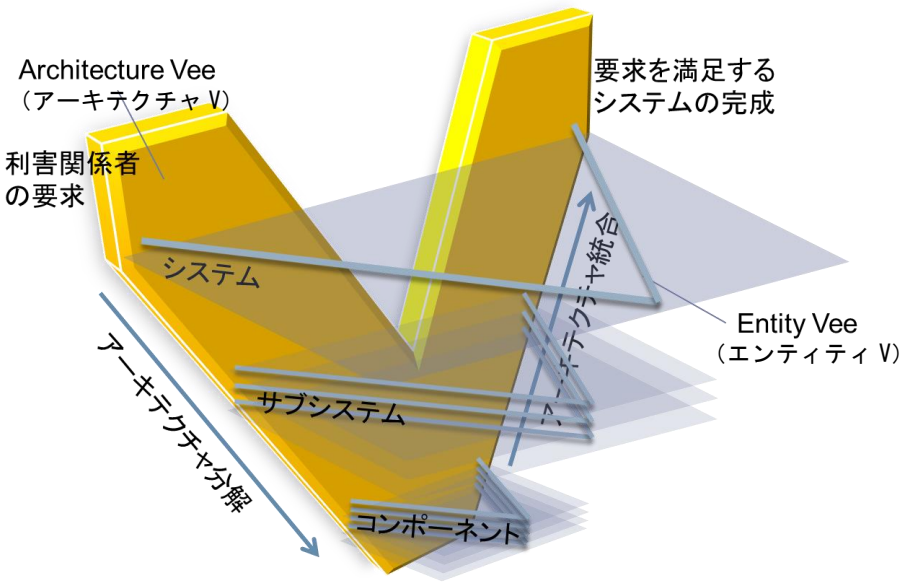

図 2　二元 V 字モデル[3]

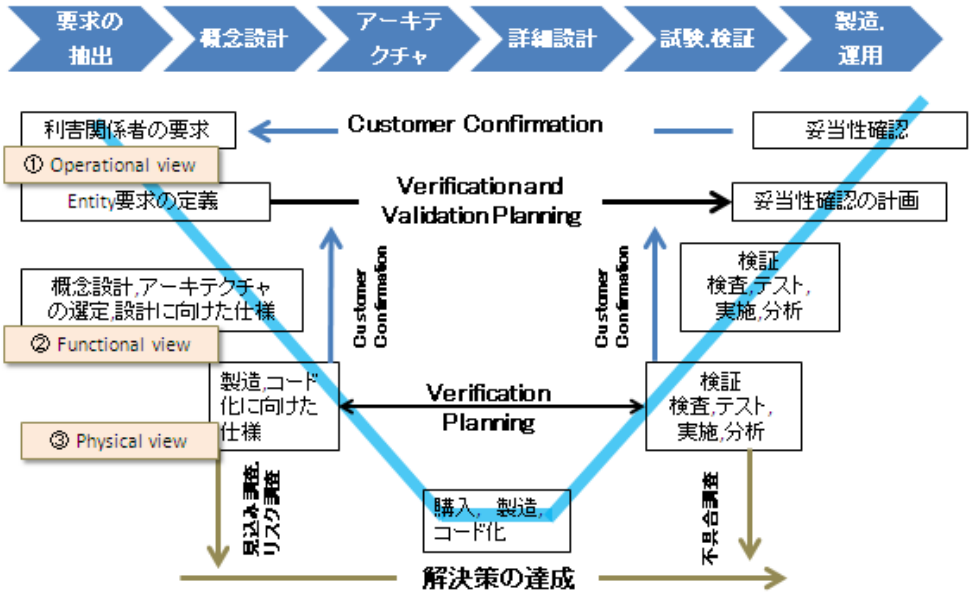

図 3　エンティティ V

システムズエンジニアリングの特徴であるこうした関係性や開発のプロセスを把握するのに適した二元 V 字モデル(Dual Vee Model) [3]及びエンティティ V を図 2 と図 3 にそれぞれ示します。詳細については、次の章で触れますが、このような図に基づいて、プロジェクトマネージャやシステムズエンジニアなど開発関係者がコミュニケーションをとることで、現在の開発プロセスがどのような状況にあるのか、どこで手戻りが発生する可能性があるのかなどを把握することができます。こうして、複数の専門チームがコンカレントデザインを実施することができます。

システム開発は、システムレベルでの解析結果に基づき、進めていくことになります。

そして、システム開発の初期の段階であるシステムレベルでの検討時から、一貫してモデルベースでシステムズエンジニアリングプロセスを実行するために、SysML(Systems Modeling Language) [4] [5]が脚光を浴びています。システムモデルを記述する言語の一つである SysML を利用することで、モデル駆動型システム開発[6]が容易になります。

SysML を用いる MBSE にも様々な方法論がありますが、どの方法論においても、システムの本質を表すモデルを作成するため、まずはじめに、システム解析のために最低限必要な、要求、振る舞い、構造、パラメトリック制約といった、基本的なシステムモデルを作成します。それとともに、どのような問題をどのような条件で解くべきである

かといった、決定しておくべき関心事を導き出すためのモデルを、MBSE プロセスの中で作成することになります。

IEEE 1220 の図　1 における右側の「システム解析」と書かれたブロックでは、要求、機能、設計それぞれについてトレード分析と評価を行うフローが記述されています。ここで、MATLAB/Simulink、AMESim や、Dymola、SimulationX、MapleSim などの Modelica 言語に基づく物理シミュレーションソフトウェアなどが用いられることがあります。また、Co-simulation（協調シミュレーション）と言われるように、必要に応じて複数ドメインのソフトウェアを同時に実行する場合もあります。

よく誤解されることですが、SysML で記述されたものをはじめ、MBSE で扱われるシステムモデルは、必ずしも、実行可能であることを意味していません。すなわち、システムレベルより粒度の細かいソフトウェアサブシステムなどで用いられるような具体的なモデルを作成することが MBSE の目的ではありません。MBSE で実行可能モデルを用いるのは、そのモデルを実行することが「システム解析」に役に立つ場合です。すなわち、実行可能モデルを実行することで、システムの本質的な振る舞いを明確にし、機能や設計に関するトレード分析と評価を行うためです。

1.3. 品質（ilities[1]）を確保すること

システムの高機能化、複雑化に伴い、安全性、信頼性、セキュリティといったシステムに必要とされる品質を保持することが困難になっています。このため、品質を保持することの重要性はますます増大しています。交通関連や医療関連の製品やサービスでは、人命に関わる事故に繋がり兼ねませんので、大変重要です。安全性や信頼性に関わる法規、規制、標準などを満たす製品やサービスを提供することは必須です。これら安全性や信頼性の法規などに共通するのは、市場に出荷された製品が安全であることをメーカが保証することです。

近年ではそれだけではなく、その製品の開発時から、安全性や信頼性がどのように考慮されて開発されたかを説明できなければなりません。できあがった製品やサービスは、安全性があり、信頼性があると主張するだけではなく、開発のプロセスで安全性、信頼

[1] Reliability、Usability、Maintainability、Dependability、Security など、多くの場合は~ility が品質の指標に使われる。これらを総称して、quality の代わりに ilities という表現が世界的に使われている。

性を考慮しておかなければなりません。これは、最終的にできあがった製品やサービスについて、開発の上流から要求やテスト、製造に至るプロセス全体を通してトレースがとれた形で示されなければならないことを意味します。開発の上流プロセスで利害関係者の要求を明確化し、システム要求を導き出し、アーキテクチャを選定するだけでは不十分です。システム要求を導き出したら、そのテスト方法を計画しておかなければなりません。そうしておかなければ、システムが正しく作り上げられたのかどうかを判断できません。なお、標準 IEEE 1220[2]で、システム要求は測定可能かつテスト可能でなければならないと規定されています。

要求のトレーサビリティが確保されていると何が良いのでしょうか？利害関係者の要求が明確になり、そこからシステム要求の導出に至るまでのトレースがとれると、導出された要求の元となる要求を知ることができます。システムに与えられた仕様がそもそも、どこから来ているのかを明確にできます。また、システム開発を進め、サブシステムレベル、コンポーネントレベルに進んでいくと、それらに対する要求が細分化されていきます。場合によっては、当初目標としていた要求を満足しないことが判明する場合があります。そうした場合に、要求の変更を行わなければならないことがあります。トレーサビリティがとれていれば、システムモデルに基づき、どこに要求変更による影響があるのかを見極めることができ、臨機応変に対応することができます。この要求のトレーサビリティとともに、要求の検証及び妥当性確認の方法をまとめた、RVTM（Requirement Verification Traceability Matrix、略して、ベリマトとも呼ばれます）があります。こうした表を用いて、要求のトレーサビリティを明確に表すことがシステム開発には求められます。

それでは、テスト方法を計画するにはどうすれば良いでしょうか？次節では、検証と妥当性確認について述べます。

1.4. 検証と妥当性確認を行うこと

通常、一言でテストと言われるものは、大きく、以下のように検証と妥当性確認に分けることができます。

検証：テストするべきことをテストしたか？

妥当性確認：正しいことをテストしたか？

あるいは、

検証：システムを正しく作り上げているかどうかを調べること。

妥当性確認：正しいシステムを作り上げているかどうかを調べること。

と区別することもあります。

仕様として明確になったシステム要求、サブシステム要求、コンポーネント要求をテストするのは、検証です。一方、妥当性確認は、利害関係者が要求していたシステムがそのとおりに作り上げられているかどうかを問うもので、これは非常に難しいことです。しかしながら、利害関係者から受容されないシステムは最終的に、意味がありません。システム開発の初期の段階から、利害関係者の要求を明確にしたら、どのように妥当性確認をとればよいかを検討する必要があります。このような開発初期の段階で行われる妥当性確認は Early validation と呼ばれます[7]。図 1 に示した IEEE 1220 のシステムズエンジニアリングプロセスで述べたとおり、要求の妥当性確認、機能の検証、設計の検証が順次行われることにご注意ください。

最終的にシステムができあがりつつあるフェーズで検証や妥当性確認を行うことも大変重要ですが、それと同じくらいに、開発初期のフェーズ、すなわちシステム要求を明確にしていく過程やアーキテクチャを選定する早い段階で、検証や妥当性確認をすることも重要です。もし、ここで利害関係者の要求とは異なった方向に開発が向かおうとしているのであれば、開発を停止し、方向性を修正する必要があります。早い段階から利害関係者を巻き込んで開発を進めることは極めて重要なことです。

一方で、開発するべきシステムをブラックボックス（黒いために中身がよく分からない状況にある）として扱い、コンテキストレベルでシステムに要求されていることを明確化する過程では、システムの境界を決める必要があります。このとき、開発対象であるシステムの外部に別のシステムが存在します。これを外部システムと呼びますが、この外部システムと開発するべきシステムの間の関係性が重要となります。このコンテキストレベルでシステムと外部システムとの関係性が明確になると、システムの検証を行うために、何を用意する必要があるのかが分かります。SILS（Software In-the-Loop Simulation）、 HILS（Hardware In-the-Loop Simulation）でやるべきことは何か？を明確にする必要があります。

また、SysML の要求図では、要求とテストケースとを関連づけることができますし、その達成度まで管理することができます。テストケースが何を根拠に設定されているのかを明確にすることは重要です。もし、SILS、 HILS を行う部門で、開発対象である

システムの検証を行う段階で、何をどのように検証すれば良いかが定まっていないとしたら、それは、開発が失敗に近づいている可能性があります。

1.5. シリーズ化、ファミリー化、モジュール化のためにモデルの再利用性を考えること

製品やサービスの開発では、シリーズ化やファミリー化を考慮することが求められることがあります。簡単に言えば、製品やサービスに松・竹・梅のランクをつけることになります。こうしたラインナップを揃えるためには、システムのモジュール化が一つのキーポイントとなります。これを実現するためには、モジュール間のインタフェースが重要となります。更に、SoS (System of Systems)の中で、システムを投入しようという場合にも、外部にあるシステムと繋がるようにインタフェースを揃えることが重要となります。

特に、最近では、スマートグリッドやスマートシティのように、センサーや機器などの実世界に関与する物理的なモノが、地球上を覆う情報ネットワークで繋がっています。一瞬にしてこれらの情報が国境を越えて動くことができ、そして、これがビジネスと結びつきつつあります。エネルギーや医療、教育、交通などを管理する IT システムが実世界の様々な活動と緊密に結びついたシステムは CPS（Cyber -Physical Systems）と呼ばれます。そこでは、ある企業が開発したシステムが他のシステムと結合し、より大きなシステム、すなわち SoS を形成することになります。 SoS の世界では、その構成要素である各システムに期待されていることやその使われ方までもが、開発した企業の当初想定したものとは異なるものになることさえあります。すなわち、SoS の中でのシステム開発では、要求を定めること自体が極めて困難になっている場合があります。

モジュール化を進めることで、製品やサービスのバージョンアップや改良、または派生製品の開発などを容易にし、市場の環境変化に即座に対応できます。こうしたことに、システムモデルは大いに役立ちます。それは、システムズエンジニアリングプロセスで作成したモデルには再利用性があるからです。これは MBSE の特徴の一つになっています。3DCAD のソフトウェアでは、過去に設計した 3 次元の図面を、別の機会に再利用できます。これと同じように、SysML で作成したシステムモデルを、次のシステム開発の際に、再利用することができます。サブシステムをモジュールとして管理し、そのシステムモデルをリポジトリに保管しておけば、派生製品を開発する際や、ファミリー

を増やす際、シリーズ化を推進する際にシステムモデルを再利用できます。システムモデルの再利用性は、製品やサービスの迅速な開発やカスタマイズをもたらします[4] [5]。

2. 「システムズエンジニアリングについて」

本章では、MBSE について簡単に紹介します。冒頭で述べたように、MBSE は、開発の複雑さに対応するための技法であり、開発プロセスでもあります。また、MBSE は、拡張されたシステムズエンジニアリングと捉えることもできます。そこで、まず導入として 2.1 節では、MBSE の元となったシステムズエンジニアリングの概要を説明し、その上で「モデルベース」について考えます。2.2 節からは、簡単な例を用いて、MBSE への理解を深めます。まずは、システムをブラックボックスとして考えてシステムに求められる要求を明確化し、その後、システム内部に要求される機能を分解し繰り返し検討することでシステムアーキテクチャを構築していきます。このような一連の開発上流モデルの必要性を理解してもらいます。また、開発を通じて終始必要となる要求からの一貫したトレーサビリティについても触れています。

2.1. システムズエンジニアリングの概要

2.1.1. システムズエンジニアリングとは何か

システムとは、相互に関連があって全体として機能するコンポーネントの集まりと定義することができます。システムズエンジニアリングとは、システムの開発を成功裏に実現するための複数の分野にまたがるアプローチ及び手段です[1]。開発の初期の段階で利害関係者の要求を明確化し、機能要求などのシステム要求を定義し、関連する問題を全て考慮しながら設計のための統合とシステムの検証と妥当性確認を行います。システムズエンジニアリングは、利害関係者の要求に合致した品質の製品やサービスを提供することを目的とし、ビジネスと全ての顧客の技術的要求の両者を考慮します。製品やサービスを一つのシステムと考え、それを開発することをシステム開発と言いますが、システムのライフサイクルは、システム開発の始めから終わりまでばかりではなく、その運用、廃棄までを含みます。システムズエンジニアリングのカバーする範囲は、このシステムのライフサイクル全般にわたります。

2.1.2. モデルベースで考えることの意味

従来のシステムズエンジニアリングでは、複数の分野にまたがるシステム開発を、文書を中心としたやり取りで進めていこうとしていました。しかし、文書による表現には

曖昧性があり、いわゆる行間を読むようなことを読み手側に求めてしまうことが良くあります。逆に、できるだけ正確に文書で書こうとすると、その文書量は膨大なものとなってしまいます。

そこで、図的に表現することが提案されました。従来から、例えば、建築設計や機械設計の分野では、設計図を元に関係者が検討を繰り返しますし、制御工学の分野では、言わば設計図としてブロック線図により、信号の流れを記述して検討を進めます。そもそも、人が何かを語るときには、頭の中に図的なものを思い浮かべて、それを説明しています。人と人のコミュニケーションで、頭の中に思い浮かべた図をお互いに共有できれば、その意思疎通はうまく行きますが、もし、共有できなければ、意思疎通は成功しません。また、自分自身の考えを整理し、深掘りする場合にも図的表現は大いに役立ちます。

では、システムを図的に表現するとは、どういうことでしょうか？ 設計工学の分野では、ある製品の設計で、「価値」と「機能」と「構造」を考えることが重要であると以前から言われてきました[8]。QFD（Quality Function Deployment：品質機能展開）と呼ばれる方法は、設計する製品に求められる機能が何かを明確にし、その上で製品の構造を決めようとするもので、これにより、価値と機能と構造を結びつけることができます[9]。ある製品やサービスをシステムとして考える場合、同様に、価値と機能と構造が重要になります。 1 章にも書きましたが、要求には、利害関係者の要求、そこから導かれるシステム要求, 機能要求や、サブシステム要求、コンポーネント要求などがあります。利害関係者の要求やシステム要求などは、価値として解釈することができます。

設計工学で重視している価値と機能と構造は、システムズエンジニアリングでも、要求と機能と構造として重視しています。更に、機能には、「どのように振る舞うか」ということを含むことにも注意が必要です。また、製品やサービスの設計を行うには、パラメトリックな検討が極めて重要になります。例えば、トレードオフ分析を行うためには、何らかの評価を行うこととなります。そこでは、様々な設計パラメータを変化させて、評価しながら設計を進めていくことになります。このようなシステムモデル表現は、パラメトリック制約と呼ばれます。

システムを図的に表現する際、「要求」、「振る舞い」、「構造」、「パラメトリック制約」の 4 つの柱を用いることが重要と考えられます。SysML（Systems Modeling Language）は、これら 4 つの柱でシステムを記述することができるモデリング言語です。そして、

SysML は、システム開発の初期の段階に、利害関係者の要求を明確にし、アーキテクチャを選定し、システム要求を明確にするまでのプロセスのみならず、要求変更管理や構成管理、検証と妥当性確認のプロセスにも利用することができます。

2.1.3. 二元 V 字モデル[3]

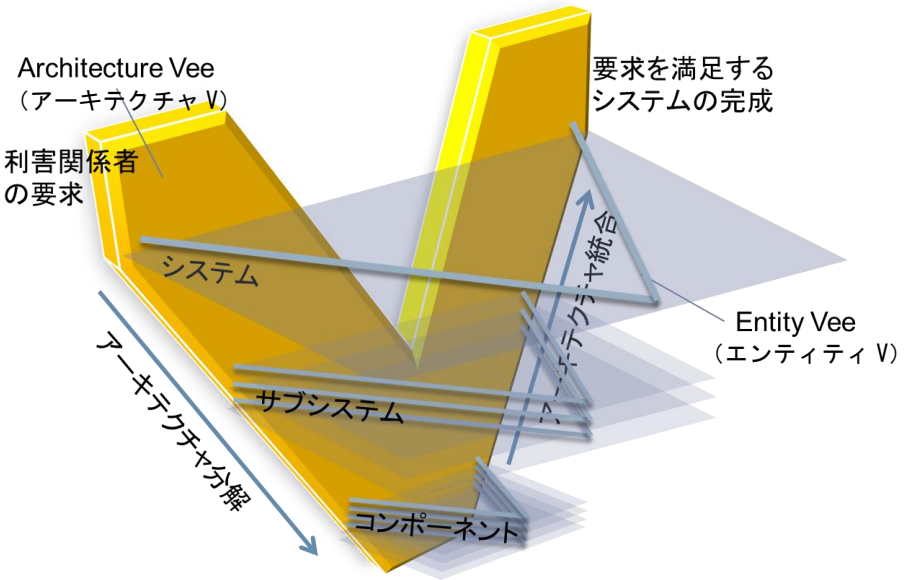

図 4　二元 V 字モデル[3]

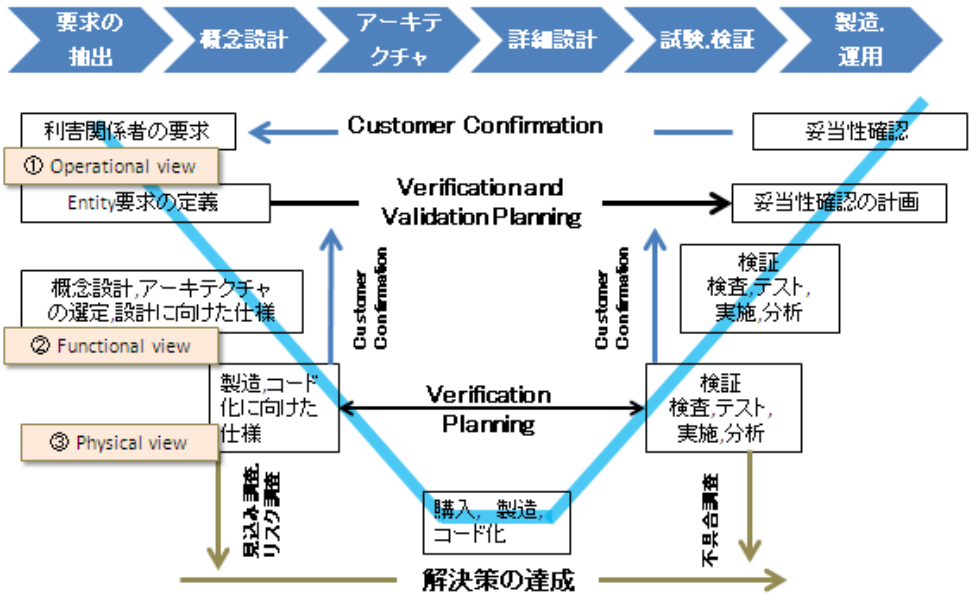

図 5　エンティティ V

1章で触れたように、システムズエンジニアリングプロセスは図 4に示す開発対象とするシステムの分解レベルを明示した二元V字モデルで表すことができます。この二元V字モデルは垂直方向のアーキテクチャVと水平方向に複数並んだエンティティVからなります。アーキテクチャVは、下に向かうに従い、開発対象のシステムが、システムレベルからハードウェアやソフトウェアなどの複数のサブシステム、コンポーネントへと分解されていくことを表します。そして、システム、サブシステム、コンポーネント、それぞれのレベルの開発時には、図 5に示すエンティティ V に表されたプロセスにしたがって開発が進むことを意味します。後述するとおり、これらのエンティティVの上下間でのやり取りが生じることもあります。

これらの図により、システムズエンジニアリングプロセス[1]全般を可視化することで、複数の分野にまたがるチームに対してこのプロセスの中でのそれぞれの活動を明確に示すことができます。例えば、最上位のシステムレベルにおいて、要求の明確化がなされ、機能が明確になり、機能の割り当てに基づくシステム全体のアーキテクチャが幾つかの候補の中から選定されると、サブシステムやコンポーネントレベルの選定がなされ、それぞれに対する仕様を明確にすることができます。これらの仕様はサブシステムレベルやコンポーネントレベルにおける要求となり、それぞれのエンティティ V に従って、開発プロセスが進められることになります。

アーキテクチャ V の最下層にあるコンポーネントが要求に対して検証・妥当性確認(Verification and Validation)が行われて完成すると、これらが統合されてサブシステムの検証及び妥当性確認が行われ、更にシステムレベルの検証及び妥当性確認が行われることになります。サブシステムやコンポーネントを統合し検証・妥当性確認する過程がアーキテクチャV及びエンティティVの右側のプロセスで行われます。

エンティティVの左側ではDAR(Decomposition Analysis and Resolution：分解の解析と決定)、 右側でVAR(Verification Analysis and Resolution：検証の解析と決定)が行われます。詳細は文献[3]に譲りますが、DAR では、システム開発における EDG（Engineering Design Gate：エンジニアリング設計ゲート)、PDR（Preliminary Design Review：事前設計レビュー)、CDR（Critical Design Review：最終設計レビュー）などの決定判断のための通過ポイントが設けられます。また、VAR では検証・妥当性確認における不具合が発生した場合の対応を明確にしています。その際、二元V字モデルのアーキテクチャVの縦方向、すなわち、例えばシステムとサブシステム間にやり取りが

生じ、繰り返し検討を重ねることがあります。こうした手戻りは開発の比較的初期の段階で意図されたものであれば、問題ない場合もありますが、QCD（Quality（品質）,Cost（コスト）, Delivery（納期））を守るためにも意図しない手戻りは極力減らしたいものです。例えば、要求の変更管理が必要となった場合、要求と検証のトレーサビリティが容易に確保できる SysML を用いることが有効となります。

システムズエンジニアリングプロセスでは、①Operational view（運用のビュー）、②Functional view（機能のビュー）、③Physical view（物理のビュー）の順に、利害関係者の要求分析からシステム要求を導き、概念設計やアーキテクチャの選定まで進めます。すなわち、①開発するべきシステムの運用シナリオ（ユースケース）を検討しこれを明らかにした上で、②システム要求の機能への分配を行うことで、システムに必要な機能を明確にし、そして、③その機能を実現するためにハードウェアやソフトウェアとして何が必要かを定めます。図 5 には①～③の view を明記しています。①Operational view で明らかにした運用シナリオは、最終的にシステムの完成を確認する際のテストケースの元となります。また、アーキテクチャの選定において、②機能アーキテクチャの次に③物理アーキテクチャを定めるフェーズでは、ハードウェアやソフトウェアなどのエレメントやコンポーネントへ機能要求を明確に割り当てることが重要です。物理アーキテクチャの一部が最初から仕様やシステム要求として与えられているときには、これは制約として扱われますが、いずれにしても機能の割り当てを行う必要があります。

2.2. MBSE の要求分析 - まずはブラックボックスとして考える

システムとは、相互に関連があって全体として機能するコンポーネントの集まりと定義されます。したがってシステムは、ある目的の機能を持つ必要があります。システム開発時には、こうした目的が要求として提示されます。そして、システムは、性能や、運用コスト等の達成すべき様々な品質を備える必要があり、これらも要求として提示されます。要求の元は開発するべきシステムの複数の利害関係者にあります。各利害関係者には、様々な観点からの要求があります。要求には様々な粒度があり、また様々な種類があります。

システムズエンジニアリングは、これらの様々な要求をまとめあげ、そこから技術的に実現可能なシステム仕様を構築します。ここで大事なことは、実現することだけではなく、実現すれば、ビジネスの成功が期待できるようなシステム仕様を構築することで

す。ですので、モノづくりの最上流の工程から、いわゆる QCD を強く意識しなければなりません。

近年開発されるシステムの多くはとても複雑で、高機能です。表面的には単機能の製品に見えても、実は、裏では様々な機能が含まれていることや、既存の製品に比べ、目に見えない品質が大幅に向上していることも珍しくはありません。また、スマートフォンやタブレットの流行からも分かるように、近年のユーザは、システムを自分なりにカスタマイズすることを好みます。それぞれのユーザは、様々なアプリケーションをネットからダウンロードして、様々に組み合わせます。同じ機能であっても、一人一人、それぞれが、気に入った色使いのものや、使いやすいと感じられるような別のアプリケーションを選択します。

一方、近年のシステムの多くは、インターネットなどを介して、それぞれが繋がり合っているため、同じエコシステムの中に存在する、と考えてよいことが多々あります。しかし、そうだからといって全てのデータが、全ての構成要素から閲覧可能である、ということは通常セキュリティやプライバシーの観点から望まれません。また、異なる複数のシステムが互いに複雑な関係をもつ SoS（System of Systems）などでは、あるシステムは、他のシステムと異なった寿命をもつことになります。これらのよい例が、スマートグリッドの一部である、充電池を搭載した自動車です。スマートグリッドシステムの中での自動車の役割はどうやって定義するのでしょうか？ スマートグリッドに繋がることができる自動車、という要求はどのようにまとめあげればよいのでしょうか？ そして、どうしたら親システムであるスマートグリッドシステムと子システムである自動車を、お互いにできるだけ依存、干渉することなく並行に開発することができるのでしょうか？

モノづくりの中心が完全にハードウェアであったころ、製品開発は常に CAD や機械部品表(メカニカル BOM (Bill of Materials))を起点として行われていました。ハードウェア開発チームは、CAD システムを使用して、要求を満たすと考えられるハードウェアを設計し、ソフトウェア開発チームは、効果的にハードウェアを動かすことができるようなコードを作成しようとしていました。CAD の設計データ及び 部品表 が製造部門に引き渡され、ハードウェアが完成し、最後に統合やテストを担当するエンジニアが、ソフトウェアを読み込み、製品全体をテストしていました。そして、このような開発は、シーケンシャルでウォーターフォール型であり、当時のシステム開発の複雑さや規模に

おいては、これが最も高速で安価な製品開発の方法でした。

今日の大規模で複雑なシステムの開発において、そのようなシーケンシャルかつ、文書のやり取りが基本となっているような製品開発プロセスで開発された製品は、市場で勝利することが困難になってきています。例えば、予測していなかったような市場での出来事や、競合製品に勝利するため、開発途中で突然生じた新しい機能要求などに素早く対応しようとしている場合を想像してください。それらの変更ごとに、要求文書を読み、過去の文書と見比べ、最初から開発工程を始め、シーケンシャルなプロセス全体を実施するといった時間のかかる方法をとっていたのでは、あっという間に競合他社に追い抜かれ、勝機を失い、市場から見捨てられてしまうことでしょう。例えば、SoS のようなシステムは､将来の発展の方向性の予測が困難です｡このようなシステムを開発し、ビジネスを成功させるためには、これまで以上に要求変更に対して迅速な対応が可能な開発プロセスが要求されます。その最初のステップであり、新しい開発プロセスの基盤となるのが MBSE です。MBSE は、様々な工学的領域、様々なチームにわたる開発の対象となる製品の要求とアーキテクチャを、モデルを用いて整理することで開発の複雑さを管理、低減しているのです。

それでは、MBSE のプロセスと成果物の概念を理解するため、簡単なエレベータシステムの例をとりあげていきましょう。

MBSE では、システムを要求、振る舞い、構造、パラメトリック制約の 4 つの側面からみたモデルとして捉えます。ここで、これらの活動を常に先導するのは要求の側面のモデルです。

要求の導出やまとめ方には、これまでも様々なやり方が提唱されてきましたが、MBSE では、SysML の要求図（Requirement Diagram）やユースケース図(Use Case Diagram)、ブロック定義図(Block Definition Diagram)、状態機械図(Statemachine Diagram)、アクティビティ図（Activity Diagram）、シーケンス図(Sequence Diagram)などを用います。要求を捉え、整理するために自然言語ではなく SysML を使用する最大の理由の一つは､要求をまとめた後で､アーキテクチャを設計､記述する際に､SysML を使用したいからです｡SysML の要求図を使用することで､いわば､ばらばらに提示された要求を構造化し、整理し、そこから更に必要に応じて要求を導出し、それら要求間の関係をまとめあげることが可能になります。そして、他のダイアグラムと連携して用

いることによって、開発工程の途中で要求の追加や、変更が発生した際に、設計、実装などに与える影響を、精緻に見積もれるようになり、これから修正、変更しなければならない要素を明確にすることが可能になります。また、それに引き続いて行われるテストや品質管理の工程を見積もることも容易になるなど、プロジェクトマネジメントの観点からも大いに役に立ちます。

一方、要求をまとめあげるということは、実際に動いているプロジェクトの役に立つだけではなく、今後の新規製品、派生製品の企画などにも有用です。通常、モノづくり企業において、一から製品開発を行うことは珍しく、通常は、規模の大小こそあれ、既存のハードウェアやソフトウェアの流用や再利用を行っています。何を再利用するか、ということも、開発者やビジネスの観点から上げられた要求であり、MBSE を通じて、その効果や影響を見極めることになります。

まず、あるシステムを開発するために最初にやらなくてはならないことは、そのシステムが「どのようなものであるか？（what)」を定義することです。これは、簡単なようですが、これを正確に定義して開発を始める企業は、意外とまだそれほど多くはないようです。通常、このような企業において、最初から「どのようにして？（How)」を定義しようとします。こうした要求分析の誤りによる障害は、システム完成時まで発見されません。

しかし、MBSE では、開発の可能な限り初期の段階で、要求分析の誤りを見い出すことを、大きな目的の一つとしています。そのためには、「骨となるような」大きな要求、抽象度の高い要求を見つけ出すことが重要です。

あるシステムが「どのようなものであるか？」を定義するというのは、簡単に言うと、システムを、利害関係者全員が理解できるようなブラックボックスとして考える、ということです。すなわち、MBSE では、まず、そのシステムを使用するユーザはどのような人たちか？　あるいはこのシステムを使用するのは外部の別のシステムであるか？そして、システムは、それらユーザにどのようなサービスを提供するのか？　それはどれくらいなのか？といったことについて、定義し、その内容を利害関係者と同意することからはじめます。ここでは、システムが提供する機能を記述するものとして、ユースケース図が有効で、また、これによりシステムのスコープ（範囲）が定義されます。これが、システム使用にあたっての背景や状況を分析するコンテキストレベルの作業の第一歩です。

以下にエレベータシステムのコンテキストレベルのユースケース図とブロック定義図の例を示します（図６、図７）。

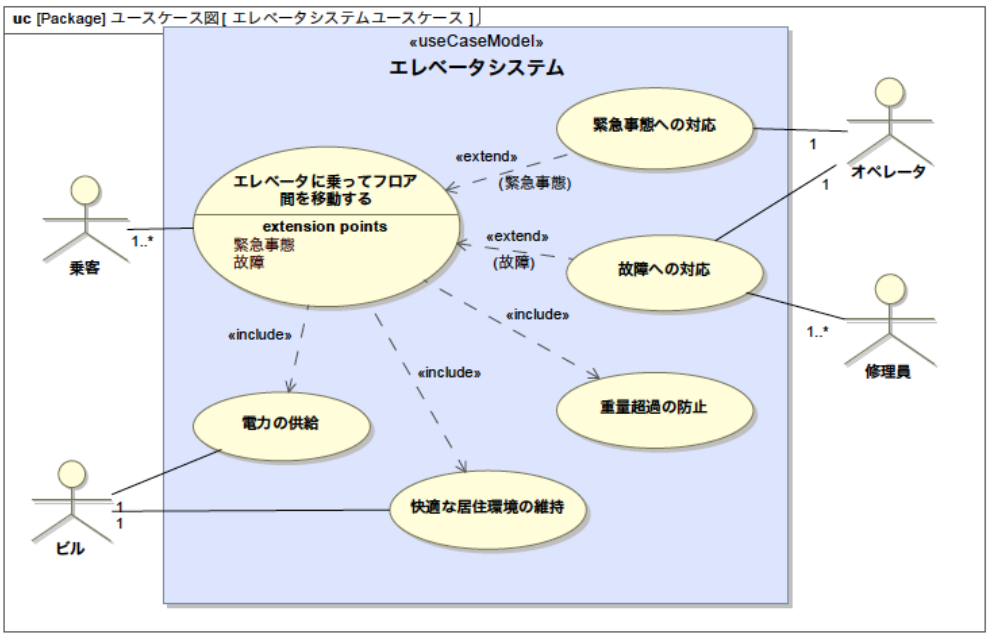

図　６　エレベータのコンテキストレベルのユースケース図

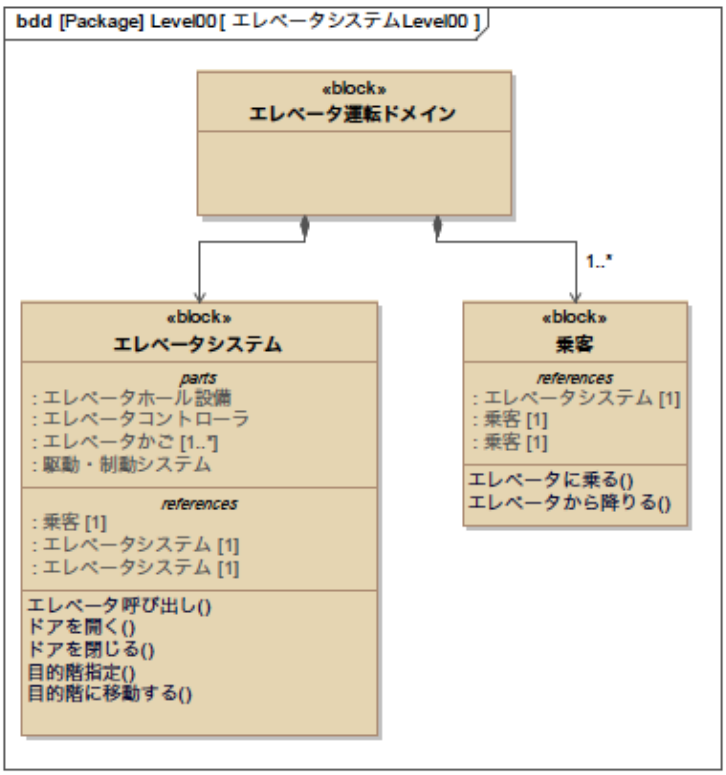

図　７　エレベータのコンテキストレベルのブロック定義図

これらの図は、ブラックボックスとしてのエレベータシステムを表しています。ユースケース図では開発すべきシステムのスコープ（範囲）を機能面から見極め、このブロック定義図では、エレベータ運転というコンテキストを定めています。すなわち、本例では、エレベータ運転を考えるときには、その文脈内に、修理員は登場しないこととしています。

システムが「大雑把に言えば、どのようなサービスを提供するか？」ということを、運用概念(Concept of operations: ConOps)ともいいます。運用概念は以下のとおりです。

・主な利害関係者のニーズ
・システムが提供する役割と責任
・全体的なシステムの能力
・求められている性能や品質の検証可能な尺度

システムをブラックボックスとして考えるときには、これらの運用概念について確認した上で、モデル化します。言い換えれば、モデルを把握するためのこれらの側面が、図５で述べた Operational view（運用のビュー）となる訳です。

簡単そうに見えるので、軽視してしまいがちですが、これはその後の要求分析の詳細化に繋がる大事な作業です。大事なことは、開発したいシステムはこれでよい、という合意のために使用した様々な資料が、その後の要求分析の成果物や、設計成果物の根拠として使い続けることができるということです。これがトレーサビリティです。言い換えれば、MBSE は、「トップにある要求という根拠に基づいて、トレースのとれる形で上流から途切れず、順番に開発を行う」トップダウンのエンジニアリング手法です。

ところで、どうやって要求を抜け漏れ重複なく、また誤解なく見つけ出せばよいのでしょうか。これは、簡単な問題ではありません。要求は、様々な関心をもつ様々な利害関係者たちによって、いろいろな言葉で表現され、そしてそれは様々な抽象度をもちます。また、要求の中には、それ以外の他の要求項目に依存しているものも多くあります。例えば遠隔操作できる自動車のおもちゃを考えてみましょう。以前は、遠隔操作と言えばプロポ(無線コントローラ　“プロポーショナル・ システム”の略称)を用いたラジコンでしたが、最近はその仕掛けにも様々な種類があります。また操作方法として、プロポのようなジョイスティックによる操作のみならず、ハンドルによる操作、近年ではスマートフォンで操作することなども考えられます。もし「小学生がうまく操縦できること」という要求があったとすれば、それは、明らかに遠隔操作の方式とおもちゃの応答

性という要求だけではなく、少なくとも、これらの様々なユーザインタフェースの要求にも関わる要求となるでしょう。

これら様々な側面をもった多くの要求を、完全に文書としてまとめ上げることは、百科事典を書き上げるくらい大変なことです。しかし、製品開発の目的は百科事典と同じ分量を書き上げることではなく、要求を満足する製品を出荷し、ビジネスに勝利することです。SysML の要求図も、後続するシステム開発を容易にするため、様々な粒度の要求をモデル化し、インデックス化し、構造化し、抜け漏れ重複や、誤解を減らすことを目的としていることを忘れてはなりません。

SysML によって記述されたエレベータシステムの初期の要求図の例を以下に示します。この例では、エレベータに対する要求を大きく、「フロア間の移動」、「電力の供給」、「快適な居住空間の維持」、「重量超過の防止」という、4 つの領域に切り分けていることが分かります。また、「フロア間の移動」という領域の要求が、「フロア呼び出し」、「ドア開閉」、「かご移動」の 3 つの要求から構成すると分析したことを示しています。

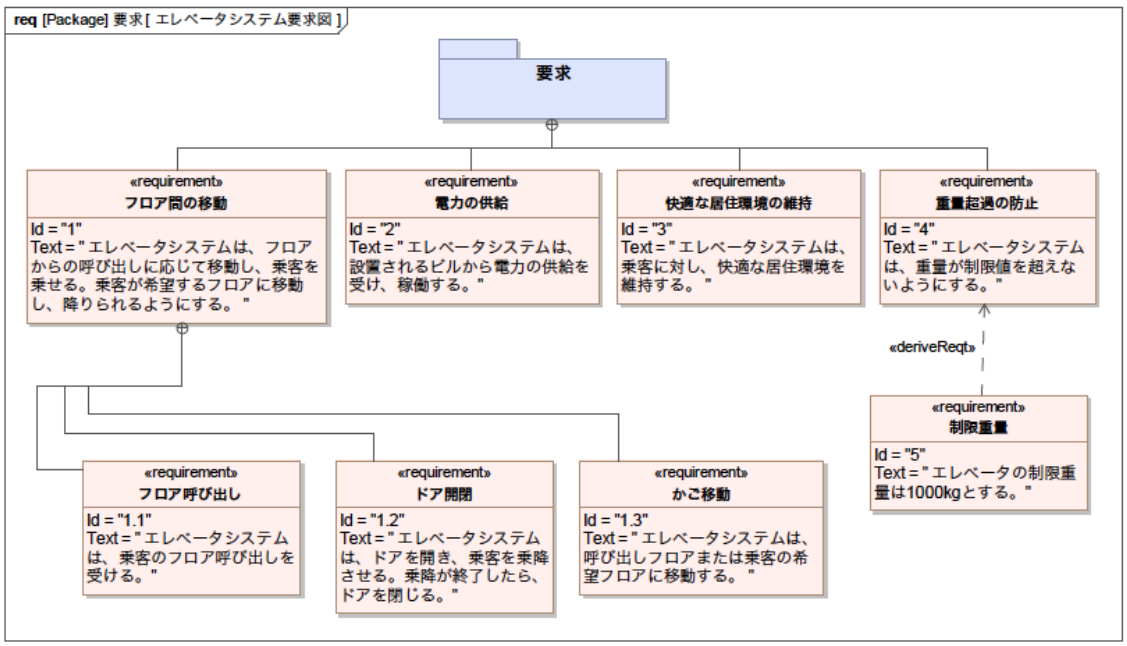

図 8　エレベータの要求図

この要求図は初期のものであり、まだ十分ではなく、この先、詳細化する必要があります。しかし、このようなフェーズであっても、恐れずにどんどんモデルを構築して行く必要があることに注意してください。ここで知っておかなくてはならないことは、手元に集ってくる数多くの要求は、様々な抽象度をもち、様々な種類があり、全ての要求

が揃ってからこれらを整理しようとしても、その作業はおそらく終わらなくなる、ということです。要求の整理は一筋縄ではできないので、手元にあるものからどんどん整理して行くことが重要であるということを理解してください。

整理する方法は様々です。要求には様々な種類があります。それらを分類するには、ハードウェア要求とソフトウェア要求という分類をすることもあります。機能要求、非機能要求といった分類軸もあります。機能性、信頼性、使用性、効率性、保守性、移植性といった品質の分類軸を使用する方法もあります。発見した要求を、例えば、これらの軸で分類して構造化してみるということも要求を取りまとめるための良い方法の一つです。また、User experience（ユーザーエクスペリエンス）やユーザビリティを考慮しながらユーザ要求を導き出す、人間中心設計（Human Centered Design, 以下 HCD と略す）のプロセスも、よりよいシステムを構築するためには有効と考えられています。（コラム参照）

コラム：要求の分析に「利用の状況」を考慮する

ここで、このユースケース図に HCD の観点を加えてみる。すなわち、引越しの現場では、どのような事象が起きているか、「利用の状況」を考察する。

引越し作業では、重く、量の大きな什器を運ぶため、積み込みや積み出しに時間がかかる。通常、エレベータの扉が開いてから、一定時間が経過すると、システムは人が乗り込んだことを予測し自動で扉を閉じるが、引越しの際はこれが災いして「開ボタン」を押し続けなければならない。エレベータが開発されて約 80 年後に、この解決策として「開延長ボタン」が登場した。

しかし、残念なことに現在は、ここまでの利用状況しか考えていないため、別のフロアでエレベータ待ちをしている乗客のことは考慮していない。すなわち、引越しの最中は他のフロア階の乗客はエレベータの待ち時間が長くなってしまう。既にお気づきの読者がいると思うが、「エレベータ待ち」と言えば、「満員」の状況も該当する。すなわち、このユースケース図には乗客が「一人」しか、描かれていないが、実態は複数人であり、場合によっては多人数（グループ行動中の一群）、車椅子利用者など、多岐に渡っている。従来ならば「ユーザは多様なので対応は困難」と片付けられてしまうが、共通する問題の本質が「待ち時間」の解消にあるならば、この視点に着目することで、開発すべき機能を設定し、技術目標を絞り込むことが可能となる。以下に例を示す。

・エレベータ待ちの人の存在を認識する技術（フロアセンサー、スマホアプリでの対応など）

・トータル待ち時間のカウント技術（同一人物による呼び出し押下回数のカウントなど）

・満員の定義のリアルタイム変更機能（待ち時間の長い人を優先的に運ぶために、空きスペースを含む満員状況の定義を設定するなど）

・諸状況に合わせた情報提供機能（満員状態でフロアを通過する際のアナウンスなど）

10 年前に、このような利用品質向上の発想があったとしても実現しなかったかもしれない。しかし、現在のネットワーク環境や IT 技術を駆使すれば、いずれも成立するような技術開発目標となり得る。例えば、乗客が乗り込んだ際に無常に鳴り響く、重量オーバーのブザー音を、状況によっては鳴らさずに済ませることはできないだろうか。乗客に恥ずかしい思いをさせずにスマートにエレベータを利用できるようにすることは容易なはずである。まさか、そんな時代が、これから、更に 80 年後だとすれば、エンジニア魂が廃る、というものである。

株式会社 U’eyes Design

代表取締役　鱗原晴彦

前に掲載したユースケース図（図 6）と、この要求図の関係を表すと図 9のようになります。ユースケースは、その名の通り「使い方」です。この図を見ると、「使い方」によって、漠とした自然言語による要求を詳細化（refine）していることが分かります。

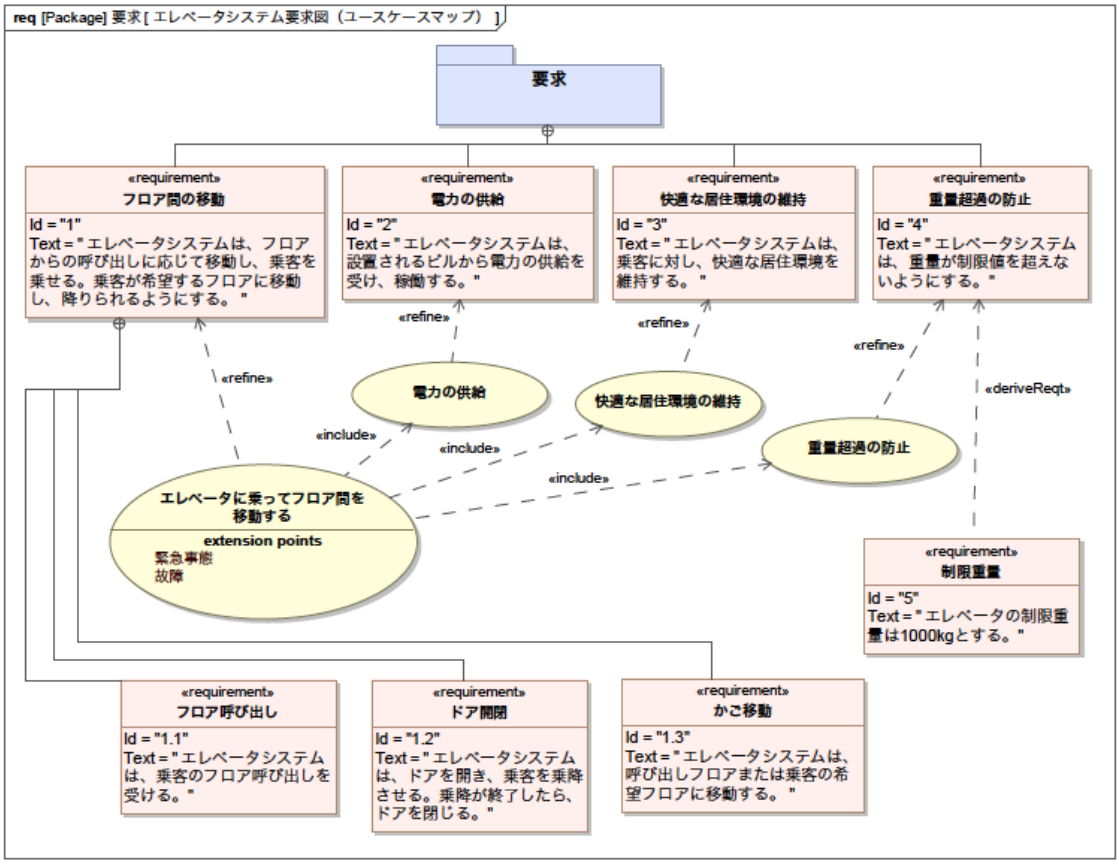

図 9　ユースケースを対応させた要求図

一旦決めた要求の全てを、開発中に一度も変えることなくモノづくりができることは、まずありません。したがって、もし、要求の追加や変更の依頼が起こったときに、その依頼を、根拠に基づいて、納得して受け入れる、あるいは納得できる理由をもって拒否できるようにしておく必要があります。また、要求変更があったとしても、可能な限り開発作業がロスなく進められるようにする必要もあります。このためには、これまでのある要求が、どのような根拠で、どのように分析されたのか、をすぐにトレースがとれるよう、要求を保存、管理できるようなリポジトリで一元管理しておくことが重要です。

SysML はシステムモデルを記述する言語にすぎません。しかし、SysML のような、共通言語を使用することで、利害関係者や開発者間で共通理解を得やすい成果物を構築することが可能です。そして、これらの成果物をどのように作り上げるのか？という技法とプロセスを示すのが MBSE と言えるでしょう。

システムのスコープ（範囲）が明確になれば、次にシステムが提供するサービスや機能の要求を詳細化することを考えます。言い換えれば、この視点が先に図５で説明した Functional view（機能のビュー）となります。ユーザや外部システムがシステムをどのように使用するのか？ということを明確化するこのフェーズでは、多くの場合、そのシステムを一つのブラックボックスとして考え、そのシステムが、ユーザや外部システム、そして環境などに対し、どのようにサービスや機能を提供するか？ あるいは、そこから影響を受けるかといった振る舞いを定義すると分かりやすくなります。MBSE ではこの際に、振る舞いの記述に特化したシーケンス図、アクティビティ図、状態機械図などを用いてシステムの振る舞いをモデル化します。

以下に、エレベータシステムにおける振る舞いのモデル化の例を示します。

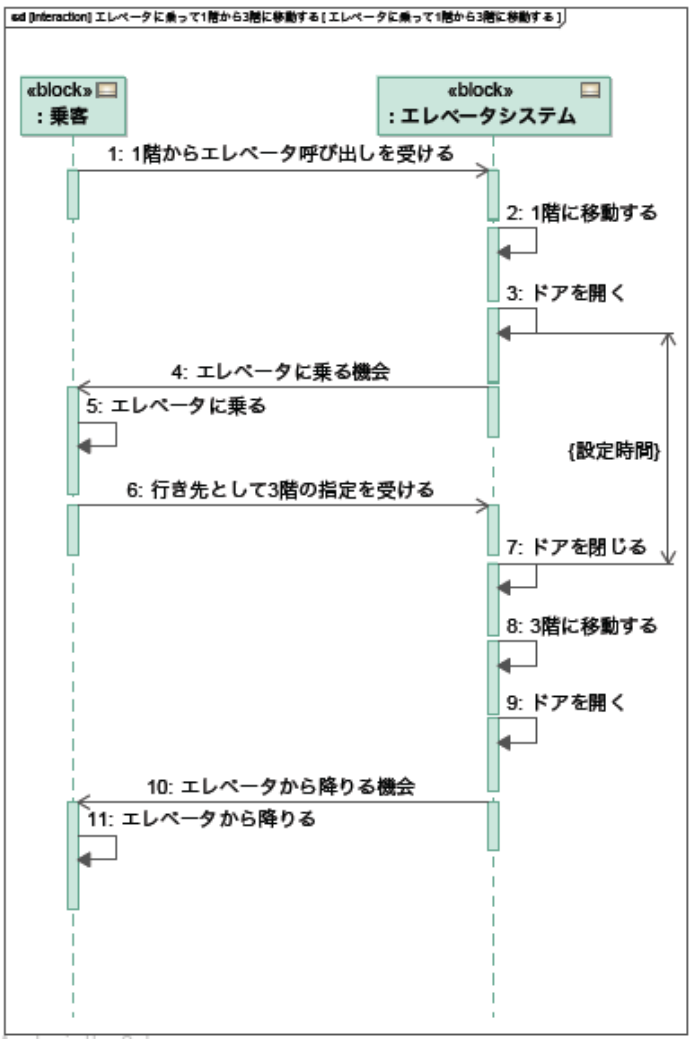

図 １０　エレベータのシーケンス図

このシーケンス図は、ユースケース「エレベータに乗ってフロアを移動する」をもう少し具体的にした「エレベータに乗って 1 階から 3 階に移動する」というシナリオを表現したものです。ここでは、一例のみの提示にとどめますが、このように、あえて具体的に振る舞いをモデル化したものを幾つか例として見比べてみることで、抽象的なシステム仕様が浮かび上がってきます。すなわち、ブラックボックスであるエレベータシステムに、どのような順番で、どのような入力をすると、どのような出力が得られるのか、ということを見極めます。そして、これが、システムレベルでのエレベータシステムの仕様となる訳です。

次の図 11 は、エレベータシステムのシステムレベルでの普遍的な振る舞いを表現した状態機械図です。これが、具体的なシーケンス図を色々と分析し、その中で明らかになってくるブラックボックスレベルのシステム仕様の例です。

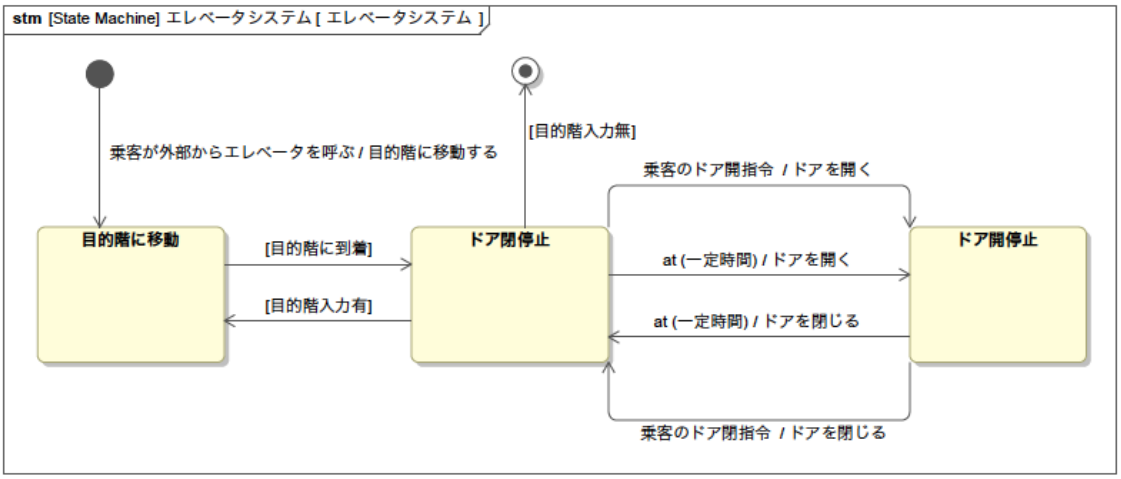

図 １１　エレベータの状態機械図

ここで注意すべき大事な点は、あくまでここで表現されているのは、ブラックボックスとしてのシステムがユーザや外部システムに提供するサービス、機能の振る舞いであるということです。すなわち、分析ではブラックボックスであるシステムの中身にどのような部品が入っているか、ということには全く触れておらず、分析内容がコンテキストレベルを超えてはいません。

MBSE では、要求をまとめた後、引き続きアーキテクチャの設計を行いますが、この際にはブラックボックスを開き、システム開発を担当する単位を意識したサブシステムという部品に分解します。そしてサブシステム同士がどのように相互作用し関係し合いながら、システムとしての振る舞いを提供するかについてもモデル化することとなりま

す。この際にも、振る舞いを表現するシーケンス図、アクティビティ図、状態機械図などを使用することになります。大事なことは、例えば同じシーケンス図であっても、要求の分析では、ユーザとブラックボックスとの相互作用の分析に使用され、後述するアーキテクチャの設計では、システムを構成するサブシステムやコンポーネント間の相互作用の分析に使用されているというところです。すなわち、要求の分析作業やアーキテクチャ記述作業の際、ある要素間の相互作用に伴う振る舞いという観点からシステムをモデル化したいという目的があるので、道具として「振る舞い」を記述しやすい SysML 記法のシーケンス図を使用しているという訳です。

ここで誤解のないように付け加えるならば、MBSE では、要求や、システムの構造、振る舞いなど、全てを SysML で記述すべし、と言っている訳ではありません。一例をあげれば、納得できる要求を取りまとめるときや、分析するために、SysML 以外の記法を補足のために使用することが有用な場合も多くあります。例えば、年齢別のユーザの嗜好や、業界のこれまでの慣習に基づくようなロジカルではないシステムプロパティのようなものを記述するためには、スプレッドシートを使用した対応表が便利です。また、フィードバック制御を用いたコントローラのように、コンピュータシミュレーションを行いながらその性能について詳細化していく際に用いる非常に複雑な数式も、明らかにシステムのある側面を記述するためのモデルです。

また、モデル化される要求の元を管理するために、要求管理ツールや、スプレッドシートを使用しても一向に構いません。大事なことは、ある設計の根拠がどの要求にあるのかを明確にできる、言い換えれば、現在の要求と現在の設計、変更後の要求と変更後の設計をうまくトレースがとれるようにしておくことです。

MBSE は、このように、システムに求められる要求をモデルとしてまとめあげる、すなわち SysML を用いて、様々な関心事に関する様々な要求を構造化し、見通しを良くし、一元的に管理するところから始まります。

2.3. MBSE のアーキテクチャ設計 - ブラックボックスを分解する

前述したように、システムズエンジニアリングは、様々な要求をまとめあげ、技術的に実現可能で、それを実現すれば、ビジネスの成功が期待できるシステムの仕様を構築することを目的としています。

基本的に MBSE の目的は、これまでシステムズエンジニアリングが目指してきたも

のと同様です。ただし、これまでシステムズエンジニアリングの成果物の多くが自然言語の文書としてまとめあげられてきたのに対し、MBSE では、その名の通りモデルを用いてまとめあげられています。

ここでいうモデルとは、システムをある側面から見て抽象化したもの、言い換えれば、システムに関する何らかの特徴や性質を説明するための簡易化のことです。そして、共通言語である SysML を用いることで、一般性をもってシステムのある側面を記述することが可能となります。

前節では、まず、要求をまとめながら、システムをブラックボックスとして考えました。誤解を恐れずに言えば、このコンテキストレベルにあるブラックボックスを、開発しやすいように分解するのが、次のレベルである分析（アナリシス）レベルで行うアーキテクチャ設計になります。ここで重要なことは、前のコンテキストレベルの作業とのトレーサビリティをうまくとりながら、この作業結果に論理的に矛盾することなく、要求が実現できるよう、設計要素にブラックボックスを分解することで、システムのモデルを詳細化するということになります。

それでは、エレベータシステムの例を用いてシステムアーキテクチャのモデル化を試みましょう。

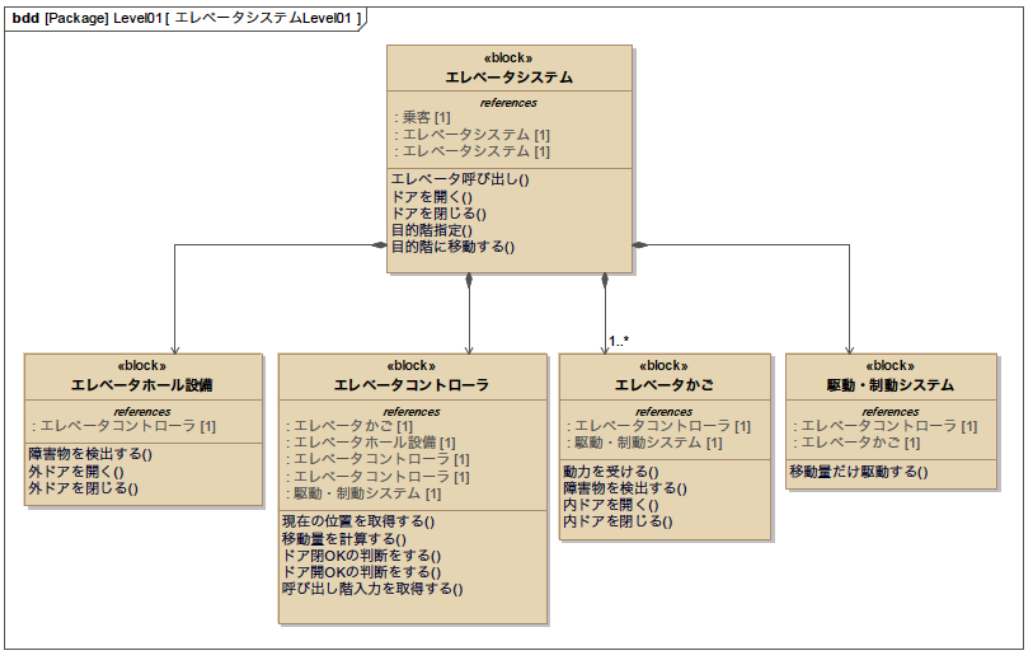

図 １２　ブロック定義図：エレベータシステムのサブシステム分解

アーキテクチャ設計では、まずシステムを、アーキテクチャ構成要素であるサブシステムに分解して行きます。例に挙げたブロック定義図（図　１２）ではエレベータシステムを４つに分解されたサブシステムによって構成されることを示しています。

また、ブロック定義図で表現されるのは、全体-部品の構造だけですので、次に内部ブロック図（Internal Block Definition Diagram）で、それぞれの部品間の関連、どのサブシステムとどのサブシステムにインタフェースがあるか、ということを定義します。すなわち、システムは、構成しているサブシステム同士がこの関連線（コネクタパス）に沿って相互作用し、互いに関連しながら、システムとしてのサービスを提供すると考えます。この例では、白抜きの四角で表されたポートを使用しています。ポートはブロックの属性で、情報のインタフェースのみならず、エネルギーや、物理的なモノなど、ブロックに対して、どのような「フロー」が流出入する接続点があるのか、を定義します。ポートを定義することで、各ブロックの責務が明確になると同時に、ブロックの再利用の可能性が大きく高まります。

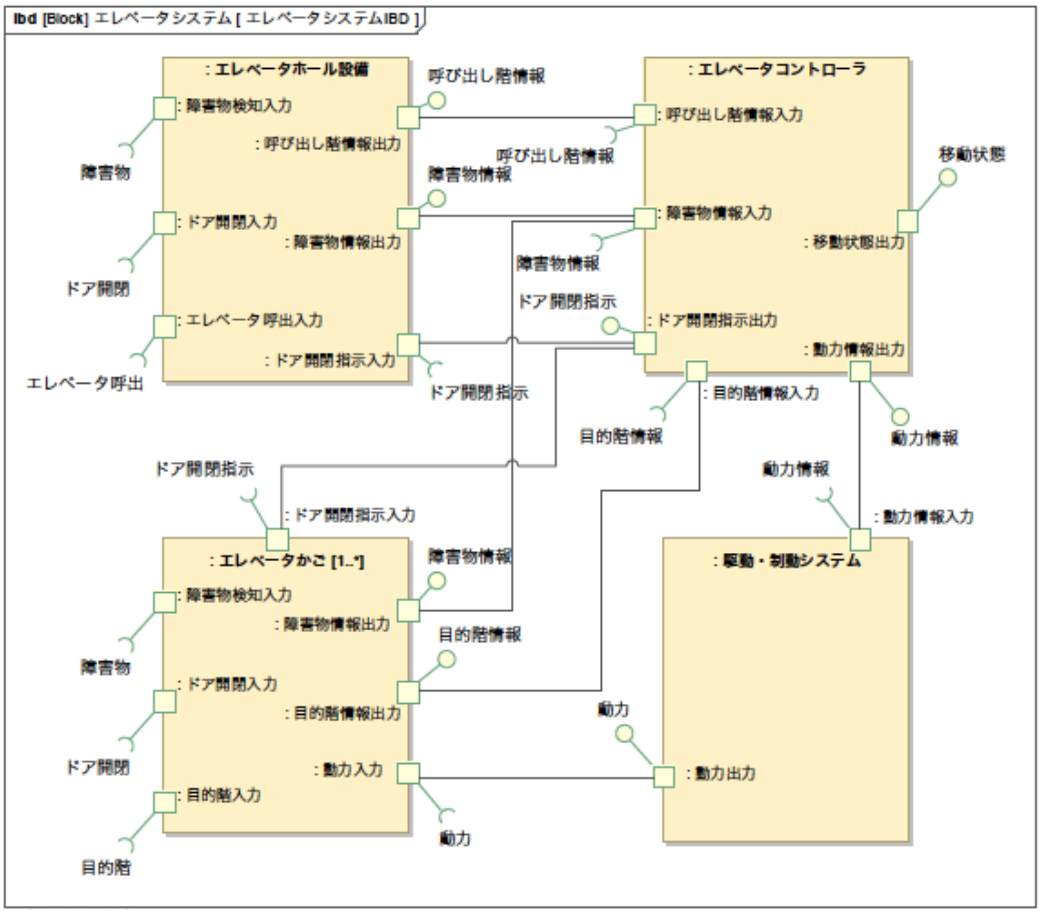

図　１３　内部ブロック図：サブシステム間のインタフェース

ここで、これまで、ただ一つのブラックボックスであったエレベータシステムが、相互に関連し合う、複数のサブシステムに分解されました。これらが構造のモデルです。このモデルによれば、「エレベータホール設備」サブシステムは「エレベータコントローラ」サブシステムに接続され、「駆動・制動システム」とは直接やり取りを行わないことが分かります。

以上のように、アーキテクチャ構造モデルは、システムアーキテクチャ要素の階層構造を捉え、各要素間の関係性を示します。しかし、これらの単純な図では、要素間の関係は示していますが、実際にどのようにやり取りを行うのか、相互作用がどのように行われているのか、が分かりません。そこで相互作用については、次に説明するような振る舞いモデルを用いて表現します。

前掲した「エレベータに乗ってフロア間を移動する」というシステムレベルのユースケースの実現も、サブシステムの相互作用によって達成されるはずです。そこで、今度は、シーケンス図「エレベータに乗ってフロア間を移動する」を分解します。分解には様々な方法がありますが、ここでは、まず最初に、縦方向の分解といわれるシナリオのサブ機能分解を行います。

図　１４　シーケンス図：シナリオの縦方向の分解（サブ機能への分解）

次に、横方向分解といわれるサブシステムへの分解です。ここではアクティビティ図を使用してみましょう。

アクティビティ図は、よく使用されるフローチャートあるいは FFBD（Functional Flow Block Diagram）を拡張したもの、と考えてよいでしょう。振る舞いを、「アクティビティ」の連携により表現することが可能です。以下の例では、「目的階に移動する」という部分的なシーケンス図を、アクティビティ図を用いて分解しています。各アクティビティを、それぞれサブシステムを表現する 4 本の区画（スイムレーン）に割り当てます。この割り当てを実行することで、各サブシステムの責務と、関係のある他サブシステムとのインタフェースが明確になってくることが分かります。例えば、この例では、エレベータコントローラサブシステムは、このシナリオを実現するために、「停止階を決定する」、「エレベータを移動させる」、「エレベータの移動状態を出力する」という 3 つのアクティビティと、「エレベータホール設備」、「エレベータかご」、「駆動・制動システム」とのインタフェースを備える必要があることが分かります。

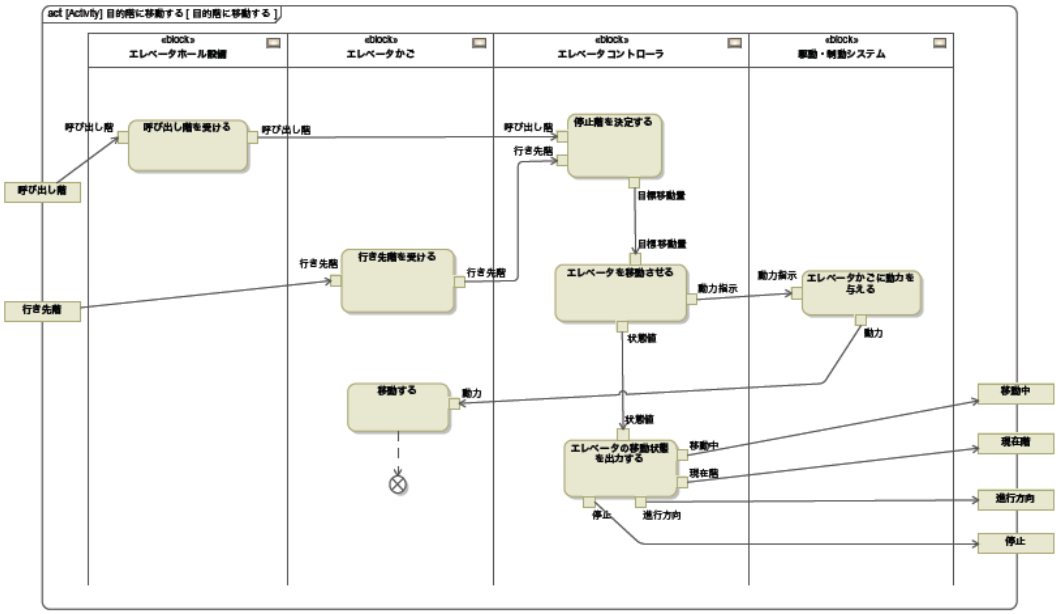

図 １５　アクティビティ図：シナリオの横方向分解

この同じ振る舞いをシーケンス図で表現すれば図 16 のようになります。アクティビティ図では、振る舞いの内容に主眼がおかれ、シーケンス図では、振る舞いの順序に主眼がおかれていることが分かります。ここでは、理解のため、アクティビティ図とシーケンス図を用いましたが、システム分析で用いたように状態機械図で、各サブシステムの責務を分析することも可能です。

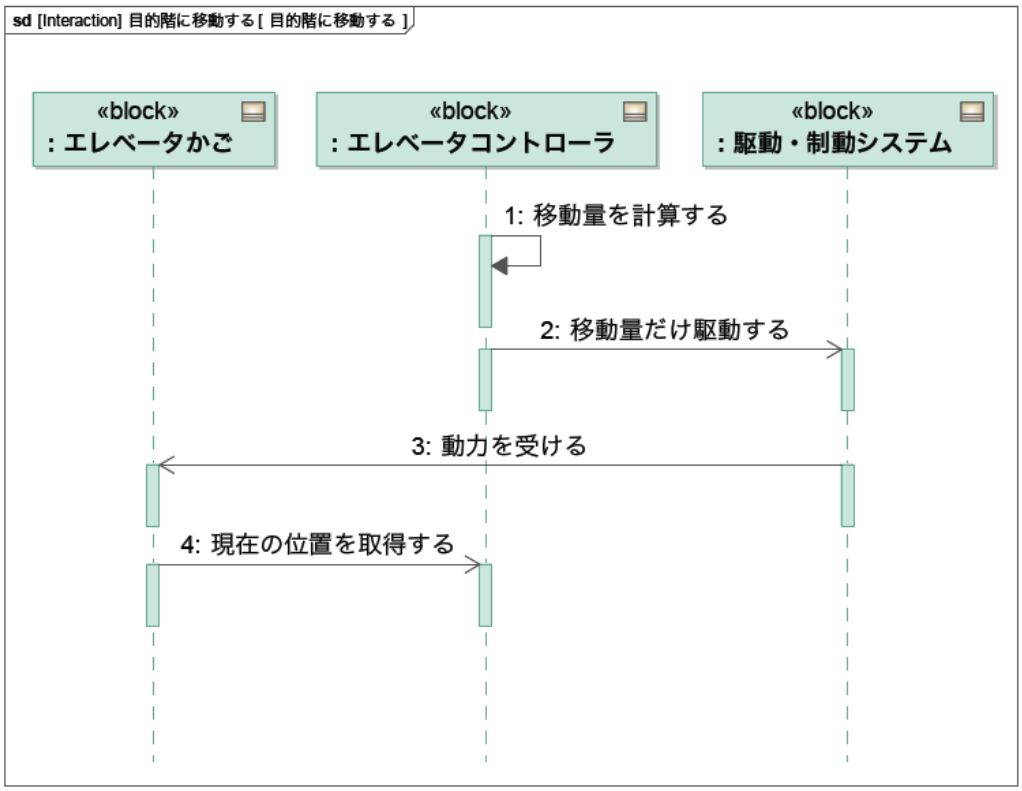

図 １６　シーケンス図：目的階に移動する

このように、振る舞いのモデルを使用することで、アーキテクチャを構成するサブシステム同士がどのように相互作用して、システムが提供するサービスや機能を実現しているかということが分かります。このようにシステムをサブシステムへ分解し、そこへ機能を割り当て、他サブシステムとのインタフェースを明確にするということは、システムを矛盾なく、物理的に異なるサブシステムに分割するということです。プロジェクト管理の視点からは、これ以降、各サブシステム単位で独立して開発作業を進捗可能であるということです。これが図５で述べた Physical view（物理のビュー）の視点となります。

ここで、MBSE の過程をおさらいしてみましょう。

MBSE は従来のシステムズエンジニアリングの拡張です。システムズエンジニアリングがやろうとしているプロセスは、入れ子になったシステムをトップダウンで分解し、開発するためのアプローチです。簡単に言うとシステムの全体像を浮かび上がらせ、アーキテクチャ設計というシステムレベルでの設計を行うことです。まずは、システムが全体としてどのように機能し、振る舞うか、ユーザや他のシステムとどのように関わる

かを考えます。そしてシステムを構成するサブシステムには、どのようなものがあり得るか、サブシステム同士が、どのように相互作用し合い、コラボレーションするのかを考えます。また、コンカレント開発を進めている様々なエンジニアリング分野の作業成果をどのようにまとめるか、といったことも考慮しなくてはなりません。

本手引きのエレベータの例で言えば、まずはシステム設計での最上位のレベルにあるシステムレベル（例えばエレベータそのもの）から始めます。コンテキストレベルのモデルと使用に関するモデルを構築した後、システム要求に基づいて、システムレベルでのアーキテクチャの構造モデル及び振る舞いモデルを定義します。ここで、モデルが確かに、要求段階で定義したシステムの仕様を実現できることを確認します（実証するためにモデルを実行することもあります)。システムレベルに対してプロセスを完了した後、次のレベルであるサブシステムレベルでそのプロセスを繰り返します。大きなブラックボックスを分析し、一つのブラックボックスを、その中身となるサブシステムに分解するということです。

上記のモデル実行や、ツールにあるシミュレーション機能は、開発の上流工程の作業において、要求の妥当性確認や、それに対応する設計の検証を、開発の後期の実機テスト工程まで待たずに、モデルで確認可能とするものです。これによって、手戻りを激減させ、大幅な開発コスト削減も可能となるのです。

そして、ここで得られた設計の検証が完了しているシステムアーキテクチャの要素、すなわちサブシステムを開発するグループは、改めてこれをブラックボックスとみなして、再度 MBSE を適用し、更に分解、詳細化していきます。すなわち、サブシステムへの要求を最上位のシステム要求と見立てた同様のプロセスを行うことで、論理的に矛盾のない設計の詳細化を進めることが可能となります。そして、次々と分解のレベルを掘り進み､各レベルでコンテキストを考慮しながら､モデルを用いた詳細化を進めます。このように MBSE のプロセスは必然的に、それぞれの作業レベルにおける最上位要素の分析、上位要素を下位要素へ分解することによる設計、そして、新たなレベルにおける要求の妥当性確認や設計の検証というステップを繰り返すことになります。

それでは、MBSE では、どの程度まで分解をすればよいのでしょう。本手引きでは紙面の都合上、十分に紹介できませんが、通常、最後の完了レベルは、コンポーネントレベルです。コンポーネントという言葉は、業界や、工学領域によって様々な使い方をされる言葉ですが、基本的には、電子設計、ソフトウェア設計、または機械設計など、そ

れぞれの工学領域で実現する対象を表現した抽象度の高いブラックボックスのことを言っています。しかし現実的には、このレベルは様々であるようです。例えば、自動車業界においては、コンポーネントがハードウェア／ソフトウェアの混在した「CPU の入った箱」である ECU であったり、ソフトウェア開発中心の開発であれば、「概念的なソフトウェア」であるフレームワーク要素であったりするようです。すなわち、後続する開発チームが、どのような「システム仕様」を必要としているか、ということに依存するのです。

利害関係者の様々な要求を実現するシステム全体を、たった一枚の設計図で表現できる訳がありません。「どのようなシステムか？」を人に伝えたいときには、MBSE では、関連し合った異なる種類の複数のモデルを用いて、様々な側面からシステムを捉えることを試みます。モデルとは、実現したいシステムにおいて、ある関心事がどのようにそのシステムで実現されるかということについて、抽象的に示したものです。基本的には、MBSE のダイアグラムの一枚一枚が、システムをある側面＝関心事からみたモデルだと考えてよいでしょう。

逆に言えば、システムを表現した様々なモデルの全てをまとめればシステムの全容が明らかになるよう、一貫性をもったモデル群を構築しなくてはなりません。そのために、それぞれの成果物間での意味のミスマッチが起きないように、文法とセマンティクスが定義され、標準化された記法である SysML を用いると良いのです。もちろん、記法だけでなく、モデルの一貫性を機械的に維持し、管理するため、ツールを使用してモデルのリポジトリを維持することも重要です。

また、市販されているツールによっては UML 同様、SysML のモデルレベルでシステムの振る舞いモデルの実行が可能なものもあります。このようなツールを使用することで、システムのアーキテクチャレベルでのモデル実行を通じた検証も可能になります。これも SysML のような、標準化された文法やセマンティクスを使用することによる大きな利点です。

これまでに要求、構造、振る舞いのモデル化を簡単に紹介してきましたが、SysML にはもう一つ特徴的な柱となるモデルがあります。それがパラメトリック図（Parametric Diagram）を用いた制約のモデルです。パラメトリック図は性能や信頼性、物理的な性質といった、システムに対する制約を記述します。例えば、システムの重量制限や電力消費量などを表現可能です。ツールによっては、これら制約を計算する

ことや、シミュレーションすることが可能です。例えば、連続系の制御則のモデル化や、複数の技術的な選択肢から一つの技術を選択する際の尺度の構築といったときに使用します。

要求分析の観点で言えば、制約は通常、パラメトリック要求として扱われます。以下の例は、エレベータシステムのパラメトリック要求を表した要求図です。

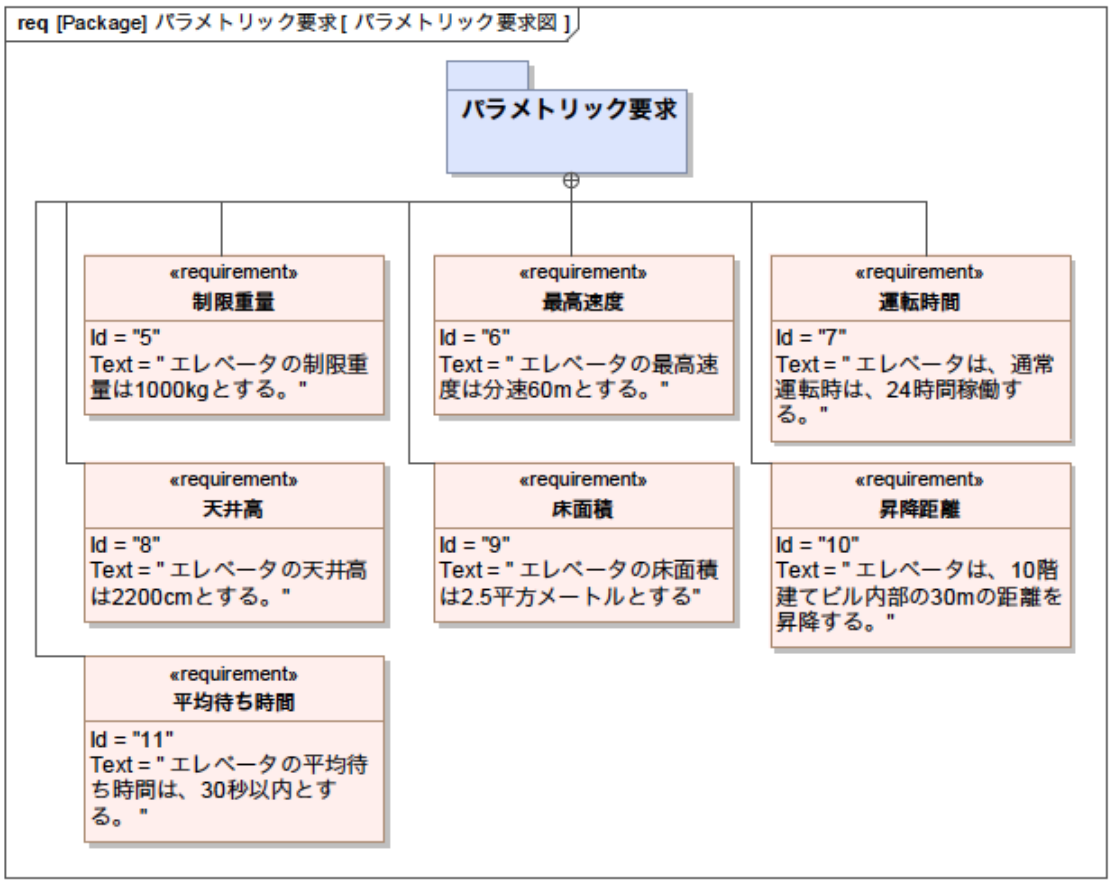

図 １７　エレベータのパラメトリック要求図

これらのようなパラメトリック要求を組み合わせて、新たな制約を導くときにパラメトリック図は有効です。例えば、エレベータの最高速度の理論値は、エレベータかごの重量や駆動・制御システムの牽引力などから導き出されます。そして、かごの重量はかごの材質や物理的な設計などから、牽引力は駆動装置への供給可能電力やモーターの出力などに依存します。パラメトリック図は、このようなシステムの様々な特性間のパラメトリックな関係を示します。

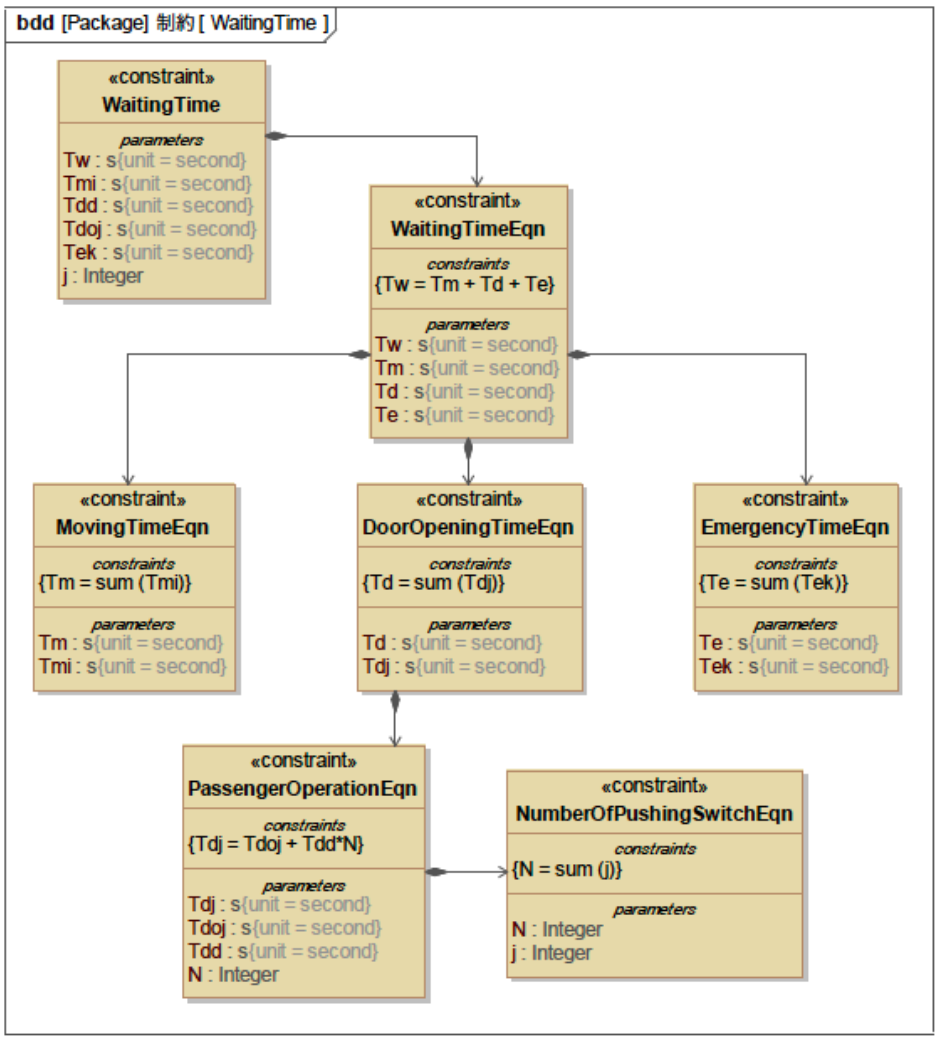

図 １８　「エレベータ待ち時間」のブロック定義図

ここでは、「エレベータ待ち時間」のブロック定義図（図 １８）とパラメトリック図（図 １９）を例示します。待ち時間は、どのような時間から成っているのかがブロック定義図で表され、それぞれ、どのようなパラメータをもち、そしてどのような制約（式）で関係しているのか、がパラメトリック図で示されています。図 １８では、乗客がある階でエレベータを呼んでから、その階にエレベータが到着するまでの「エレベータの待ち時間」が、「かごの移動時間」と「ドアの開時間」と「緊急時対応時間」からなるものとしています。エレベータが呼ばれてから到着するまでの間に途中階では他の乗客の乗降がありますし、その際に、乗客はドア開の操作をするかも知れませんし、場合によっては、ドアへの挟み込みを防ぐためのセンサーが作動し、ドア開閉の緊急停止をするかも知れません。

図 １９のパラメトリック図は、これらをパラメトリックな関係性で表しています。j 回目のドア開操作によりドアが開いている時間 Tdj は、乗客によるドア開の操作が行われている時間 Tdoj に加えて、ドア開操作終了後、インターバル時間 Tdd をおいてドアが閉まります。また、ドア開閉の緊急停止は、センサーの感度によっても、その発生回数(k 回)やドアの開閉停止時間 Tek が変化します。更に、インターバル時間 Tdd と緊急時におけるドアの開閉停止時間 Tek の間には、トレードオフの関係があるかも知れません。

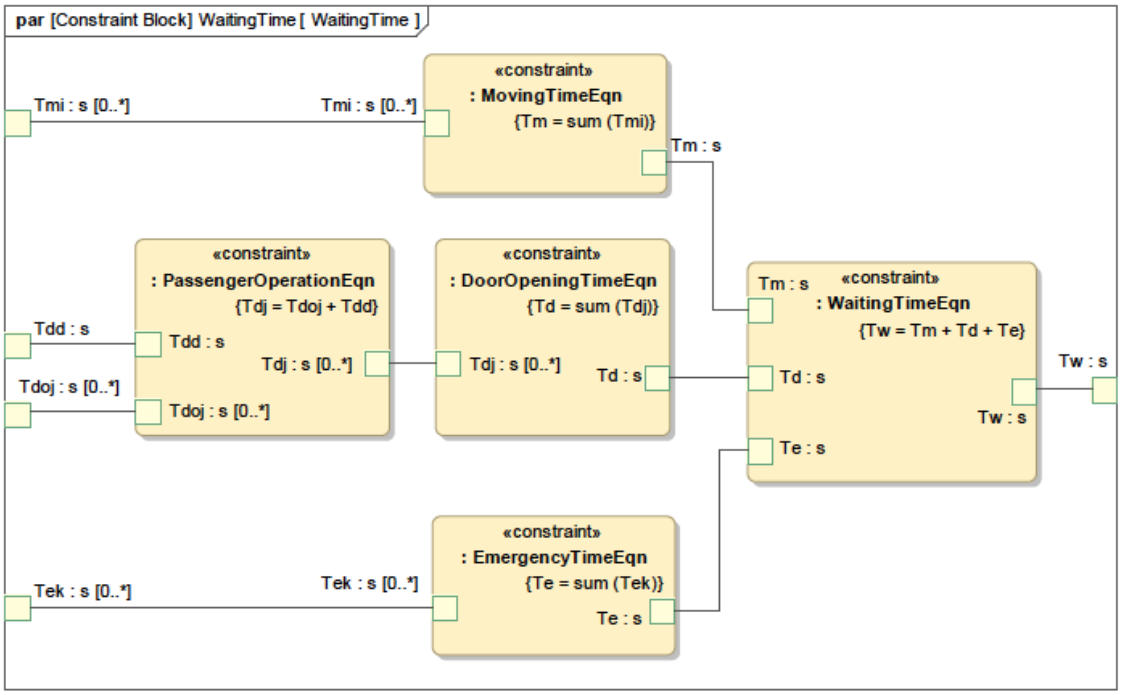

図 １９　「エレベータ待ち時間」のパラメトリック図

すなわち、インターバル時間 Tdd を長めに設定しておくと、ドアの開閉を停止させるような緊急事態が起きにくくなる可能性がありますので、これらのパラメータを調整することで、エレベータの待ち時間を短縮できるかも知れません。しかしながら、エレベータの待ち時間をトータルに短縮できたとしても、ドア開操作の後、ドアが閉まるまでのインターバル時間が長いと、操作の反応が遅いことについて、乗客は不快に感じるかも知れません。エレベータの待ち時間もドア開操作後のインターバル時間も、ともにユーザビリティに関連する性能ですが、トレードオフの関係をもつ可能性があります。

図 １９のパラメトリック図を基に、モンテカルロシミュレーションを行うことで、平均エレベータ待ち時間を算出することができます。設計パラメータとして、インターバ

ル時間 Tdd やドアセンサー感度を調整することで、エレベータ待ち時間がどのように影響を受けるかをシミュレーションし、それに基づき、設計パラメータを決定することができます。

このようにパラメトリック図で表現された制約は、理解性も高く、最適な設計を進めて行く過程や、ある評価指標を表す場合に有効です。また、具体的な数値演算やシミュレーションを行うことを表現するのにも適しています。

ある要求を実現するやり方は、ただ一つとは限りません。しかし、通常は、幾つもの設計上の可能性のうちから、解として一つの設計を選択することになります。そして、どうしてそれを選択したのか、という根拠を、うまく他人に説明できるでしょうか？ 例えば、前節で挙げた遠隔操作するおもちゃの UI（ジョイスティック/ハンドル/スマートフォン）は、最終的にどれを選択すべきなのでしょうか？ 開発の容易さや、ビジネスのキーファクタ、市場の流行など、それを決定する要因は一つではありません。しかし、開発のためには、どれかに決定しなければ仕様は定まらず、開発は不可能です。

これらの決定をするために行う作業がトレードオフ分析です。トレードオフ分析は、複数のものごと、ここでは複数の遠隔操作操縦の UI から、どれか（全部でも良い）を選択するための根拠、尺度を作り、比較します。

エレベータシステムの例を用いてトレードオフ分析を考えてみましょう。

近年のエレベータでは、消費電力に関しての要求が高まってきています。それに伴い、最近のエレベータでは、モーターをきめ細かく制御可能なインバーター技術が採用されています。しかし、アクチュエータを細かく制御することが省エネの全てではありません。エレベータかご内の照明は、蛍光灯なのか、LED なのか？といったユーザの目につきやすい省エネ技術もあります。 また、かごの軽量化や、モーターの小型化などの新技術も省エネに大きく寄与します。しかし、製品化まで考えれば、これら新技術の導入は、そう簡単ではないことがすぐにお分かりだと思います。そもそもモーターの大きさは、機械室の大きさにも依存します。機械室の大きさや位置は、設置される建物のデザインによって左右されます。近年ではブレーキ時のエネルギーを回生する技術などもあり、近年の消費電力に関する要求は、とても複雑なものとなっているのです。それでは製品としては、これらのどの技術やパラメータを組み合わせればよいのでしょうか？ カット＆トライ、スクラップ＆ビルドでこれらを制御するソフトウェアを作っていたのでは、開発コストが膨大なものとなってしまいます。したがって、部品の供給は安定している

か？ コストのバランスはとれているか？ なども含め、様々な角度から、全体最適を行いながら設計を決定するために、要求に対して、ある設計の効果を測定する尺度（MOE, Measure of Effectiveness）を設定し、設計判断を導く活動が、トレードオフ分析です。パラメータを変更しながら数値シミュレーションすることを容易にするパラメトリック図は、トレードオフ分析のためにも重要です。

トレードオフ分析そのものの技法は様々ですので、ここでその詳細については省略します。ただし、特に、安全性や信頼性の法規や標準に準拠する必要があり、プロセスとしてトレーサビリティの確保が必須な製品開発では、「XX という理由で YY という要求が決定した」「YY という要求があって ZZ という設計が行われた」というそれぞれの根拠とそこから導きだされた開発の決定事項を記録しておく必要があります。この観点からみれば、トレードオフ分析は、設計上の判断の根拠を作る作業といってよいでしょう。

ここまで駆け足で MBSE の紹介をしてきました。現在、実際の産業界で導入実績があり、参考としやすく、資料が入手しやすい MBSE プロセスとしては、OOSEM（Object-Oriented Systems Engineering Method）[4] [5]や Harmony-SE[12]、SYSMOD など[13] [14]があります。しかし、基本的な目的や成果物レベルで違いはなく、本質に変わりはないと考えてよいでしょう。

MBSE のプロセスは、効率的なシステム開発を支援することを目的とし、モデルを利用して、要求をまとめ、それを実現するアーキテクチャを設計します。そして、成果物として得られたモデルを使用した設計情報が、開発の次工程である各工学領域チームへの入力となります。一方、MBSE の技法面について誤解を恐れずに簡単に言えば、モデルを使用することにより、ある根拠のもとに、システムのブラックボックスを、その後の作業が容易となるような手頃な大きさに分解する技法、検査する技法、そして統合する技法です。

2.4. 品質の確保について

製品の品質を確保するための第一歩は、その品質が満たされていることを確認できるようになっているかどうか？ ということです。すなわち、品質要求を検査する際に、その品質をどうやって測定するか？ また、可能な合格点はどのくらいであるかといった尺度を定義しておく必要があります。当たり前のことですが、合格点がとれたシステムだけが、求められた品質を満たしている製品なのです。

一般的な製品開発プロセスにおいて、MBSE が適用されるのは、いわゆる V 字プロセスの左側の上流工程です。すなわち、MBSE プロセスが進捗すると、それによって V 字プロセスが進捗するというイメージをもっていただくと分かり易いでしょう。

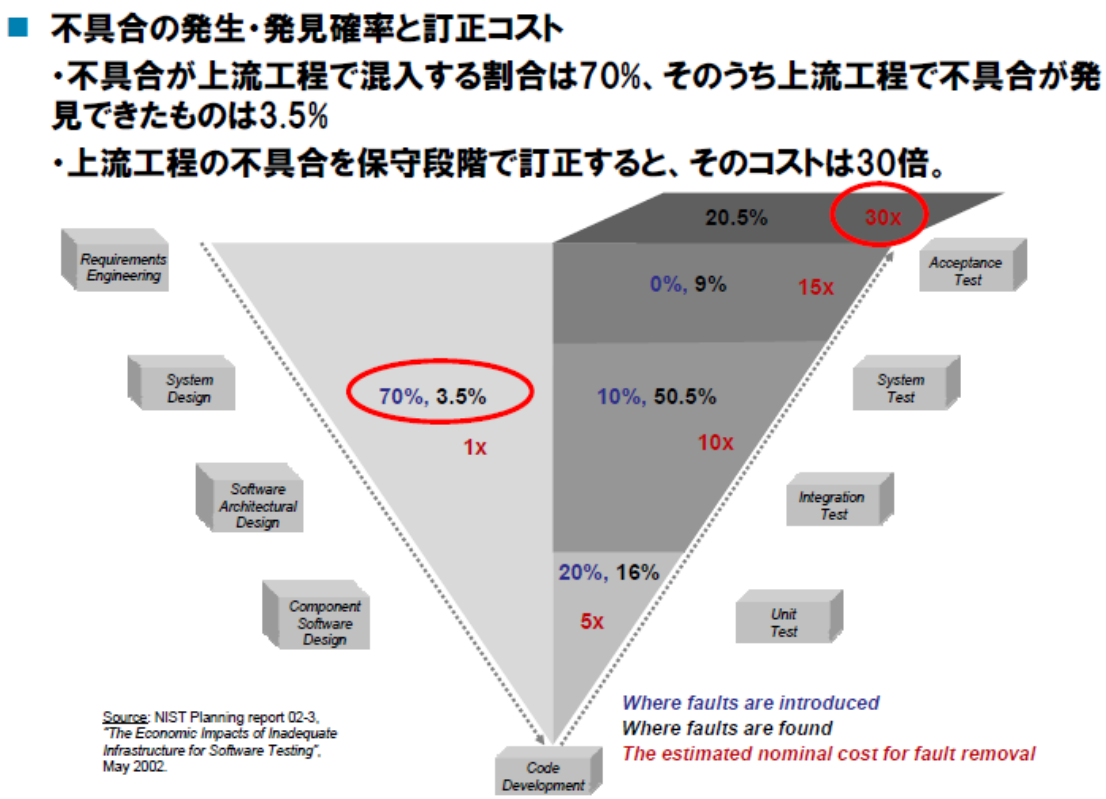

図 ２０　参考資料：不具合の発生・発見確率と訂正コスト

ここで大事なことは、ある作業フェーズ、ある作業レベルで何かの作業を行ったら、その検証（verification）や妥当性確認（validation）などの V&V と言われる作業も、そのフェーズ、そのレベル内で行うということです。これまでの主に自然言語を用いるシステムズエンジニアリングと異なり、成果物がモデルである MBSE では、実ハードウェアがなくても、検証作業をモデルに対して行います。そして、行った作業を「やりっ放しにしない」ことが重要です。そして、各作業の終了時には、行った作業の検証や妥当性確認を行うだけではなく、回帰試験を行い、これまでに行ってきた成果を壊しておらず、整合性が保たれていることの確認をすることも重要です。

究極的には、MBSE で品質を確保するためには、システムの本質を全て表すことができるモデル、つまりソフトウェアで構築されたバーチャルシステムを用いることで、上

流から一貫した品質の保証ができると考えられています。これを実現するため、実行可能モデルや、数学的シミュレーションなどについて、様々なツールベンダーによるツールの提供や、研究機関による研究が進められています。

しかし、研究が進まなくても、製品開発は進みます。そして、私たちのシステムの品質は、設計時からどんどん作り込まれていきます。ツールがあろうがなかろうが、システムの品質を確保するためには、そのシステムが複雑であればあるほど、実装段階に到達する前に、できるだけ早く、できるだけ多くの確認を行うことが重要であるという意識が最も重要です。「やりっ放しにしない」ために、現場では、要求分析レビューや、アーキテクチャ設計レビューをするという習慣があれば良いでしょう。

よく知られているように欠陥の修正コストは開発が進むにつれ、指数関数的に増加します（図 ２０）[15]。可能な限り、開発の早期に欠陥を除去し、開発の手戻りを防止する必要があります。しかし、現実的には、開発現場において、品質を保証するという責務は、V 字開発の最終工程であるいわゆるテスト工程、品質保証工程に任せきりにされていることが多く見られました。もし、製品の品質を効率的に保証したいのであれば、製品の品質保証を、これら V 字の下流工程に任せきりにするのではなく、開発上流の要求分析､アーキテクチャ設計レベルのハードウェアが手元にない時点の作業であっても、その作業成果物(MBSE であればモデル)の品質を保証できるようにするとともに、常にそれを行い続けなければならない、という開発現場での意識の変革が必要です。特に、ISO 61508、ISO 26262 などの安全基準に準拠するためには、この開発最上流からの一貫した品質のトレーサビリティの確保が重要な条件となっていることに注意が必要です。ですので、例えば、後述する図 ２４のパラメトリック図は手戻りを抑制するためのフレームワークにもなります。

3. 「導入事例や、導入のための TIPS」

MBSE は、総合的、多面的にシステム開発の効率を向上させるための技術的な方法論であり、プロセスでもあるため、逆に「最も効果のあるところはどこか？」という典型的な問いに答えることは簡単ではありません。実際、MBSE を適用した企業においても、その導入の目的は様々で、報告も様々な視点からされています。

2009 年にまとめられた MBSE 導入に関する OMG レポート[16]によれば、挑戦した人の多くが、MBSE は、適用したプロジェクトに大きな価値をもたらしたと考えているようです。（図　２１、スコアは 3.89（5 段階））。

そして、図　２２に示されるとおり、MBSE はソフトウェアエンジニアのみならず、ハードウェアエンジニアやテスターに対しても大きな一助となったという評価が得られています。

ここでは、MBSE 導入の効果を「製品開発プロセスへの効果」「多人数開発への効果」「複数製品開発への効果」という大きな 3 つの側面から見てみましょう。また、先行して導入を行った企業で見られる「導入のためのベストプラクティスやパターン」も考えてみることにしましょう。

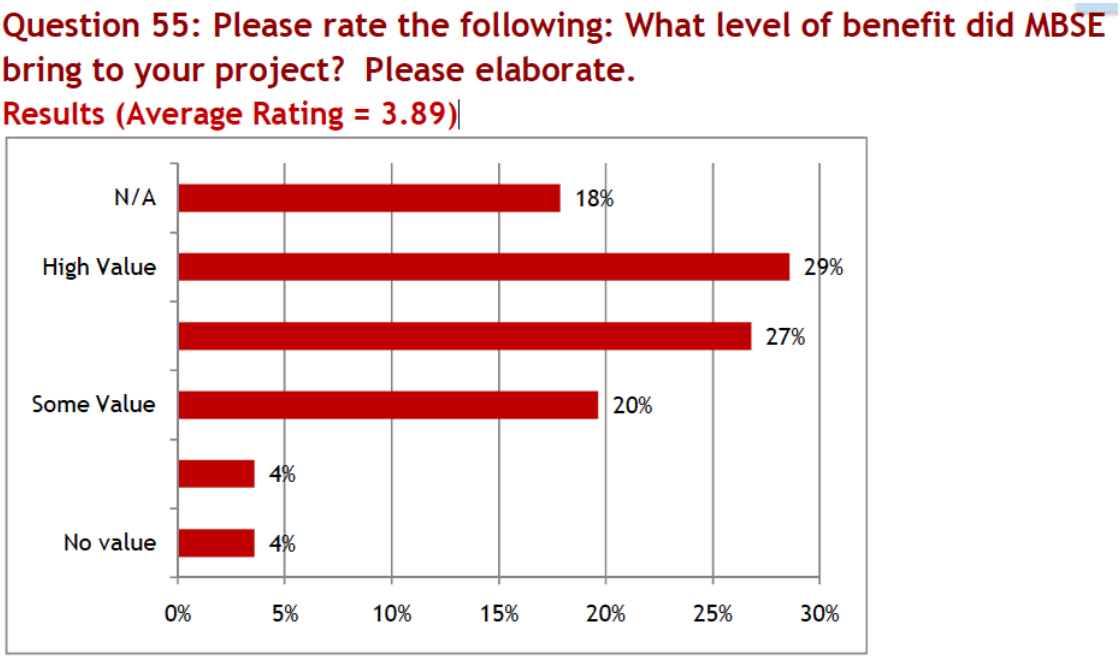

図　２１　MBSE の価値

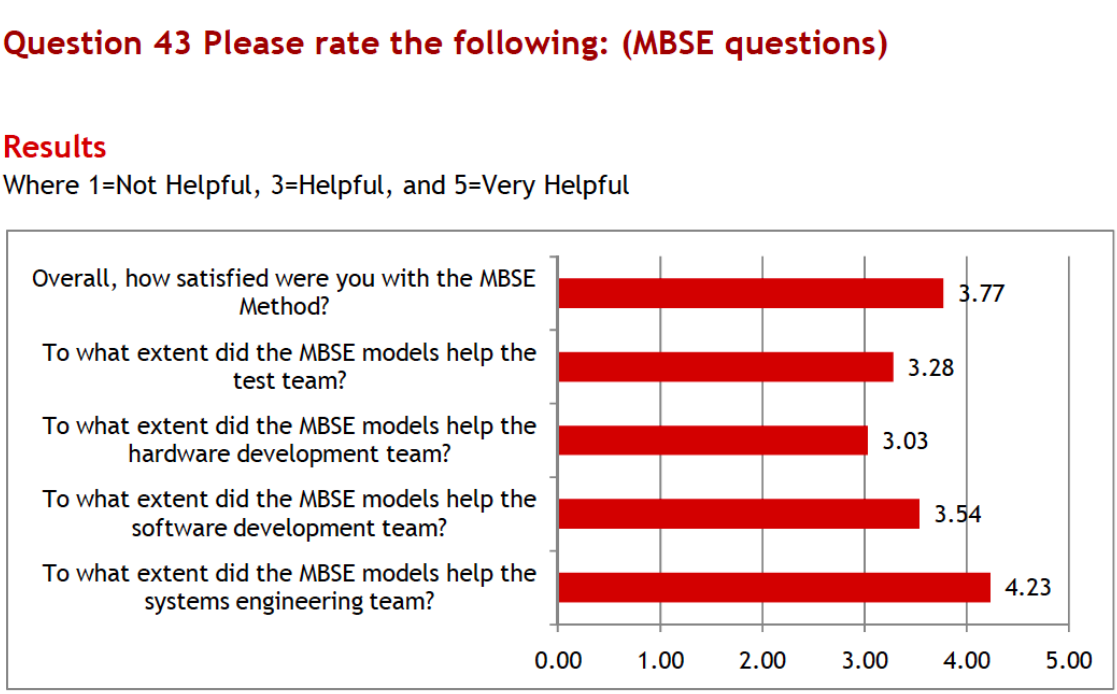

図　２２　MBSE 開発者別の評価

3.1. 製品開発プロセスへの効果

システムが高機能になり複雑になればなるほど、開発プロセスの早い時期からシステムに隠された欠陥を見つけることは困難になります。よく知られているように、ほとんどの欠陥は、開発初期の設計工程までの間に作り込まれてしまいますが、それらは、その時点では発見されず、テスト段階で障害として初めて検出されることになります。そして、もし、これらの障害が製品の製造段階に進むまで検出されず、出荷後に市場で初めて発見されたとすれば、大問題となります。そしてもし、その製品が大量生産されているとすれば、その対策には莫大なコストがかかり、大きな損失を生じる可能性もあるのです。

MBSE 導入の効果として最もよく知られているものは、欠陥を早期発見することによって、製品化の時間を短縮し、開発費を抑え、出荷にかかる時間を短縮するというものです。

例えば米国のある防衛システムの開発会社では、システムズエンジニアリングを導入することで製品化の時間を 40 %短縮しました[17]。また、アプリケーションの信頼性と安全が改善されました。様々な企業からハードウェアやソフトウェアパーツの供給を受けているこの企業の場合、キーとなったのは、サブシステム間のインタフェースを明確

にすることでした。ここでは、開発初期に実行可能なプロトタイプを作成し、モデル実行を通じ、インタフェースの欠陥を早期に発見しています。

General Motors 社（以下 GM と略す）初のシリーズ方式プラグインハイブリッド車であるシボレー・ボルトには約 1000 万行のソフトウェアコードが搭載されています。同社は、これまでよりもシステム開発が遥かに複雑になることを予測し、開発の開始時に、モデル駆動型システム開発のための準備をしました。GM の開発者は、モデルを使用して、組込まれているシステム間のやり取りを視覚化し、シミュレーションによってモデルのテストを行いました。GM は、通常 5 年かかる新車開発をわずか 29 ヶ月で完了したとのことです[18]。

セーフティクリティカルなシステムを開発する企業では、その開発プロセスが国の安全基準及び国際的な安全基準を満たしていることを示すことが要求されます。そして特に地理的に分散して並行開発を行っている場合にコンプライアンスを示すのは困難です。世界中にセーフティクリティカルな鉄道信号・制御ソリューションを提供しているオーストラリアの鉄道設備会社では、グローバルな安全性及び信頼性の標準へのコンプライアンスを促進する統合開発環境を必要としました。そこで、要求と設計の整合性を保つ環境を準備しました。ここでも MBSE が重要な位置を占めることになります[19]。

3.2. 多人数開発への効果

適切な製品を適切に作るためには、効果的な要求管理とそれに基づいた基本設計が必要です。利害関係者のニーズをしっかり理解し、そのニーズに基づいてシステム要求を注意深く規定しなければなりません。しかし、そのために何百人もの開発者が共通して使うことができるような一冊にまとまった要求文書や設計書を作ることは、開発者の苦痛を増やす活動になるだけです。

大規模な開発組織では、地理的に分散した複数の開発チームで、要求を共有しながら並行開発を行います。この共有がうまくいかなければ、要求トレーサビリティを十分にとることができず、要求のテストも非効率になり、開発効率は減少します。MBSE はこうした場合にも効果的です。システム全体の要求をモデルとしてまとめた要求リポジトリを作成し、分散された各開発者はこの要求を共有しながら、これを基に必要とする各ドメインのシステムモデルを構築することができます。

例えば、先ほど挙げた GM の例[18]では、ボルトの開発前に開発者の作業内容を調べた

ところ、開発者は、開発に当たって使用すべき文書類のバージョンの確認に、時間の大半を費やしていることが分かりました。

EADS（European Aeronautic Defence and Space Company）の航空宇宙部門の子会社でありシステムズエンジニアリングに古くから取り組んできたアストリアム（Astrium）でも、情報を見つけることが困難、様々な開発活動の一貫性が保てていない、コミュニケーションが困難、という 3 つの問題に悩まされていました。そこで、MBSE を適用し、これまでの文書ベースによるシステムズエンジニアリングの効率について測定し、コスト評価を行いました。その結果、コスト効果という側面に加えて、設計情報をほぼ完全に記述可能で、必要な情報を容易に抽出可能という理由から、MBSE は明らかに文書ベースのシステムズエンジニアリングよりも効率的であり、投資効果が高い（投資指標 0.85 -> 1.15）ことが分かりました[20]。

日本でもコンシューマエレクトロニクスの分散・協調設計[10] [11]と言う観点からの MBSE への取組みの例があります。

コンシューマエレクトロニクスの開発期間は比較的短く、近年では、設計が国際的に分散化されています。こうした環境下での不具合の発見の遅れに起因する設計の手戻りは、コストや納品スケジュールへのインパクトが大きく、不具合の原因の早い段階での対応が必要となります。サブシステムやモジュールが完成してからインテグレーションし検証する段階で、調整可能な箇所の部分的な修正をしたのでは、コスト的に見合わない場合が多くあります。このため、あるモジュール設計に必要となるその外部環境となる境界条件を仮に設定した上で、事前に機能レベルでの検証を行っておくことが重要なポイントとなります[10]。

こうした問題に対して、文献[11]では、開発するシステムの定義を明確にした上で、熱や音の交換が行われるキャビティ（空間）を一つのモジュールとして扱うことの有効性を示しています。特に、キャビティを一つのモジュールとして扱うことで、機能アーキテクチャが単純化されることを SysML のブロック定義図（図 ２３）から導き出しています。

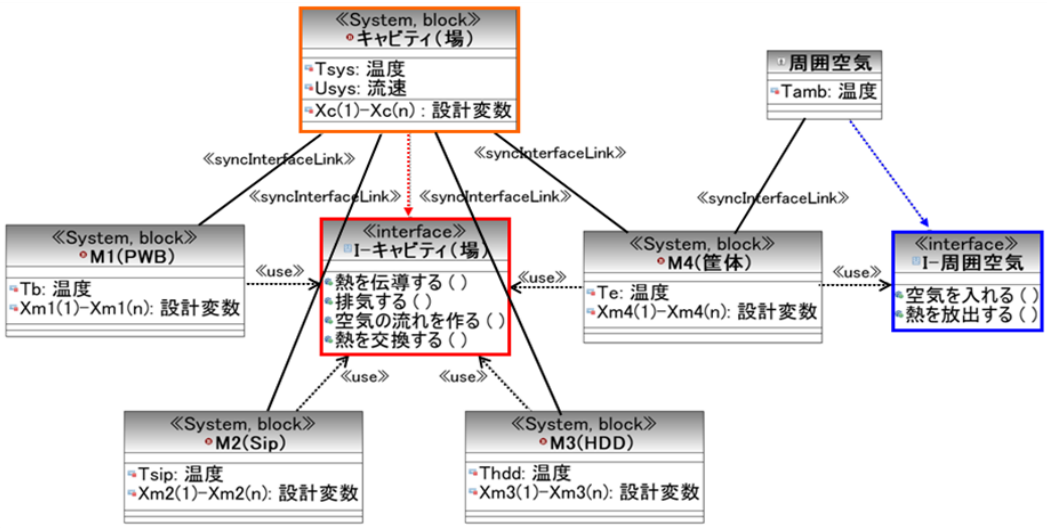

図　２３　キャビティをモジュールと見なしたブロック定義図

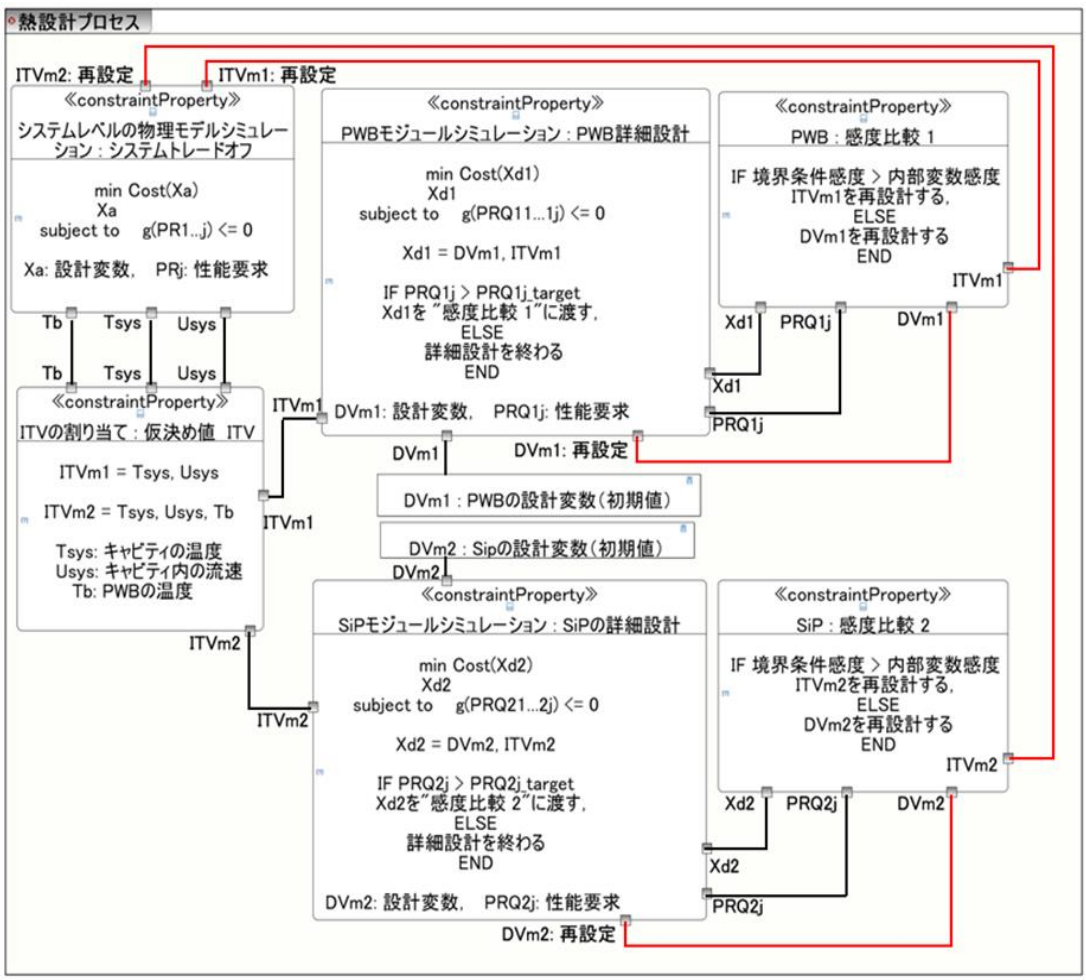

図　２４　熱設計プロセスを表すパラメトリック図

キャビティをモジュールとして扱うことにより他のモジュール同士のインタフェースはなくなります。この図は、国際的に分散する開発チームはキャビティでやり取りされる各モジュール間の物理パラメータを共有できることを表しています。また、この考え方に基づき、要求仕様に対するシステム境界条件または各モジュール内部の設計パラメータに関する最適化の問題を設定し、システムレベルとモジュールレベル間の相互依存性を考慮した最適化の手順を示しています。図 ２４に示すパラメトリック図には上述の一連の設計プロセスが記述されています。このようにキャビティをモジュールとして扱うことで、キャビティに関係する熱量や温度などの設計パラメータが分散された開発チームで共有され、設計の手戻りを減らすことに繋がります。

3.3. 複数製品開発への効果

現代社会においては、一つの製品だけを作り続けることでビジネスが存続するモノづくり企業はない、と言い切っても良いでしょう。通常、企業は、程度の差こそあれ、既存のハードウェアやソフトウェアを再利用しながら次の開発を進めます。MBSE を用いることにより、再利用の単位を明確にするとともに、いわゆる「車輪の再発明」を防止し、無駄な再利用を減少させることで、効率よく製品系列の製品群を開発することが可能になります。すなわち、MBSE の効果は、一製品の開発に対するものだけでなく、長期にわたる複数製品の開発という、企業の開発戦略に寄与します。

制御、計測に特化したテクノロジープロバイダである iPLON GmbH は、太陽光発電システムのための遠隔制御、管理システムを開発しています。分散した発電装置を制御・管理するソフトウェアは、これまで、全てハンドコーディングに頼っており、工数や、品質の面で大きな問題がありました。そこで、サブシステムとソフトウェアの再利用を促進するため、SysML を用いた MBSE と、UML を用いたソフトウェア開発を組み合わせ、要求分析からテストまで一貫してモデルを用いる開発プロセスを作り上げたのです。その結果、開発の初期に変更した要求に対応するテストモデルを構築、妥当性確認を行った上で、対応するソフトウェアを自動生成することが可能になりました。開発早期にエラーを発見、修正が可能になったことで、新プロセス適用後は、およそ 25%の障害が減少しました。それに伴い、新規プロダクトラインの開発時間が 5 ヶ月から 2 ヶ月と 60% 減少し、製品ごとにおよそ 10,000 ユーロの節約が達成されました[21]。

MBSE をビジネスモデルの分析や設計に適用する企業もあります。システムの欠陥は、

物理的なシステムだけのものではありません。ビジネスモデルそのものに欠陥や不具合があることもあります。そこで、システムズエンジニアリングを用いて最適なビジネスインフラを計画、構築することで、ビジネスを急速に立ち上げ、新しい市場に参入することも可能となります。ある企業では、車両データを利用するテレマティックス市場に参入する際に、システムズエンジニアリングを用い、取引関係やサプライチェーンモデルというビジネスの要素と、設計上の再利用単位であるオープンテクノロジーを組み合わせたビジネスインフラを作り上げました。これにより、その企業は、市場の状況に応じ、30 日以内で提供サービスを変更できるようになりました[22]。

2013 年 1 月末の INCOSE[2] International Workshop では、米 FORD 社の MBSE に関する大規模な取組み事例が発表されました[23]。FORD 社では、自動車の急速なエレクトロニクス化と、それを統合するソフトウェアの爆発に対応するため、まず最初に明確な製品群の全体の要求の定義から取り組みました。そして、要求の構成の可変性に対応可能な、技術要素のベースラインを構築したのです。すなわち、これまでハードウェア要素を管理していた PLM(Product Lifecycle Management)と、ソフトウェア要素管理の ALM(Application Lifecycle Management)というツールチェーンそれぞれを MBSE によりまとめあげられた要求とアーキテクチャを接続したのです。その結果、要求分析、コンポーネント設計、コード生成、そしてテストまでを一貫してモデルベースで行うことができるようになりました。彼らはこれを「モデルベースの再利用」と言っています。

3.4. 導入にあたってのベストプラクティスやパターン

近年、前述した FORD 社をはじめとして、様々な企業が MBSE に取り組んでおり、その結果が、INCOSE[2] をはじめとする様々な場所で発表されています。

これらの事例を調べてみると、MBSE 導入を成功させるためには、幾つかのベストプラクティスがあるようです。

まず、最も重要なことが目標を見失わないことです。MBSE の導入そのものは、新しい技法を用いた新しいシステム製品開発プロセスの導入にしか過ぎません。その目的は、あくまで「システムの開発を成功に導くこと」です。したがって、導入が成功したか失敗したかというのはシステム製品開発プロジェクトが成功したかどうかでしか評価できないことを常に意識しましょう。また、MBSE は開発の上流プロセスから適用されます。

[2] INCOSE: The International Council on Systems Engineering.

このため、新たな活動が、これまで行ってきた、なじみのある活動が始まる前に発生することになり、すなわち、利害関係者はこれまでよりもずっとプロジェクトの「立ち上がりが遅い」ように感じることが多いようです。しかし、新しく導入した全体最適のための MBSE の効果は、これまで通常行われてきた活動が円滑になり、結果的に全体としては効率が上がるというところに現れるため、そこは我慢が必要です。

MBSE を導入することで、これまでの開発とは異なり、開発初期から、プロジェクトに対する様々な要求が明確になり、それを実現するための作業も明確になります。これまで開発スケジュールを遅延させていた手戻りが減少することで、これまでよりも開発の後半戦がスムーズに進捗します。ですから、MBSE を適用したことで、プロジェクトが遅れた（ような気がした）としても、混乱していた（ような気がした）としても、常にプロジェクト成功のためにこの作業は必要なのだ、という信念をもって取り組むことが重要です。

要員の配置と言う観点では、システムズエンジニアリング専任の組織、もしくは少なくとも専任者を作りましょう。特定領域の実装という抽象度の低い業務と、システムズエンジニアリングという業務を同じ人が担うと、結果の出易いように感じる局所最適にばかり目が向いてしまい、システム全体への全体最適に目が行き届かなくなりがちです。明確にシステムズエンジニアリングの実行という責務をもつ組織を作ることで、それぞれの企業に必要な MBSE のスタイルができあがります。また、それだけでなく、導入に責任をもつ上位マネージャと、成功体験のあるメンターを準備することで、導入の成功率は大幅に上がります。

また、現時点で、システムズエンジニアリングを導入している企業では、開発コストの 3～8%をこの工程に掛けています。しかし、本当にプロジェクト開発全体を最適化した場合、およそ 15～20%になるだろうと考えられているということ[24]も念頭に置いておいてください。

「目標を見失わない」ためには、技術的な解決策を想定する前に、常に問題を明らかにすることを心掛けましょう。 解くべき課題を明らかにした上でも、なお、目的への手段、実装技術にはできるだけ先入観をもたず、複数の選択肢を検討できるようにすることが肝要です。目的達成のための手段、実装技術の決定をできるだけ遅くすることで、その製品やサービスが世の中に出るころに既に時代遅れになるか、あるいは不必要になるようなことを避けることができます。そして、技術的な手段に時間をとられることな

く、目的達成のために最も適した解決策を選択できる可能性が広がります。特に、初めてある製品やサービスの開発に取り組むときには、具体的な技術の選択の時期を意識的に遅らせましょう。

そして、問題が扱いやすくなるよう、システムを分割することを常に心掛けましょう。分割統治はローマ時代からの人類の叡智です。これまでに説明した通り、システムをより小さなサブシステムへ分解し、各サブシステムを開発する単位としてのコンポーネントに分割することが MBSE の基本となります。MBSE によってうまく分割されたシステムは、それぞれの部分でテスト可能で、また、全体への再統合も容易になるという特徴をもちます。そのため、テストや統合について十分に考慮した上で部分間のインタフェースを定義し、それが開発の進捗とともに勝手に変化してしまわないよう管理する、ということも重要なポイントです。

要求と設計との間を結び付けましょう。トレーサビリティという言葉で知られるように、設計上の決定の正当性を、具体的なビジネスニーズや技術ニーズとリンクさせることによって示せるようにしておきます。モデルのリポジトリ管理を行うことで、これが容易にできるということも MBSE の大きな特徴の一つです。特に、信頼性や安全性についての要求の優先順位が高いシステム開発では、トレーサビリティが、実現のためのキーポイントになります。

そして、テストは早めに頻繁に行いましょう。テストと要求とをリンクし、トレーサビリティをとることによって、開発中のシステムが常に要求を満たすことをテストによって保証できるようにしましょう。利害関係者全員が開発初期の段階からシステムを見ることができるように、プロトタイプ、シミュレータ、エミュレータなどの方法を利用しながら MBSE を進めましょう。

特に、初めて MBSE を導入する際には、主要な利害関係者を巻きこむことを必ず意識してください。様々な段階で判断するときには、顧客やユーザ、オペレータ、企画部門、上位レベルのマネージャなど、様々な観点をもつ利害関係者から意見を聞くことで、全体が最適化されるとともに、プロジェクトとしても、突然の大きな要求の変更を避けることが可能になり、開発リスクが減少します。

MBSE によるモデル化によって、多角的、多面的なシステム設計の側面、すなわちシステムの要求、構造、振る舞い、パラメトリック制約など、それぞれを分離して、視覚化するとともに、各側面ごとに一貫性を保つことが可能になります。これによって、要

求、アーキテクチャ、設計、実装、入力データ、出力データ、開発計画など、システムを複数の観点から確認することがより容易になります。そして、システムを構築し始める前に、モデルを通じて、様々なアーキテクチャや設計について熟慮が容易になることによって､プロジェクトのリスクや開発コストを抑えることができるようになるのです。

このような抽象度の高いレベルでの確認を行うということは、実装作業が始まる前から製品保証をし続けるようなものです。ですから、手戻りの少ない製品開発を行いたいならば、これは必須の作業となるのです。

また、SysML などの業界標準のモデリング言語の使用によって、曖昧さを減らし、開発チームのメンバ間に存在する理解の壁を取り払うことができます。そしてモデルリポジトリによる開発情報の管理は、開発の状況や文書などの情報の一元化を推進し、プロジェクトの混乱を防ぎます。

このように、MBSE による共同作業の改善と文書の明確化・正確化が、開発効率の向上、開発期間の短縮、そして製品品質の向上をもたらすのです。

4. おわりに

MBSE は日本市場に於いても、導入している企業が増えつつあります。本手引きでもご紹介した MBSE に関する多くの有効性が、認識され始めたと考えられます。本手引きの公開により、更に導入が加速することを期待いたします。

本手引きは、平成 24 年度の上流品質技術部会・統合システムモデリング技術 WG（主査：電気通信大学　新誠一教授）の活動を通して議論され、その内容を反映する形で WG の委員であり、慶應義塾大学大学院システムデザイン・マネジメント研究科の西村秀和教授と株式会社コギトマキナ代表取締役の鈴木尚志氏に執筆していただきました。また、コラムは株式会社 U'eyes Design 代表取締役の鱗原晴彦氏に執筆していただきました。この場をお借りして、厚く御礼申し上げます。

文献

[1] Systems Engineering Handbook, A Guide for System Life Cycle Process and Activities, Ver.3.2, International Council on Systems Engineering, (2010)

[2] IEEE Std 1220™-2005, IEEE Standard for Application and Management of the Systems Engineering Process, IEEE Computer Society, (2005)

[3] Kevin Forsberg, Hal Mooz, Howard Cotterman, Visualizing Project Management, Third Edition, John Wiley & Sons, Inc., (2005)

[4] Sanford Friedenthal, Alan Moore, Rick Steiner, A Practical Guide to SysML, The Morgan Kauffman OMG Press, 2008

[5] 西村秀和：監訳、システムズモデリング言語 SysML、東京電機大学出版局、2012

[6] Laurent Balmelli, An Overview of the Systems Modeling Language for Products and Systems Development, The Journal of Object Technology, Vol. 6, No. 6, (2007), pp. 149-177.

[7] Dennis M. Buede, The Engineering Design of Systems, - Models and Methods -, 2nd Edition, John Wiley & Sons, Inc., 2009

[8] 大富浩一、設計工学の目指すところ：設計からデザインへ、日本機械学会論文集 C 編、75 巻、７５１号、２００９、p. 516－523

[9] 石井浩介、飯野謙次、設計の科学　価値づくり設計、養賢堂、2008

[10] Kenichi Seki, Hidekazu Nishimura, Kosuke Ishii, Laurent Balmelli, Thermal/Acoustic Trade-off Design for Consumer Electronics in A Distributed Design Environment, Proceedings of The 19th Annual International Symposium of INCOSE, (2009), 0723

[11] 関 研一, 西村 秀和, 朱 紹鵬, Laurent Balmelli, 民生機器開発における機能・構造モデルを用いた分散協調設計 (SysML 製品モデルと DSM を利用したモジュール設計プロセスの計画), 日本機械学会論文集Ｃ編, Vol. 78, No. 785, pp.187-200 (2012)

[12] Model-Dased Systems Engineering with Rational Rhapsody and Rational Harmony for Systems Engineering -- Deskbook 3.1.2
http://www-01.ibm.com/support/docview.wss?uid=swg27023356

[13] http://model-based-systems-engineering.com/tag/sysmod/

[14] SysML/UML によるシステムエンジニアリング入門―モデリング・分析・設計, テ

ィム ワイルキエンス (著), 今関 剛 (翻訳), 貝瀬 康利 (翻訳)

[15] NIST Planning report 02-3, “The Economic Impacts of Inadequate Infrastructure for Software Testing”, May 2002

[16] Cloutier,Bone,"Compilation of SysML RFI-Final Report", OMG Document: syseng/2009-06-04
http://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:omg_rfi_final_report_02_20_2010-1.pdf

[17] Brockwell Technologies, Inc,
http://www-01.ibm.com/software/success/ cssdb.nsf/CS/CCLE-8DMT4B

[18] “シボレー・ボルト スマートな地球の実現へむかって”
http://www-01.ibm.com/software/jp/rational/announce/volt/

[19] “Invensys Rail Dimetronic speeds innovation and solution deployment”
http://www-01.ibm.com/software/success/cssdb.nsf/CS/STRY-8M7J3X

[20] Maurandy, Julien, et al. "Cost-Benefit Analysis of SysML Modelling for the Atomic Clock Ensemble in Space (ACES) Simulator." (2012). 22nd Annual INCOSE International Symposium - Rome, Italy
http://therightrequirement.com/pubs/2012/Getting%20the%20right%20requirement%20right%204.pdf

[21] “Increasing the efficiency of embedded software development”,
http://www-01.ibm.com/software/success/cssdb.nsf/CS/CCLE-84XRJ8

[22] Hughes Telematics helps steer the future of the auto industry's "next big thing"
http://www-01.ibm.com/software/success/cssdb.nsf/CS/JSTS-7XMS68

[23] Christopher Davey “Automotive Software Systems Complexity:
Challenges and Opportunities “ INCOSE International Workshop MBSE Workshop, 2013
http://www.omgwiki.org/MBSE/doku.php?id=mbse:incose_mbse_iw_2013

[24] “Understanding the Value of Systems Engineering”, Proceedings of the INCOSE International Symposium, 2004
http://www.incose.org/secoe/0103/ValueSE-INCOSE04.pdf

編集・著作

独立行政法人 情報処理推進機構

技術本部 ソフトウェア高信頼化センター