Vol.2017-SE-195 No.5
2017/3/12

変更要求に対する論理的な影響範囲分析に基づくソフトウェア動作検証方式の提案

磯田　誠[†1]　伊藤益夫[†1]

概要：我々は，製品ライフサイクルにわたる製品品質確保，省リソース，スケジュール遵守のバランス最適化を目的に，システム開発における重要度がますます高まっているソフトウェア生産技術に取り組んでいる．この取り組みの長期的な技術目標を「2025 年に計算機上で仮想設計・仮想検証・仮想生産するバーチャル・エンジニアリングの実用化」としている．

従来は CAD を中心に機械・電気・熱・流体といった分野の設計・検証技術が整ってきている．しかし，これらの分野と電子システムはまだ十分に融合しておらず，電子システムがハードウェア主体からソフトウェア主体へ，さらにはネットワーク中心へと変化が進むことによる複雑さの急増に対処できていない．我々はこのような問題に対処するための施策立案，技術開発，および開発現場への導入を進めている．

本稿では，我々のこれまでの仮想設計の施策の結果を受けて，仮想検証の施策の一つとして立案したソフトウェア動作検証方式について報告する．本報告では，「ソフトウェア主体に変化した製品の開発作業のほとんどが変更要求の実現」であることに注目し，変更管理プロセスで求められる影響範囲分析を高度化するための要素技術である機能テスト生成と等価性検査の方式検討結果を報告する．

バーチャル・エンジニアリングを実用化した際に得られる想定効果として，仮想設計では開発成果物の再利用量拡大による S/W 品質確保，仮想検証では作業量の大幅圧縮による省人員リソース，仮想生産ではリアルタイムな顧客からのフィードバックによる重大欠陥の撲滅およびスケジュール遵守を見込んでいる．

キーワード：変更管理，等価性検査，機能テスト，要求ベーステスト，構造カバレッジ解析，有界モデル検査，モデルベース開発

An Approach of Software Verification Method using Logical Impact Analysis recommended for Change Management

MAKOTO ISODA[†1]　MASUO ITOH[†1]

Keywords: Change Management, Equivalence Checking, Functional Testing, Requirements Based Testing, Structural Coverage Analysis, Bounded Model Checking, Model Based Development

1. はじめに

我々は，製品ライフサイクルにわたる製品品質確保，省リソース，スケジュール遵守のバランス最適化を目的に，システム開発における重要度がますます高まっているソフトウェア生産技術に取り組んでいる．この取り組みの長期的な技術目標を「2025 年に計算機上で仮想設計・仮想検証・仮想生産するバーチャル・エンジニアリングの実用化」としている．

従来は CAD (Computer Aided Design) を中心に，DMU (Digital Mock Up), CAE (Computer Aided Engineering), CAM (Computer Aided Manufacturing) を活用した機械・電気・熱・流体といった分野の設計・検証技術が整ってきている．中でも CAD は標準図面形式 ISO 10303 STEP (Standard for the Exchange of Product Model Data) が策定されており，今後 DMU, CAE, CAM とのデータ交換が活発になると考える．しかし，これらの分野と電子システムはまだ十分に融合しておらず，電子システムがハードウェア主体からソフトウェア主体へ，さらにはネットワーク中心へと変化が進むことによる複雑さの急増に対処できていない．

このようなソフトウェア主体への変化による複雑さの急増に対処するためのキーワードを“Modeling, Simulation and Visualization”と呼び，これを現実のものにするための施策立案，技術開発，および開発現場への導入を進めている．

【仮想設計】開発対象の抽象度に合った”Modeling”を活用して，設計情報や実装情報を図面形式で表記してデータ交換を可能にする．

【仮想検証】設計情報や必要なら詳細な実装情報も用いた”Simulation”により，製品の動作を十分に確認する．

【仮想生産】製品の生産コストを支払う前に，ハードウェア・ソフトウェア・ネットワークを顧客に体験させる．

本稿では，我々のこれまでの仮想設計の施策の結果を受けて，仮想検証の施策の一つとして立案したソフトウェア動作検証方式について報告する．本方式の核は，「ソフトウェア主体に変化した製品の開発作業のほとんどが変更要求の実現」であることに注目し，変更管理プロセスで求められる影響範囲分析（Impact Analysis）を高度化することである．これにより，S/W 品質確保と省人員リソースの両立が

†1 三菱電機(株) 情報技術総合研究所
Mitsubishi Electric Corporation, Information Technology R & D Center

可能になると見込んでいる．

本稿の構成は以下のとおりである．2 章では長期的な技術目標「バーチャル・エンジニアリング」を説明する．3 章では今回立案する仮想検証の一施策「S/W 同等性検証」の技術的な進歩性を述べる．4 章では施策全体を一言で表した”Modeling, Simulation and Visualization”の要約を示す．最後に 5 章では施策を実用化した際の想定効果を示す．

2. 2025 年にバーチャル・エンジニアリング

INCOSE (International Council on Systems Engineering) は「Systems Engineering Vision 2025」の中で，2025 年におけるシステムズエンジニアリングのあるべき姿を提唱している[1]．このあるべき姿と我々の主な事業領域である制御システムや制御装置の開発プロセス・手法の現状を照らし合わせて，長期的な技術目標を「2025 年に計算機上で仮想設計・仮想検証・仮想生産するバーチャル・エンジニアリングの実用化」としている．

この技術目標に至った背景を，自動車分野を例に説明する．従来の自動車分野では，図 1 に示すように，CAD を中心に DMU, CAE, CAM といった設計・検証技術の活用が進んでいる．DMU は形状やデザイン，CAD は機械／電気システム，CAE は熱・流体・振動解析など，CAM は製造オペレーションのための作業支援環境である．自動車の各種制御の電子化が進んだことで制御装置の搭載数が急増してきているが，制御装置で構成される電子システムは依然としてシステムテストの段階で他の部位と本格的に結合することが多く，手戻り作業発生によるロスコストが大きくなるという問題点がある．

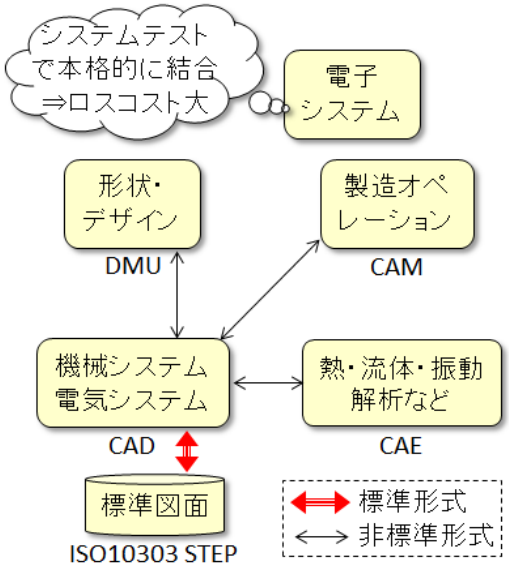

図 1　従来の設計・検証技術と問題点

Figure 1　The current state of engineering methods and issues.

電子システムの特徴がハードウェア主体からソフトウェア主体へ，さらにはネットワーク中心へと変化が進むことで，相互接続による複雑さが急増している[1]ことに注目し，図 1 に示した従来の設計・検証技術に電子システムを組み入れたあるべき姿を図 2 に示す．これにより，S/W 品質確保と省人員リソースの両立が可能になると見込んでいる．

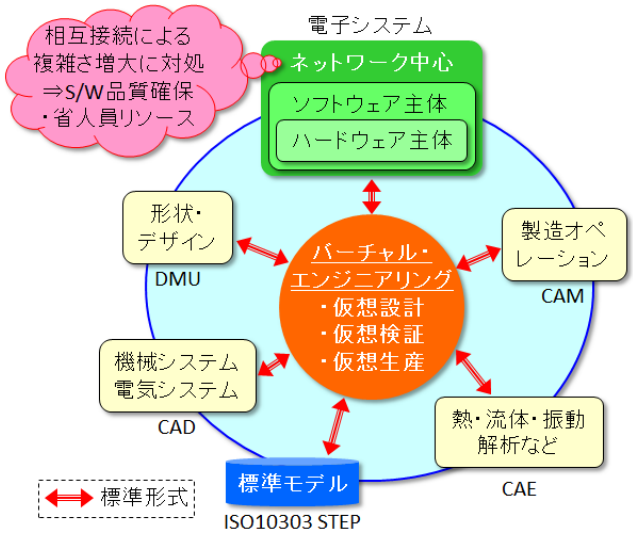

図 2　設計・検証技術のあるべき姿

Figure 2　The future state of engineering methods and goals.

以降では，先に 3 章で今回立案する仮想検証の一施策「S/W 同等性検証」の技術的な進歩性を述べ，その後 4 章でキーワード”Modeling, Simulation and Visualization”の中身となる仮想設計, 仮想検証, 仮想生産の各施策を紹介する．

3. 仮想検証の一施策「S/W 同等性検証」

3.1 想定するシチュエーション

バーチャル・エンジニアリングを実用化するための施策を要約すると，仮想設計は「開発早期に準形式言語を用いて仕様を正確に記述する」，仮想検証は「開発終盤のテスト前に十分にレビューや論理検証をする」，仮想生産は「実物はなくとも顧客に製品を体験させる」となる．

また,「ソフトウェア主体に変化した製品の開発作業のほとんどが変更要求の実現」であることに注目すると，バーチャル・エンジニアリングというあるべき姿は，変更前後の二つの製品の間に適用することになる．

本稿では変更要求の中でも最も基本的な機能変更に焦点を合わせ，図 3 に示すようなシチュエーションを想定する．

(A)　一つの製品を日々開発していくときに，時間とともに機能的に成長していく二つのバージョンの間．

(B)　同じ製品系列に属する複数の製品の中で，機能的に異なる二つのバリエーションの間．

シチュエーション(A)では，機能変更前の母体 S/W に加える機能変更の種別に対応して，機能変更後 S/W を表 1 に示すように論理的に分割できる．ここで論理的に分割できるとは，S/W をメソッドや関数という単位で見ると種別が異なる機能変更を実現するための S/W の修正が混在してしまって単純には分割できないが，S/W を処理の実行経

路を表すパスという単位で見ると機能変更と一対一に対応付けることができる，という意味である．

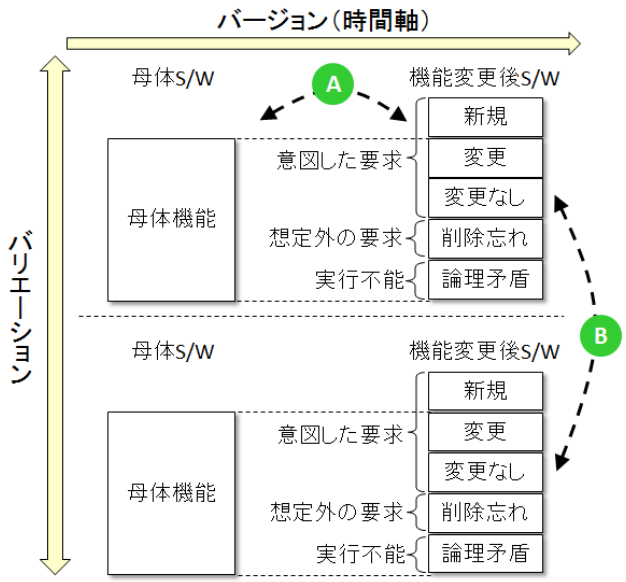

図 3　機能変更に焦点を合わせた想定シチューション
Figure 3　The assumed situation focused on functional changes.

表 1　機能変更種別に対応した S/W の論理的分割
Table 1　The mapping between functional changes and logical decomposed parts of software.

機能変更分類	機能変更種別	S/W の論理的分割
意図した要求事項の実現	新規	S/W の**新規部分を構成するパス**の集合
	変更	S/W の**修正部分または未修正部分を構成するパス**の集合
	変更なし	S/W の**未修正部分を構成するパス**の集合
想定外の要求事項の混入	削除忘れ	削除した機能を実現していた**残存 S/W を構成するパス**の集合
どんな条件でも実行不能	論理矛盾	機能とは無関係に，**構造上存在しないに等しいパス**の集合

以降では，表 1 に示した S/W の論理的分割を実現する施策である機能テスト生成と等価性検査，およびこれらを統合した施策である S/W 同等性検証の技術的な進歩性を述べる．

3.2 十分かつ効率的に動作確認する機能テスト生成

表 1 で S/W の論理的分割をパスの集合で定義したが，実物の電子システムに搭載される S/W を構成するパスの数は，一般に天文学的な数字になり容易に求めることはできない．そこで本施策では，各種の機能テスト手法の基準に適う動作確認のためのテストケース（以降，機能テストケースと呼ぶ）の一式を，S/W に与えたときに実行するパスの集合を対象にする．このように定義したパスの集合は，S/W を構成するパスの真の集合の部分集合にすぎないが，機能テスト手法の基準に適うという意味で，S/W の機能的な品質確保の点で十分なパスを選択したと判断する．

機能テスト手法に基づいて S/W の論理的分割を再定義したものを表 2 に示す．

表 2　機能テスト手法に基づく S/W の論理的分割
Table 2　Logical decomposed parts of software based on functional testing methods.

機能変更分類	機能変更種別	S/W の論理的分割
意図した要求事項の実現	新規	S/W に対する**機能テストで実行するパス**の集合
	変更	
	変更なし	
想定外の要求事項の混入	削除忘れ	削除した機能を実現していた**残存 S/W を実行するパス**の集合
どんな条件でも実行不能	論理矛盾	機能とは無関係に，**決して実行できないパス**の集合

上記の機能テストケース一式は S/W の動作確認のためには十分と言えるが，これを人手で作成するのはかなりの作業工数を要することが開発現場で問題になっている．そこで著者らは文献[2]において，このような機能テストケースを自動生成するアルゴリズムを考案した．本稿ではどのような機能テストケースを作成するべきかという機能テストに対する要求事項，および従来手法と本施策の差異を改めて示す．実現方法の詳細は文献[2]を参照されたい．

機能テストに対する要求事項にはさまざまなものがあるので，航空分野や自動車分野のように，人命の危険に直結するため高い安全性が求められる製品を想定する．具体的には機能安全規格（航空分野の DO-178B[3]や自動車分野の ISO26262[4]）とその適用の考え方[5]などを総合して，以下のような要求事項を採用する．

- 要求ベーステストの考え方に従って，外部的な機能仕様の実現を確認すること．これを機能網羅と呼ぶ．
- 機能テストケースだけを用いて構造カバレッジ解析することで，構造カバレッジの上限値を確定すること．これを構造網羅と呼ぶ．

従来手法では，構造カバレッジが 100%にならない場合は任意の入力値を用いてとにかく 100%にすることを目指すことが多い．しかし，任意の入力値でテストしても「ある入力値ではクラッシュしない」という意味だけしか持たず，機能テストに対する要求事項を満たすことはできない．

これに対して本施策では，図 4 に示すように機能テストケースでは決して実行できない S/W の部分があるかどうか，形式手法を用いて解析して確実な根拠を提示する．この解析結果を用いて，表 2 に示した 機能変更種別「新規／変更／変更なし」，「削除忘れ」および「論理矛盾」の 3 つに論理的分割することで，「新規／変更／変更なし」の実行結果が構造カバレッジの上限値，「削除忘れ」と「論理矛盾」はともに実行不能であると確定できる．

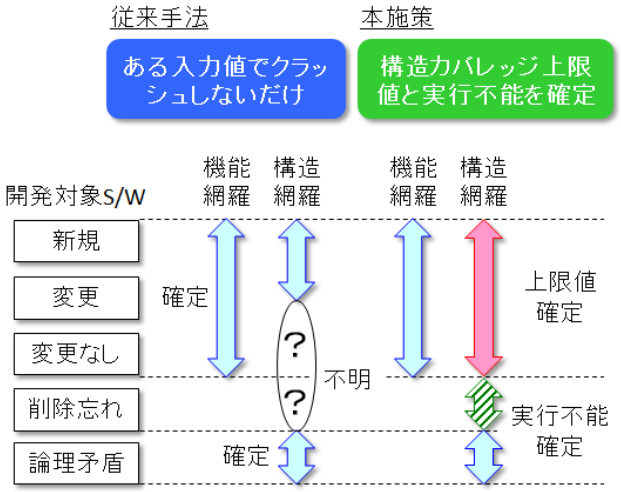

図 4　構造カバレッジの上限値の確定方法

Figure 4　Structural coverage analysis using formal method.

機能テスト生成の適用の一例として，機能仕様のサンプルとする状態遷移，これを実装した S/W，自動生成した機能テストケース一式をそれぞれ図 5，図 6，表 3 に示す．

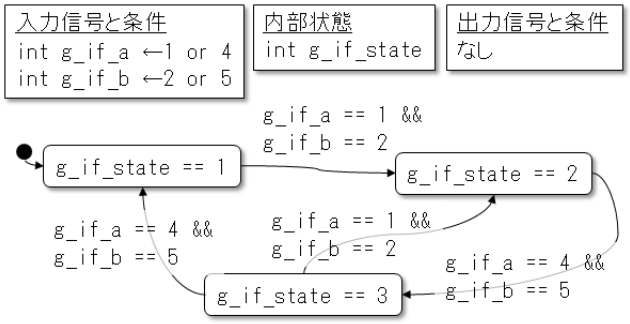

図 5　機能仕様のサンプルとする状態遷移

Figure 5　The state chart as functional specification sample.

```
short g_if_a, g_if_b, g_if_state;
void statechart_if_init() {
  g_if_a     = 0;
  g_if_b     = 0;
  g_if_state = 1;
}
void statechart_if() {
  if ( (g_if_a == 1) && (g_if_b == 2) ) {
    if ( g_if_state == 1 ) {
      g_if_state = 2;
    } else if ( g_if_state == 3 ) {
      g_if_state = 2;
    }
  } else if ( (g_if_a == 4) && (g_if_b == 5) ) {
    if ( g_if_state == 2 ) {
      g_if_state = 3;
    } else if ( g_if_state == 3 ) {
      g_if_state = 1;
    }
  }
}
```

図 6　状態遷移を実装した S/W

Figure 6　The software implementation of state chart.

表 3　自動生成した機能テストケース一式

Table 3　The functional test-cases generated automatically.

ステップ数	No.	入力 g_if_a	入力 g_if_b
1	1-1	4	5
	1-2	1	2
	1-3	0	2
	1-4	5	5
	1-5	1	1
	1-6	4	6
2	2-1	1	2
		1	2
3	3-1	1	2
		4	5
		1	2
	3-2	1	2
		4	5
		4	5

表 3 に示した機能テストケース一式は分岐網羅の基準を満たすように生成したものである．この機能テストケース一式を S/W に与えて実行し，構造カバレッジを計測したところ，確かに 100%になった．この結果を可視化したものを図 7 に示す．網掛け部分は実行した分岐箇所を表しており，すべて実行されていることが分かる．

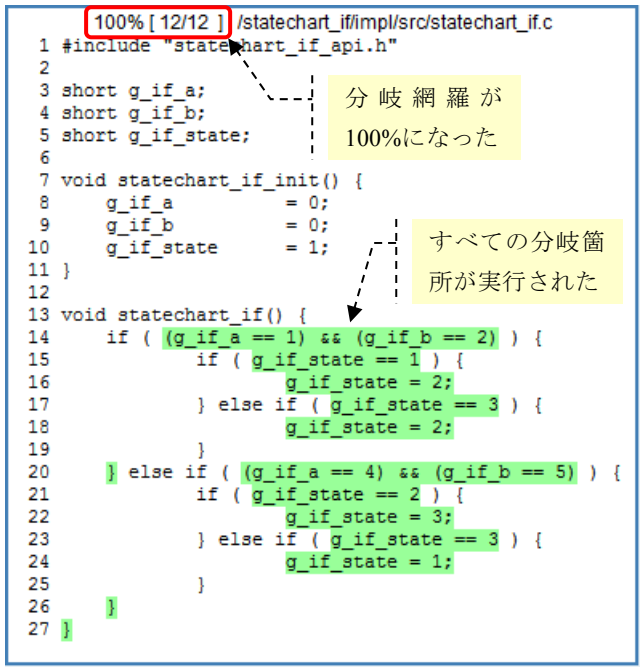

図 7　分岐網羅の構造カバレッジ計測結果とその可視化

Figure 7　The visualized structural coverage measured for branch parts

3.3 論理的な影響範囲を確実に特定する等価性検査

一般に変更管理プロセスでは，機能変更を加えたときの影響範囲を特定することが求められる[3][4]．従来手法では機能変更を実現するために S/W を修正した前後で，機能仕様書やプログラムコードのテキスト差分を分析している．しかし，関数名や変数名を修正しただけでも差分が出るた

め，演算内容を変更したかどうか正確には分からない．

そのため，機能テストケースも期待出力値も再利用できる回帰テストで済む範囲を狭く，機能テストケースと期待出力値の作成からやり直さなければならない範囲を広く取らざるを得ず，テスト工程の作業効率を悪化させている．

これに対して本施策では，3.2 の機能テスト生成を用いて機能変更後 S/W を論理的に分割した後に，機能変更種別「新規／変更／変更なし」をさらに機能テストの対象か回帰テストの対象か細分化することで，表 4 に示すように論理的な影響範囲を特定する．

表 4　S/W の論理的分割に対応した影響範囲

Table 4　The mapping between logical decomposed parts of software and change impacts.

機能変更種別	S/W の論理的分割	影響範囲
新規	S/W に対する**機能テストで実行するパス**の集合	**新たな機能テストの対象**
変更		**追加の機能テストの対象**となる差異部分，または**回帰テストの対象**となる同等部分
変更なし		**回帰テストの対象**となる同等部分
削除忘れ	削除した機能を実現していた**残存 S/W を実行するパス**の集合	機能仕様がないため**機能テストを実施できない**
論理矛盾	機能とは無関係に，**決して実行できないパス**の集合	そもそも**機能テストを実施できない**

ここで回帰テストとは機能テストの一種であり，機能変更前後で S/W の未修正部分に同一の機能テストケースを与えたときに，出力値が一致することをもって新たな欠陥が発生していないと判断する手法である．

表 4 に示した影響範囲を特定する方法として，等価性検査（Equivalence Checking）という技術を適用する．等価性検査とは元々，ハードウェア記述言語（Verilog-HDL や VHDL）を用いた論理回路の動作レベル記述を RTL 記述やゲート・レベル記述に変換した際に，各記述内容が論理的に同じかどうか確認する LEC（Logic Equivalence Checking）という技術である[6]. この技術を S/W の設計情報や実装情報に適用し，S/W の関数単位で演算内容が論理的に同じかどうか判定する方法が研究されている[7]．以降，S/W のある部分の演算内容が論理的に同じことを「等価」，論理的に異なることを「不等価」と呼ぶ．

本施策では，文献[7]にある S/W の関数単位で等価／不等価を判定する方法を応用して，機能テストや回帰テストで実行するパスを等価パスと不等価パスに分類する．そして，機能変更前後で影響範囲に入る S/W の同等部分または差異部分を，それぞれ等価パスの集合または不等価パスの集合で定義する．具体的な手順を以下の(1)～(3)および図 8 に示す．

(1)　機能変更前の母体 S/W の関数コールグラフ上で，エントリ関数に入力を与えてから最終的な出力が得られるまでに存在するパスの集合を，3.2 の機能テスト生成を用いて求める．こうして求めた母体 S/W を構成するパスを「母体パス集合」と呼ぶ．

(2)　母体 S/W と機能変更後 S/W の関数単位で等価性検査を実施し，演算内容が論理的に同じ「等価関数」または論理的に異なる「不等価関数」を判定する．

(3)　母体パス集合と等価／不等価関数の判定結果を照合して，等価関数のみを通るパスを「等価パス」，一回でも不等価関数を通るパスを「不等価パス」に分類する．

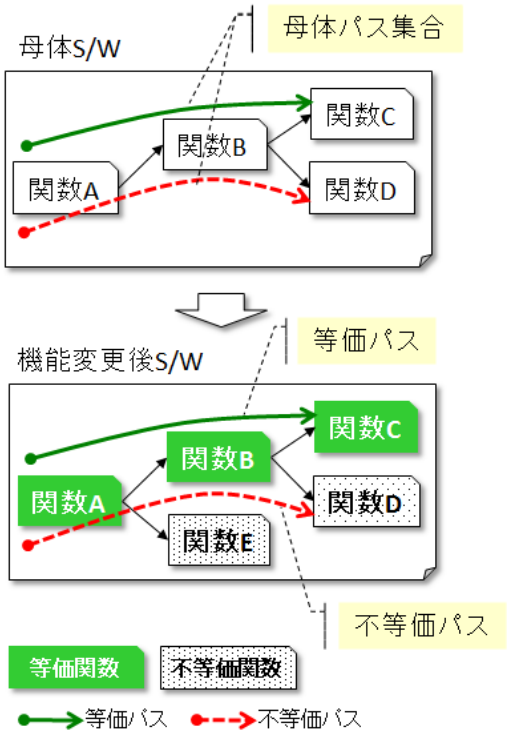

図 8　論理的な影響範囲の特定方法

Figure 8　Concept of logical impact analysis.

等価性検査の適用の一例として，3.2 で機能仕様のサンプルとした状態遷移に対して，以下のように S/W を修正することを考える．

- 機能仕様の状態遷移は変更しない．
- S/W の実装方法を，if 文から switch 文に修正する．
- 修正前後を区別するため，変数名と関数名も修正する．

上記の修正を施した二つの S/W のテキスト差分を図 9 に示す．図 9 の上半分が修正前，下半分が修正後，網掛け部分がテキスト差分を表している．機能仕様に変更がないにも関わらず多くの差分を検出しており，回帰テストで済むか追加の機能テストが必要か，判断することが難しいことが分かる．

これら二つの S/W に等価性検査を適用すると，論理的に演算内容がまったく同じすなわち「等価」という判定結果になり，回帰テストだけで済むと確実に判断できる．

図 9　修正前後の二つの S/W のテキスト差分

Figure 9　The text difference caused by software modification.

3.4　等価性検査と機能テスト生成による S/W 同等性検証

3.2 の機能テスト生成および 3.3 の等価性検査を以下のように統合したものが，今回立案する仮想検証の一施策「S/W 同等性検証（Software Equivalence Verification）」である．S/W 同等性検証の要求事項を以下のように定義する．

(1)　二つの S/W の同等部分と差異部分を特定すること．

(2)　同等部分の機能動作が一致することを保証すること．

(3)　差異部分の機能動作を十分に確認すること．

目的(1)を達成するための主要技術が等価性検査，目的(2)(3)を達成するための主要技術が機能テスト生成である．機能テスト生成と等価性検査を用いた，S/W 同等性検証の作業フローを以下に示す．

(i)　等価性検査において等価パスに分類した母体パス集合の部分集合を「同等部分」，不等価パスに分類した部分集合を「差異部分」と特定する．（図 10）
そして，表 1，表 2，および表 4 から機能変更種別に対応する論理的な影響範囲を特定し，実施するべき機能テスト手法を決定する．（表 5）

(ii)　同等部分を対象に回帰テストを実施する．等価パスに対する機能テストケースを母体 S/W と機能変更後 S/W の両方に与え，母体 S/W の期待出力値と機能変更後 S/W の実際の出力値が一致することを確認する．リファクタリングのように，機能仕様は変更せずに S/W を修正したことで機能網羅不足や構造網羅不足が新たに発生した場合は，追加で機能テストを実施する．（図 11）

(iii)　差異部分を対象に追加の機能テストを実施する．一つの S/W を対象に新たに機能テストを実施するのと同じ手順を踏んで，機能変更後 S/W の機能テストを実施すればよい．（図 12）

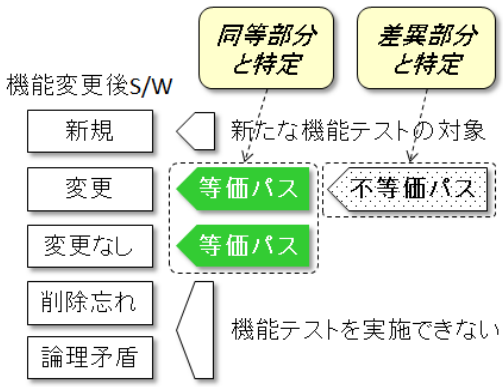

図 10　S/W 同等性検証の作業フロー（1/3）

Figure 10　The software equivalence verification flow. (1/3)

表 5　機能変更種別に対応した影響範囲と機能テスト手法

Table 5　The mapping between functional changes and change impacts, and specifying functional testing methods.

機能変更分類	機能変更種別	影響範囲
意図した要求事項の実現	新規	**新たな機能テストの対象**
	変更	**追加の機能テストの対象**となる差異部分と，**回帰テストの対象**となる同等部分
	変更なし	**回帰テストの対象**となる同等部分
想定外の要求事項の混入	削除忘れ	機能仕様がないため**機能テストを実施できない**
どんな条件でも実行不能	論理矛盾	そもそも**機能テストを実施できない**

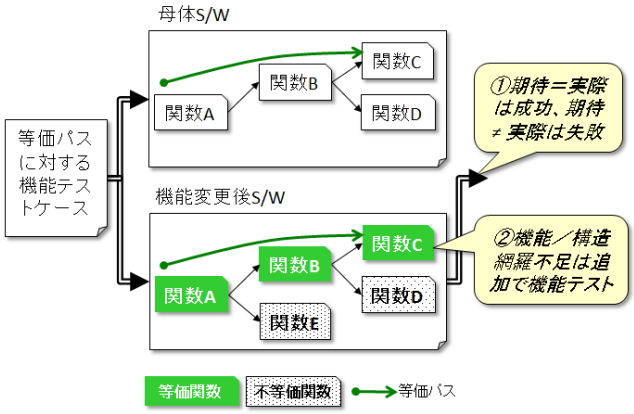

図 11　S/W 同等性検証の作業フロー（2/3）

Figure 11　The software equivalence verification flow. (2/3)

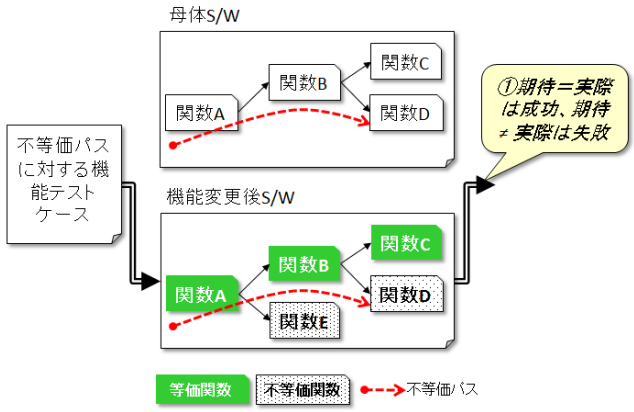

図 12　S/W 同等性検証の作業フロー（3/3）

Figure 12　The software equivalence verification flow. (3/3)

4.　“Modeling, Simulation and Visualization”

バーチャル・エンジニアリングを開発現場で使える道具箱（Toolbox）として提供することを表すキーワードを”Modeling, Simulation and Visualization”と呼ぶ．我々は，従来手法を最大限に活用した上で個々の道具（Tool）を取り揃える活動を，図 13 に示すような「開発対象の抽象度のレベル」と「開発プロセスの工程」を軸とする枠組みの下で継続している．以降では，これまで取り組んできた仮想設計の施策の結果，今回立案した内容を含む仮想検証の施策，今後取り組む仮想生産の施策の予定を紹介する．

4.1　これまでの仮想設計の施策の結果

仮想設計の核として，準形式言語（SysML[8]，UML[9], Simulink[a]といった汎用モデリング言語）を用いて開発成果物を正確に記述するモデルベース開発技術を開発現場に導入する活動を進めている．我々の主な開発対象であるシステムレベルおよびコンポーネントレベルにわたって，要件定義／設計工程で開発成果物をモデル図面で記述するための，記述内容および表記法のガイドラインを規定している．なお，顧客との契約締結への効果を見込んで，業務運用レベルの一部についても規定している．

a Simulink は The MathWorks, Inc.の登録商標である．

これまでの案件適用の経験から，記述内容のガイドラインでは，準形式言語の文法とは独立に開発成果物に記述するべき内容を規定している．表記法のガイドラインでは，特定の準形式言語の文法に従って記述内容を表現する方法を規定している．

また，システムレベルとソフトウェアコンポーネントレベルでは明らかに抽象度が異なり，採用する準形式言語も異なることが多いため，ガイドラインとしては別物になりがちである．これに対して我々は，これらのレベルの間で記述内容のガイドラインは意図的に連続性を持たせておき，採用する準形式言語が異なる場合に表記法のマッピングを規定するようにしている，

仮想設計の概念を上記のように整理することで，複数の事業領域にわたって要件定義／設計工程での相乗効果を得られると見込んでいる．

4.2　今回立案したものを含む仮想検証の施策

仮想検証は S/W 品質確保と省人員リソースの両立の要であり，テスト工程に入る前のレビュー／論理検証を十分に実施するとともに，実物の S/W の動作を確認するテストを十分に実施するべきである．しかし開発現場では，要件定義／設計工程の開発成果物を自然言語で記述することが大半のため，レビューはともかく論理検証は十分にできていない．また，上流工程での作業遅延のしわ寄せによるスケジュール逼迫のため，最後の砦のシステムテストでひたすら不具合を抑え込んでいるのが実情である．

これに対して我々は，まず仮想設計の施策により，準形式言語を用いて開発成果物を正確に記述するようにした．準形式言語で記述していることを前提に，レビュー／論理検証からシステムテストまでで実施するべきことをアルゴリズムとして定義し，既存ツールや内製ツールを用いて作業自動化するアプローチを取っている．

今回立案したものを含め，まず直近の課題であり高い効果を見込めるソフトウェアコンポーネントレベルの施策である等価性検査，S/W シミュレーション，機能テスト生成の技術開発の最中である．システムレベルの施策である表記自動チェック，並行動作解析，性能解析などについては，それぞれ基礎検討やフィージビリティスタディを進めている段階である．

4.3　今後の仮想生産の施策の予定

仮想設計および仮想検証で製品開発の主要な工程をカバーした後は，仮想生産の施策を予定している．これはまだコンセプト段階だが，製品の生産コストがかかる試作品や量産品の製造に入る前に，顧客に実物さながらの製品を体験してもらって有益なフィードバックを迅速に得ることを考えている．将来的には，製品を開発している企業が一度も実物の製品を見ることなく，製品が顧客に届けられるようになる可能性もある．

現在取り組まれている典型的な仮想生産は，CAD や CAE

などで作成したエンジニアリングデータから，物理的なプロトタイプを製造するというものである．3D プリンタを用いて単なるモックアップではない精度や強度を持った筐体を製造するのもこの一例である．

一方ソフトウェアには，上記のような意味での製造工程は存在しないので，まずソフトウェアの仮想生産とは何かを定義する必要がある．ソフトウェアは無形の情報の集合であり物理的に実体化することはできないので，コンピュータ上での可視化技術が核になると想定している．

しかし単なる GUI やダッシュボードのように作り手の観点が強いものではなく，あくまで使い手の観点で価値がある情報を可視化して実際に体験してもらった結果のフィードバックを受けることと，ソフトウェアを極めて柔軟に作り変えられる技術とが合わさった可視化技術になると考えている．

5. おわりに

バーチャル・エンジニアリングを実用化する際の主な狙いと得られる想定効果を以下に示す．

【仮想設計】複数の事業領域にわたって開発成果物のガイドラインを規定する労力を取ることで，開発成果物の再利用量を拡大することを狙う．これにより，S/W 品質確保のための基盤ができあがると見込んでいる．

【仮想検証】十分で確実な作業が求められながら人手に頼っている部分が多いところ，アルゴリズム化および作業自動化の労力を取ることで，作業量を大幅に圧縮することを狙う．これにより，S/W 品質確保と省人員リソースを両立できると見込んでいる．

【仮想生産】実物さながらの製品を別途作る労力を取ることで，リアルタイムに顧客のフィードバックを得ることを狙う．これにより，「欲しかったものはこれではない」というような重大な欠陥を撲滅してスケジュールを遵守できるようになると見込んでいる．

仮想設計の施策は，技術開発が収束して開発現場への導入を進める段階にある．

仮想検証の施策は，技術開発を進めている最中のものが多いが，本稿で報告した方式は収束に向かいつつある．収束に向かっているものは開発現場への導入に力点を移していくとともに，継続的に技術開発および改良を進めていく．

仮想生産の施策は，無形の情報の集合であるソフトウェアに求められる「生産」のコンセプト作りを始めた段階である．今後，具体的な施策を立案して技術開発するべき内容を詰めていく予定である．

参考文献

[1] *A World in Motion - Systems Engineering Vision 2025*, INCOSE, June 2014.
[2] 磯田 誠，西山博仁：制御ソフトウェア向け機能テストケース生成方式の提案，*情報処理学会研究報告, Vol. 2016-SE-191, No. 30*, 2016.
[3] *Software Considerations in Airborne Systems and Equipment Certification, RTCA DO-178B*, RTCA, Inc., December 1992.
[4] *Road vehicles - Functional safety - Part 1 - 10*, ISO 26262, ISO, 2011.
[5] Kelly J. Hayhurst, Dan S. Veerhusen, John J. Chilenski, Leanna K. Rierson, *A Practical Tutorial on Modified Condition/Decision Coverage*, NASA/TM-2001-210876, 2001.
[6] Anmol Mathur, Edmund Clarke, Masahiro Fujita, Pascal Urard, “Functional Equivalence Verification Tools in High-Level Synthesis Flows,” *IEEE Design & Test Computers*, July/August 2009.
[7] Rupak Majumdar, Indranil Saha, Koichi Ueda, Hakan Yazarel, “Compositional Equivalence Checking for Models and Code of Control Systems”, *52nd IEEE Conference on Decision and Control*, December 2013.
[8] *OMG Systems Modeling Language (OMG SysML™) Version 1.4*, OMG formal/2015-06-03, OMG, September 2015.
[9] *OMG Unified Modeling Language™ (OMG UML) Version 2.5*, OMG formal/2015-03-01, OMG, March 2015.

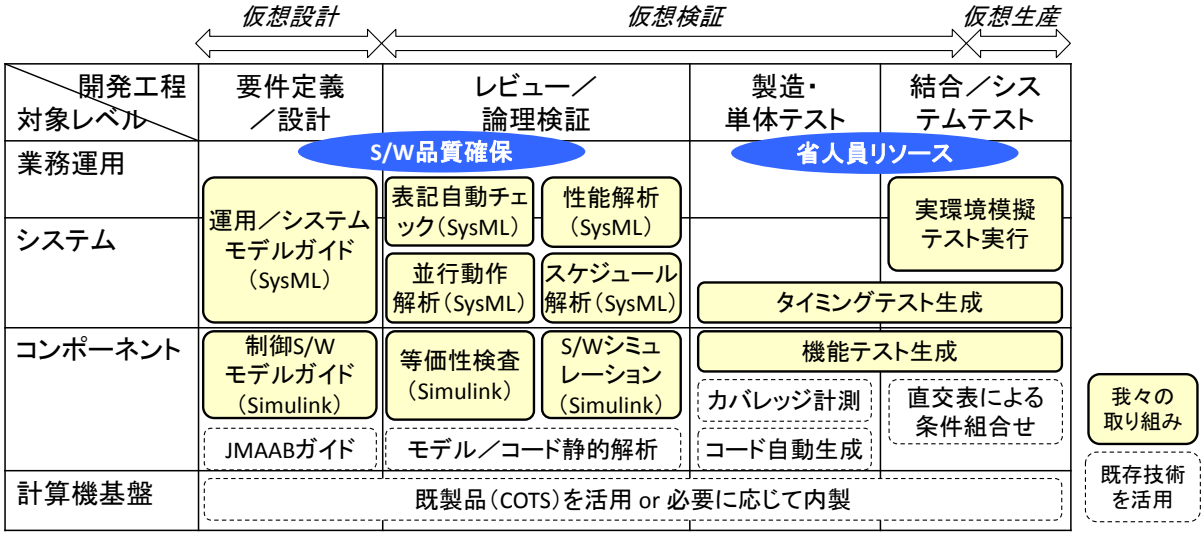

図 13 “Modeling, Simulation and Visualization”の取り組み概要
Figure 13 The overview of “Modeling, Simulation and Visualization”.