PART Ⅲ

検証事例

目次

1．はじめに

PARTⅢは、「検証」における適用事例を記載したものである。全体的な目的や共通事項は、「PART Ⅰ　概要」を参照願いたい。
ソフトウェア開発現場での認識として、設計工程における工夫の事例が多いと考えた理由は、「PART Ⅱ　設計事例」の「１．はじめに」で述べた通りである。設計時に考慮したことが確実に実装されているかを確認する「検証」が対として存在し、そこにも様々な取組みを行っている事例が多く存在すると考え、企業・団体に情報提供をお願いしたところ、設計と同様に様々な分野の企業・団体から、数多くの事例を紹介していただいた。
「４．各事例報告」で、詳細を紹介する。

なお、掲載している適用事例は、様々な技術や手法を、様々な目的、分野に適用しているものであり、それぞれに適用の可否を判断する前提条件がある。したがって、すべての企業やプロジェクトへの適用を推奨するものではないことをご了解のうえ、皆様の取組みに合った事例を参考にしていただきたい。

2. 事例一覧

PARTⅢで報告する検証事例の一覧を次に示す。

表 Ⅲ-2-1　収集事例一覧

	No.	標題	事例提供元
検証事例	B-1	宇宙システムにおける上流工程仕様の妥当性確認技術	(独)宇宙航空研究開発機構(JAXA)
	B-2	鉄道の機能安全(RAMS)認証支援のためのセーフティケース	(独)産業技術総合研究所(AIST)
	B-3	非機能要求グレードの大学ポータルサービスへの適用	名古屋大学
	B-4	冗長構成システム（クラウド等）の耐故障性に対する検証技術	(株)富士通コンピュータテクノロジーズ
	B-5	単体ランダムテスト実行／可視化ツール“Jvis”の適用事例	宮崎大学
	B-6	要求仕様明確化のための仕様記述技術(USDM)活用事例	(株)ベリサーブ
	B-7	形式手法を用いたセキュリティ検証	アーク・システム・ソリューションズ(株)
	B-8	形式仕様記述手法を用いた高信頼性を達成するテスト手法とその実践	フェリカネットワークス(株)
	B-9	Orthogonal Defect Classification 分析による欠陥除去と品質の成熟度可視化	オリンパスソフトウェアテクノロジー(株)
	B-10	モデル検査の適用による上流工程での設計誤りの発見	(株)東芝
	B-11	上流工程の要求を効率的に閉ループシミュレーションする次世代 SILS の開発	トヨタ自動車(株)

3. 検証適用事例の状況

PART Ⅰ　概要　にも記載したが、以下のような観点で事例を収集した。

（1） V 字モデルのプロセスにおける左辺の活動の成果が、右辺の検証の成果にポジティブに影響する。

（2） 左辺の活動としては、要求の早期獲得のみならず、より厳密な獲得が以降の活動における手戻りを減少させる。

（3） 結果として、左辺において、その中でもより上流、即ち要求獲得及び定義、更にそれをベースとした仕様記述への工数のシフトが起こる。

（4） 要求獲得及び定義、更に仕様記述に関わる活動として、モデル化及びモデルの実行あるいはシミュレーションが行われる。

（5） 結果として、上流における品質の作り込みが開発プロセス全体における手戻りを減少させると共に検証への工数を削減させ、生産性が向上する。

（6） 更に、リリース時の品質も向上する。

この仮定、特に「検証」工程に関するものは、ほぼ妥当であったと考えられる。それは、後述の各事例で証言されている事項をピックアップすることにより、確認することができる。なお、ここでは全ての事例を取り上げておらず、効果等がある程度わかるものを引用した。

【検証事例Ｂ－１】宇宙システムにおける上流工程仕様の妥当性確認技術

当初目的として設定した上流工程における仕様定義段階の３つの課題の改善状況は以下の通りである。

(a) 正確性の欠除に対して

チェックリストベースレビューにて多くの問題を早期に抽出できた。

仕様書間の一貫性をモデル検査技術により確認した。

(b) 不完全性の残留に対して

モデル検査技術により抽出可能となった。

(c) 使用方法・環境の想定不足に対して

モデル検査である程度抽出できるが、今後の改良が必要。

【検証事例Ｂ－３】非機能要求グレードの大学ポータルサービスへの適用

信頼性評価指標の定義表の列を D-Case の階層に対応付けることで、名古屋大学のポータルの信頼性を客観的かつ容易に確認できるようになった。

【検証事例Ｂ－４】冗長構成システム（クラウド等）の耐故障性に対する検証技術

「非機能要求グレード」を活用することにより、検証項目と確認内容の網羅性を確保しつつ、疑似故障ツールを用意したことにより、効率的に耐故障性の検証が可能になった。これにより、従来は実際の運用に入ってからでなければ見つけることができなかったハードウェアの故障を起因としたトラブル要因を未然に検出し、対策を講じることにより、運用品質の向上に貢献できた。

【検証事例Ｂ－６】要求仕様明確化のための仕様記述技術（USDM）活用事例

下記目標をほぼ達成できた。

(a) USDM を利用した要求（要求仕様）を記述するためのガイドラインを作成することで、安定した品質で作成が可能になった。

(b) 要求に対応する受入規準が明確になり、受入試験の項目作成で品質向上が図れた。

(c) テスト技術を利用して体系的に整理することで、トレーサビリティなどの影響分析に利用可能となった。

(d) W モデルでの担当する役割分担を明確にすることができた。

(e) 第３者による実施により、既存の技術者に少ない負担で行えた。

【検証事例Ｂ－８】形式仕様記述手法を用いた高信頼性を達成するテスト手法とその実践

形式手法記述を用いることによって、定められた品質目標を達成するために開発を行わなければならなかった２つの生成器を新たに開発する必要がなくなり、具体的に開発効率を向上させることができた。

【検証事例Ｂ－９】Orthogonal Defect Classification 分析による欠陥除去と品質の成熟度可視化

ODC 分析による障害原因の特定は、開発設計プロセスにおける改善領域を絞り込み主導的に手を打つ事を可能にし、信頼度成長曲線との組み合わせによって、最終ソフトウェアの品質を担保する説明責任を果たすことに寄与した。

ODC 分析では、試験を実施した障害発見者によって決定する属性と、開発サイドの障害発見者により決定する属性の両面から、ソフトウェアの品質を多面的に可視化できるため、開発者とテスターの協業が生まれ、ソフトウェアの品質を高めようとするベクトルを一致できる副次的な効果が大きかった。

【検証事例Ｂ-10】モデル検査の適用による上流工程での設計誤りの発見

組み込みシステム開発にモデル検査を適用し、実際に設計誤りを発見することができている。モデル検査の適用によって、１プロジェクトあたりの平均でテスト工程の約３０％のコストを削減できた。

4. 各事例報告

（番号の付与体系について）

各事例の報告に先立ち、番号の付与体系について触れておく。

以降の事例報告は、全体的な章立てから考えると、PART Ⅲの第４章に属するものであり、例えば事例の１番目は、PART Ⅲ．４．１という章立てになる。しかしながら、それぞれが独立の読み物としても成立するものであること、及び、その中に記載の番号体系が深くなりすぎると読者の混乱を招くことから、以下では次のような番号付与体系としている。

（１） PART Ⅲの第４章、すなわち「検証事例」を表す記号として「Ｂ」を用いる。上述の事例１番目の例では、「Ｂ－１」という表記を行うことにする。なお、PART Ⅱの「設計事例」においては、「Ａ」を用いて本篇の「検証事例」と区別している。

（２） ページ付与は、PART Ⅲで通し番号としており、Ⅲ－１ 、Ⅲ－2、・・・としている。

B－1

宇宙システムにおける上流工程仕様の妥当性確認技術 [1]

1. 概要

本編では、宇宙システムに搭載されるソフトウェアの開発において、モデル検査やチェックリストベースレビューを適用した、独立行政法人 宇宙航空研究開発機構(以下、同機構とする)の事例を紹介する。

宇宙システムに代表されるセーフティクリティカルシステムでは、要求された機能が諸条件の元で動作するという信頼性の向上のみではなく、安全性の視点でもその健全性の検証が求められる。特に、ソフトウェア開発において、仕様どおり動作することの保証は、従来の品質工学、信頼性工学に基づく手法により強化が可能であるが、要求仕様等の上流工程における仕様定義時の正確性の欠如、不完全性の残留、想定不足がシステムの安全性に脅威をもたらすことが多い。

宇宙分野でも、NASA 等の過去の事故事例からの教訓として、上流工程の仕様に関する問題に起因した事象が注目されている。JAXA(独立行政法人　宇宙航空研究開発機構)では、この解決のために、ソフトウェア IV&V（独立検証及び妥当性確認）活動として、「モデル検査」や過去経験知に基づく「チェックリストベースレビュー」を導入している。

宇宙業界標準の開発プロセスとして、同機構では、ソフトウェア開発標準(JERG-0-049)を定めており、その中で、妥当性確認を表 B-1-1 のとおり定義している。

表 B-1-1　検証および妥当性確認　（JERG-0-049 から引用）

用語	説明
検証	客観的証拠を提示することによって、規定要求事項が満たされていることを確認すること。
妥当性確認	客観的証拠を提示することによって、特定の意図された用途または適用に関する要求事項が満たされていることを確認すること。

図 B-1-1 に示すとおり、検証はあるリファレンスに基づいて評価されるものであるが、妥当性確認は、顧客や想定環境から抽出された上流要求に対し、開発全工程を通じてその成立性を確認するものである。本事例では、妥当性確認の視点で導入してきた「モデル検査」及

[1] 事例提供：独立行政法人 宇宙航空研究開発機構 情報・計算工学センター 安全・信頼性推進部 片平 真史 氏

び「チェックリストベースレビュー」の取組みをその目的や期待効果とともに解説する。

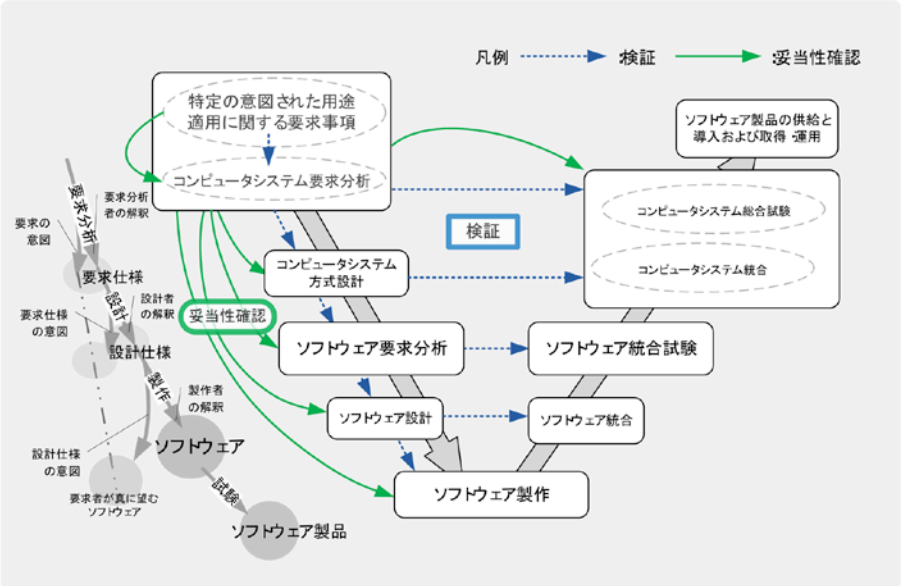


図 B-1-1　検証と妥当性確認の概念図　（JERG-0-049 から引用）

2. 取組みの目的

ソフトウェア開発の上流工程における要求仕様に関する問題に起因して、宇宙分野では以下の事故が発生している。

- NASA(アメリカ航空宇宙局)が 1998 年 12 月に打上げた火星探査機 MCO(マーズ・クライメイト・オービター)は、1999 年 9 月 23 日に火星軌道進入の際に、通信が途絶えた。事故調査の結果、予定されていた 140-150 km よりも低い 57 km の軌道で侵入してしまったことが主原因とされたが、これは、軌道モデルで使用されるソフトウェアの単位系の取り違え（メートル法のはずが誤ってヤードポンド法となっていた）により発生したとされている。**＜正確性の欠如＞**
- NASA が 1999 年 1 月に打上げた火星探査機 MPL(マーズ・ポーラー・ランダー)は、1999 年 12 月 3 日に、火星大気圏へ突入する際に、探査機からの信号が途絶え、行方不明となった。信号途絶の理由は現在でも不明であるが、最も可能性の高い原因として、着陸用の脚の展開時の衝撃を着陸と誤って判断したソフトウェアの問題とされている。また、この根本原因として、センサの振舞いに関する情報がソフトウェア要求仕様に欠落したと言われている。**＜不完全性の残留＞**
- ESA(欧州宇宙機関)が 1996 年 6 月 4 日に打上げたアリアン 5 (Ariane 5)初号機は、打上げ後 37 秒後に予定していた飛行経路を大きく逸脱し爆発した。事故調査の結果、一つの主要要因はアリアン 4 から再利用されている実行されないはずのソフトウェアが動作したためである。破壊に至った一つの副要因として、想定外のセンサ

の出力に対する処理が欠落していたことが挙げられている。**＜想定不足＞**

・ 本事例で紹介する取組みは、これらの過去の事故・重大不具合の経験に基づき、ソフトウェア開発の上流工程における要求仕様等の仕様定義時に、正確性の欠如、不完全性の残留、使用方法・環境の想定不足を極力排除し、セーフティクリティカルシステムの安全性を向上させ、事故を未然回避することを目的としている。

3. 取組みの対象、適用技術・手法、評価・計測

3.1. 取組みの対象製品と工程

ここでの取組みは、宇宙システムの中でも、変更が困難であり、また、短時間でセーフティクリティカルな判定が必要となる「人工衛星システム」や「ロケットシステム」に用いられる搭載ソフトウェアを対象として紹介する。

また、ここで紹介する取組みは、上流工程の仕様定義時の妥当性確認に限定しているが、実際の開発作業においてはそれ以外の工程にも適用されている。

3.2. 適用技術・手法

ソフトウェア開発の上流工程における要求仕様等の正確性の欠如、不完全性の残留、使用方法・環境の想定不足を排除するために、適用した技術・手法である「モデル検査」及び「チェックリストベースレビュー」をここでは紹介する。この「モデル検査」及び「チェックリストベースレビュー」は、技術・手法として、それぞれのデメリットを補完しながら使用している。通常 IV&V では、目的（観点）に対して、図 B-1-2 に示すように複数の手法を組合せ・選択し使用している。それぞれの技術・手法のメリット、デメリットを意識した適用が課題抽出を最大化するためには重要なポイントとなる。

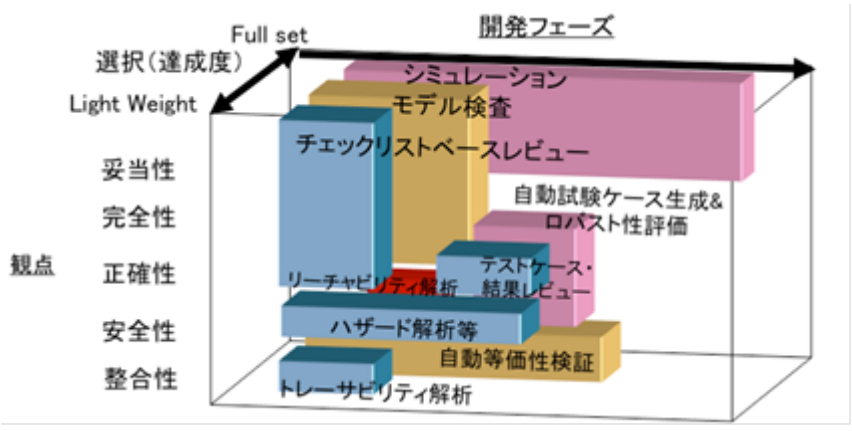


図 B-1-2　IV&V 手法の組合せ・選択

モデル検査技術は、適切なモデル作成及び検査目的を設定することにより、モデルやツー

ルにより、複雑な事象を検査し、思いつきにくい想定シナリオを抽出、検証できる。

しかしながら、モデル検査技術のデメリットとして、比較的小規模な宇宙分野の搭載ソフトウェアであっても、ソフトウェア全体に対し、詳細な動作まで機能をモデル化し検証することは困難である。この解決のためには、モデル検査では困難な大局的な確認を行う必要がある。過去の事故・不具合や開発時の留意事項に基づいた経験則から作成されたチェックリストを用いて、人がレビューを行い、検査を比較的網羅的に実施している。このチェックリストベースレビューのデメリットは、検査対象の挙動が人により可読で、検討できる程度の単純な内容であることが点検内容の限界となることである。

モデル検査技術としてここで紹介するものは、適用実績のある手法のうち、以下の検査技術を代表事例として導入した時系列順に紹介する。

（1） SpecTRM (Specification Tools and Requirements Methodology)

（2） SPIN

また、チェックリストベースレビュー手法として、チェックリストの概要とレビュー方法について紹介する。

4. 取組みの実際、及び実施上の問題、対策・工夫

4.1. モデル検査技術

モデル検査技術は、システムのある性質に対しての「振る舞い」「状態遷移」等をモデル化することにより、ツール上で仕様内容をモデル検証することができる。すなわち、適切なモデル作成及び検査目的を設定することにより、複雑な事象を検査し、思いつきにくい想定シナリオを抽出できる。

4.1.1. SpecTRM (Specification Tools and Requirements Methodology)

SpecTRM は、米 MIT(マサチューセッツ工科大学) Nancy G. Leveson が開発した方法論である。また、Safeware Engineering 社によりシステム／ソフトウェア安全設計支援ツールとして市販されている。SepcTRM は、形式的仕様記述 SpecTRM-RL(Requirement Language)により構成され、

図 B-1-3 に示す Intent Specification で体系化された安全設計をガイドする統合化手法である。これらの手法は、過去の不具合事例の解析結果から必要となる安全上の考慮事項を網羅できるように構築されている。SpecTRM を利用することにより、Intent Specification に沿った対象システム／ソフトウェアのモデリング、モデル検査、シミュレーションを行なうことが可能となり、ガイドに従って安全設計を進めることができる。

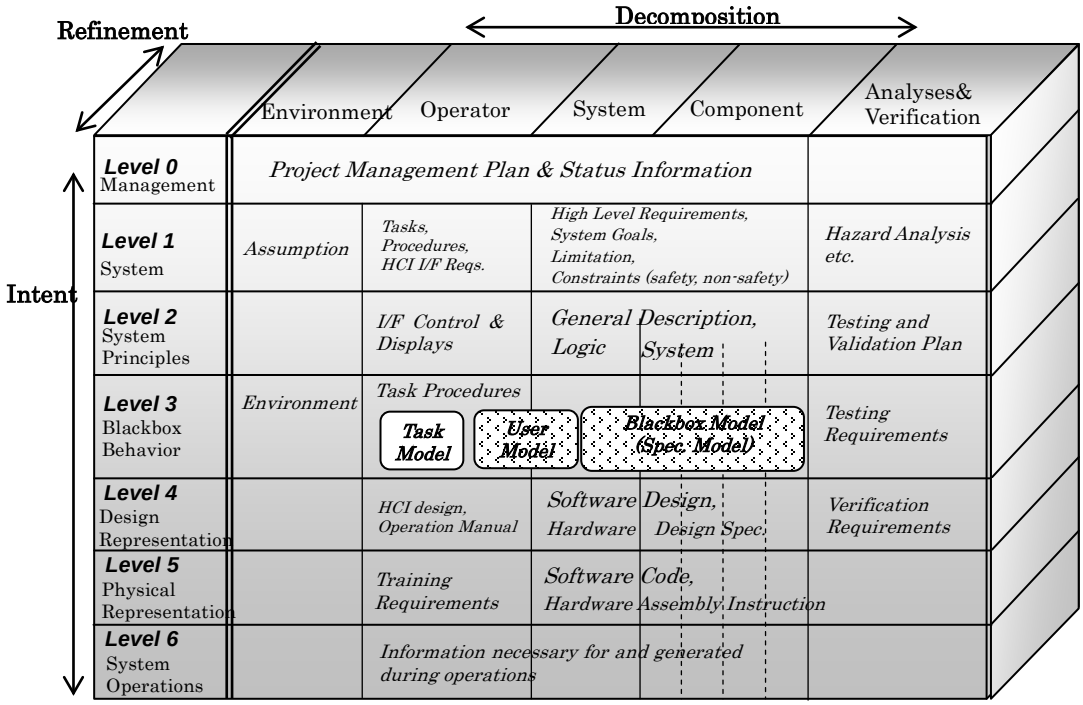


図 B-1-3　Intent　Specification の構造

モデリング及びモデル検査を行うことにより、通常の自然言語による要求仕様のレビューでは判断することが難しい、仕様の一貫性(状態遷移先が複数発生することはないか)や、完全性（予想されない状態に陥ることはないか）の問題を抽出することができる。このモデリング及びモデル検査は図 B-1-3 の Intent Specification では Level3 活動と対応付けられる。SpecTRM の特徴として、検査対象として選定した限定的な範囲で、システムが問題のある状態へ陥らないことを網羅的に自動検査し、上流工程からシステムの完全性・一貫性をモデル検査で検証しながら設計を進めることができる。

- 要求段階の抽象的な仕様から検査可能で、上流工程で問題を見つけることが可能
- ツールにより自動的に網羅的な検証が可能(他の方法で必要な検査式の設定がいらない)
- SpecTRM ツールは、Intent Specification Level3 にモデル検査として、以下の 2 つの自動分析用の「モデル解析機能」を持っている。（図 B-1-4 参照）
- 完全性分析 Completeness Analysis : 状態遷移に関する遷移条件の抜けがないこと
- 一貫性分析 Deterministic Analysis：遷移先が一意でない遷移条件がないこと

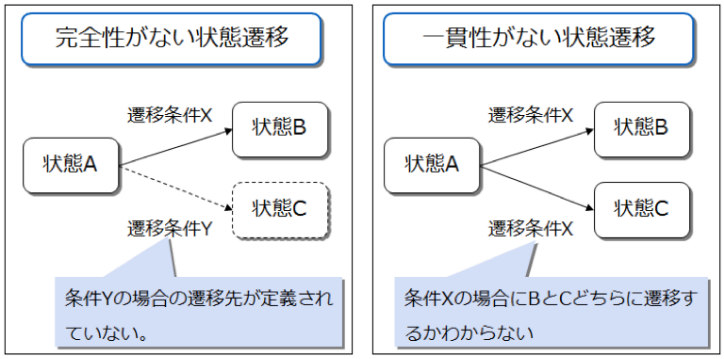


図 B-1-4　SpecTRM のモデル解析機能

SpecTRM モデルは、モデル記述言語 SpecTRM-RL で作成することになる。モデル作成は、システム開発におけるプログラミングに相応する作業と言え、モデル記述言語の文法等を学習する必要がある。検査モデルには、まずシステムの振る舞い（状態遷移仕様）を定義する必要があり、また状態遷移を表現するために、環境変化の表現、入力情報・出力情報等を文法に沿って定義する。

―SpecTRM-RL の 6 つのモデル要素―

（1）　入力(Inputs；機器・センサからの入力とオペレータ（主にディスプレイ）からの入力を区別して定義可能)

（2）　出力(Outputs；機器・センサへの出力とオペレータ（主にディスプレイ）への出力を区別して定義可能)

（3）　状態(States)

（4）　モード(Modes)

（5）　関数(Functions)

（6）　マクロ(Macros)

この 6 種類の各モデル要素の関係を図 B-1-5 に示す。対象システムの入出力情報や処理情報を識別し、各モデル要素でシステム仕様を表現し SpecTRM に登録していく。ただしモデル要素は全てを使う必要はなく、検証内容に応じてモデルを取捨選択することが可能である。

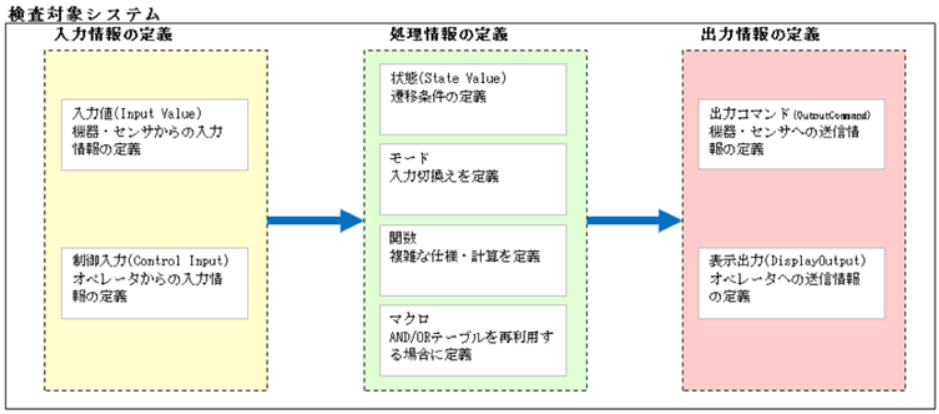


図 B-1-5　SpecTRM-RL の 6 要素の関係

導入効果：

必要なシステムや人の動作をモデル化することにより、その完全性や一貫性を自動的に検査でき問題点を抽出、検証できる。完全性解析結果は、遷移先定義が無いことを意味するが、定義が不要なパターンも出力される可能性があるため、出力内容を分析し、各パターンの意味付けを行う必要がある。一方、一貫性解析結果は、遷移条件が一意に決まらない状態遷移があることを意味するが、遷移不要なパターンも出力される可能性があるため、出力内容を分析し、各パターンの意味付けを行う必要がある。この結果、意味のあるとされたものは、「想定外・障害発生時のケース」に相当する。すなわち、この状態になった場合にシステムがどうすべきか決まっていないことを示しており、安全上の問題をもつものであるため、対応策を検討する必要がある。ここでは紹介しなかったが、安全性、一貫性の自動検査機能の他、入力値を与えて挙動の確認を行うシミュレーション機能もある。

SpecTRM を用いた検査では、システムの想定外・障害発生時のケース抜けを通常１システム辺り数十点の問題を抽出できる。また、対象システムの挙動と利用者の運用手順との不成立箇所の抽出にも利用され、手順・マニュアル開発に反映できる。

特徴：

- 様々な仕様を分析する際、モデルは比較的簡単にコピーができるため、コピーにより仕様を変えて自由度の高い分析を実施することができる
- 状態遷移や遷移条件等の情報が統一フォーマットで集約されるので、仕様の確認をする場合に設計書を見なくても必要な情報が得られやすい
- ツールがモデルの管理機能及びインポート・エクスポートを持っているため、過去に使用したモデルを蓄積し再利用する仕組みを用意すると、検査作業の効率化が図れる

課題：

以下のケースには向いていない。

- 複数の状態遷移が時間の経過とともに連携しながら発生するケース
  （SpecTRM は状態遷移変化を伝搬し探索することが困難なため）
- 状態遷移の実行条件がシンプルなケース
  （モデリング工数がかかる割には、複雑な条件に対する検査結果とならない）

4.1.2. SPIN

SPIN は、現 JPL Gerard J.Holzmann らが開発したモデル検査ツールである。この手法は、検査モデル作成に必要な情報（状態、実行条件、実行内容、外部イベント、制約条件等）が明確になっている工程が対象となる。特に、対象ソフトウェアが仕様どおりに稼働しない状態の検出、対象ソフトウェアの実行条件・状態遷移等に矛盾及び抜けがないことの検証に向いている。SpecTRM に比較して、より仕様が具体的に定義された段階に導入しやすい。

この SPIN による検査には以下の入力情報を用いて作業を行う。すなわち、これらの情報が明確に定義できる段階から適用することが適切である。

■ システム機能仕様情報
- システム機能の処理内容
- システム機能の起動条件
- システム機能の処理フロー等

■ 検証条件情報
- システム異常状態の定義情報（エラー出力する状態）
- システム正常終了の情報等（正常終了でない状態を異常状態として使用する）

検査結果として、以下の結果を得ることができる。

■ 検証条件の成立可否
■ 反例情報（初期状態から検証条件を満たす状態までの経緯）
■ 状態遷移の環境情報（検証条件を満たした状態の、状態管理変数の値等）

検査結果として得られた反例情報を一つずつ実際に問題であるかどうかを評価しながら判定していくことになる。

導入効果：

比較的仕様が明確になってきた場合に導入すると、SpecTRM では検査が困難な、複数の状態遷移が時間の経過とともに連携しながら発生する問題を抽出することができる。

宇宙システムに適用を行い､設計の初期段階の仕様に対して危険な状態になるシナリオ(反例情報）を識別することができた。このシナリオは、安全上問題に繋がる可能性の高いものであり、対策検討につながった。

課題：

効果を上げるためには、反例情報の分析及び検証条件の設定をある程度定型化しながら行う等の工夫が必要である。

4.1.3. モデル検査技術の導入上の課題と解決策

モデル検査技術の同機構における導入事例から以下の経験則がある。

- 強力なモデル検査の他、モデル検査の前のモデル作成段階で多くの仕様上の問題を抽出できる

また、モデル検査技術の導入上の課題と同機構における解決策は以下のとおりである。

- モデル検査に依存し過ぎると大観的な重要な問題を見逃すことがあるが、チェックリストベースレビューなど他の手法との組合せによってこれを防ぐことができる。
- システム及び環境をすべてモデル化することが困難なため、誤った絞り込みや抽象化を行うことにより、重大な問題を見逃すことがある。作業実施前に確認したいシナリオを分析し、関係する範囲を事前に仕様設定者と確認を行うことで回避できる。
- モデル化の段階及び検査結果から正しく問題点を抽出するためには、対象システムのドメイン知識がないと誤った問題の抽出となるため、ドメイン知識を習得してか

ら、またはドメイン知識を有する者と作業を行うことが望ましい。

・ モデル検査はモデル化した範囲である程度網羅的に特徴抽出を行うことができる反面、連続系の制御則の様なシステムを検査することが困難である。特定の検査したい対象シナリオが明確になっている場合は、モデルを動作（シミュレーション）させてランダム入力に対する検証を行うことが有効である。

4.2. チェックリストベースレビュー手法

4.2.1. ボイジャー・ガリレオチェックリスト

同機構が現在利用しているチェックリストの一つとして、1996 年に JPL（NASA ジェット推進研究所）Lutz が発表した Safety Checklist[1]がある。通称、ボイジャー・ガリレオチェックリスト（表 B-1-2）である。

表 B-1-2　ボイジャー・ガリレオチェックリスト

Interfaces
(1) Is the software's response to out-of-range values specified for every input?
(2) Is the software's response to not receiving an expected input specified? (That is, are timeouts provided?) Does the software specify the length of the timeout, when to start counting the timeout, and the latency of the timeout (the point past which the receipt of new inputs cannot change the output result, even if they arrive before the actual output)?
(3) If input arrives when it shouldn't, is a response specified?
(4) On a given input, will the software always follow the same path through the code (that is, is the software's behavior deterministic)?
(5) Is each input bounded in time? That is, does the specification include the earliest time at which the input will be accepted and the latest time at which the data will be considered valid (to avoid making control decisions based on obsolete data)?
(6) Is a minimum and maximum arrival rate specified for each input (for example, a capacity limit on interrupts signaling an input)? For each communication path? Are checks performed in the software to avoid signal saturation?
(7) If interrupts are masked or disabled, can events be lost?
(8) Can any output be produced faster than it can be used (absorbed) by the interfacing module? Is overloaded behavior specified?
(9) Is all data output to the buses form the sensors used by the software? If not, it is likely that some required function has been omitted from the specification.
(10) Can input that is received before startup, while offline, or after shutdown influence the software's startup behavior? For example, are the values of any counters, timers, or signals retained in software or hardware during shutdown? If so, is the earliest or most-recent value retained?
Robustness
(11) In cases where performance degradation is the chosen error response, is the degradation predictable (for example, lower accuracy, longer response time)?
(12) Are there sufficient delays incorporated into the error-recovery responses, e.g., to avoid returning to the normal state too quickly?
(13) Are feedback loops (including echoes) specified, where appropriate, to compare the actual effects of outputs on the system with the predicted effects?
(14) Are all modes and modules of the specified software reachable (used in some path through the code)? If not, the specification may include superfluous items.
(15) If a hazards analysis has been done, does every path from a hazardous state (a failure-mode) lead to a low-risk state?
(16) Are the inputs identified which, if not received (for example, due to sensor failure), can lead to a hazardous state or can prevent recovery (single-point failures)?

Lutz は、1991 年に Jaffe, Leveson らが過去の事故データの分析結果に基づき提唱したセーフティクリティカルのクライテリアを参考にチェックリストを作成した。また同時に NASA の探査機ボイジャー(Voyager)及びガリレオ（Galileo)で実際に発生した 192 件の不具合を詳細に分析し、チェックリストを検証した。このチェックリストを用いることで、192 件の不具合のうち 77%である 149 件は、要求分析段階で発見できたことが分かった。

4.2.2. 同機構独自チェックリスト

4.2.1.で紹介したものは、JPL の不具合分析の結果を活用したボイジャー・ガリレオチェックリストの導入例であるが、日本の宇宙業界でも同種の不具合が発生しており、これらの不具合情報や開発・検証の経験者の知見を体系化した同機構独自のチェックリストを整備、利用している。表 B-1-3 に姿勢制御系チェックリストの例を示す。

表 B-1-3　姿勢制御系チェックリスト　（抜粋）

項目	内容	詳細	事例	対象文書
入出力データ	物理量データ毎に単位が明確に記述されているか？	搭載ソフトウェアの中では、角度は、ラジアン単位、度単位が混在して用いられているので、単位の扱いについては十分な注意が必要である。	設計基準に従って入出力データインタフェースは統一されているが、既開発品の利用により、機器毎に異なる場合がある。	データ仕様
ON/OFF 手順	ON/OFF 手順、および ON 時データ処理手順は明記されているか？	使用するセンサによっては、電源 ON 後、センサ信号を用いるまでにマスキング等の初期設定操作を必要とするものがある。	マスキングをしていない場合、誤った動作をしたことがあった。	要求仕様書
異常信号処理	異常信号（雑音）に対して処置策が施されているか？	パルス信号に雑音が重畳し、その結果、物理的に可能な回転速度範囲を逸脱していないかをソフトウェア側で絶えずチェックする必要があり、要求仕様に規定されているか？		要求仕様書
フォールトトレラント設計	モード別の FDIR の設計が妥当か？	故障耐性の設計として、モードごとの故障解析（FMEA、FTA）の有無、クリティカル故障の抽出、それに対する設計上の対応策（FDIR 機能)がとられているか？冗長構成をとるのが難しい、あるいは不可能な部分を除いて単一故障点が回避された設計となっているか？	故障を特定できるか？センサを使わないモードで FDIR が機能していないか？	要求仕様書

4.2.3. チェックリストレビューの導入効果

ボイジャー・ガリレオチェックリストや姿勢制御系チェックリスト等を利用してレビューを行うことにより、通常のレビューよりも過去の不具合事例・経験知に基づく確認ポイントを確実にチェックでき、問題ない場合も含み点検結果の記録を残すことができる。ボイジャー・ガリレオチェックリストでは、宇宙システムの適用を通じて、要求仕様上の問題点を通常 1 システムにつき数点の問題点が要求定義段階で抽出できる。タイムアウト時間の設定がなく、待ち続けてしまうケースや、信頼性の観点でバックアップ計算機に切替える場合にシ

ステムの状態によっては、切替信号を送信されずにシステムが停止するケースが発見され、重大な事故の原因を要求段階で未然に抽出・解決できることが実証できた。また、姿勢制御系チェックリスト等の場合、単位系等の初歩的な間違えから想定外の故障時の処理機能の抜け等複雑な問題まで幅広いレベルの要求仕様上の問題点を抽出でき通常 1 システムにつき数十点の問題点が要求定義段階で抽出できる。2.で示した MCO の例の単位系の誤りもこのチェックリストを利用してチェックしていれば要求定義段階で発見できたはずである。同機構では同様なチェックリストを 5 つのシステム分類に対し、準備しており、重大な事故の原因を要求段階で未然に抽出・解決できることが実証できた。

4.2.4. チェックリストベースレビューの導入上の課題と解決策

チェックリストを用いたレビュー活動は、過去の不具合事例・経験知に基づく確認活動が可能となるが、課題としては、問題点を正しく抽出できるか否かが、レビュー活動を行う作業担当者のスキルに依存してしまうことである。また、限られた時間の中ですべての確認項目を同様の深さで確認することは期待する結果が得られないことになる。

このチェックリストを用いたレビューの質・効率を作業担当者のスキルに極力依存しないように、同機構では、それぞれのチェック項目に、詳細なチェック方法を補足説明している。以下に人工衛星姿勢制御系サブシステムのチェックリストをサンプルに解説する。（図 B-1-6）

例えば、チェック項目(2)

Is the software's response to not receiving an expected input specified? (That is, are timeouts provided?) Does the software specify the length of the timeout, when to start counting the timeout, and the latency of the timeout (the point past which the receipt of new inputs cannot change the output result, even if they arrive before the actual output)?

詳細確認ポイント：

- 安全上重要なソフトウェアに対し期待される入力値にタイムアウト時間が設定されていか？
- タイムアウト時間が設定されていない場合、対象のソフトウェアの挙動が安全であり、期待どおりに動作するか？
- 必要なハードウエア、インタフェースは存在するか？また、他の事象によりインタフェースが利用できないことはないか？
- タイムアウト時間はインプット、アウトプット側で定義・了解されているか？
- ・・・など。

図 B-1-6　人工衛星姿勢制御系サブシステムの補足説明例

また、複数の作業担当者で共同して実施する場合は、チェック項目、詳細確認ポイントをスキルに合わせて配分し、完了後に作業担当者間で確認を行うことで作業の質・効率を向上できる。

優先度： 高、中、低

重要な機能に関わるチェック項目、あるいは問題が多く見られる項目の優先度

難易度：

易 … ひとつの項目のチェックを、一人の担当者が 15 分以内で行える

中 … 30 分以内

難 … 1 時間以上

姿勢制御系チェックリストのチェック項目の構成と所要時間を表 B-1-4 に示す。

表 B-1-4 チェック項目の内訳

優先度	高			中			低			計
難易度	易	中	難	易	中	難	易	中	難	
センサ A	6		1	5			1			13
センサ B	3	1	1	4			1			10
センサ C				4	2					6
センサ D	2						4		1	7
センサ E				1	1	1				3
アクチュエータ A	6	1					1	1	1	10
アクチュエータ B	3			4			3	1		11
全体システム	1	1	2			1	4	2	3	14
計	21	3	4	18	3	2	14	4	5	74
作業時間 (h)	5.25	1.5	4.0	4.5	1.5	2.0	3.5	2.0	5.0	29.25

実際の利用時には、対象となる宇宙システムの特徴・リスク及び配分できる作業予定時間から、それぞれの装置に対して、優先度のレベルを選択し、点検項目を決定する。あらかじめ定義された難易度に合わせて、ボイジャー・ガリレオチェックリストと同様に作業担当者のスキルと照らしながら、作業分担を行っている。

5. 達成度の評価、取組みの結果

今回、適用した技術・手法である「モデル検査」及び「チェックリストベースレビュー」の個別の導入の効果は、4 で述べたとおりである。ここでは、当初目的として設定した上流工程における仕様定義段階の 3 つの課題の改善状況について表 B-1-5 にまとめる。

表 B-1-5 上流工程における仕様定義の改善状況

上流工程の仕様の課題	改善状況
正確性の欠如	仕様中の単位誤記･正負／極性反転等の初歩的であるが重大な事故を引き起こす過誤等､記述の曖昧さに起因する複雑な挙動のチェックリストベースレビューにて多くの問題を早期に抽出できる。また､仕様書間の一貫性をモデル検査技術により確認することで､人がチェックしにくい複雑な利用シナリオ上の過誤を検出可能となる。

不完全性の残留	仕様書の記載が不足・欠落したことによるアルゴリズムや機能の欠落・欠陥については、完全性をモデル検査技術により抽出可能となる。しかしながら、本来のシステムに期待される正しい動きを理解・把握する人とともに、意図する挙動かどうか最終確認することが、より精度を高めることにつながる。また、仕様記述の際に欠落しやすい情報についてはチェックリストなどに含め、レビューにて点検することが更に効果を上げることになる。
使用方法・環境の想定不足	システムが動作する環境が全て網羅され仕様が設定されていることは少ないのが一般的であるが、特に安全確保のためには、使用方法・環境の想定を超えた場合でも安全な動作が期待される。環境や使用方法も含めてモデル検査を行うことで、想定しない危険シナリオ等をある値度抽出することができるが、今後の改良が必要である。

参考文献

[1] Robyn R. Lutz "Targeting Safety-Related Errors During Software Requirements Analysis," The Journal of Systems and Software, Vol. 34,
Sept, 1996, pp. 223-230.

B－2

鉄道の機能安全(RAMS)認証支援のための セーフティケース [1]

1. 概要

本編では、独立行政法人 産業技術総合研究所、および西日本旅客鉄道株式会社が、鉄道分野において機能安全規格の適合を示す Goal Structuring Notation (GSN)を用いたセーフティケース作成支援とアセスメント支援の方法を開発した例を紹介する。

2. はじめに

多くのシステムは規模の拡大や多くの機能の実現のために複雑化が進む中､航空機､鉄道、自動車など様々な高信頼システムにおいては高い安全性も同時に求められている。

特に近年、安全性の中でも機能安全という考え方が注目されており機能安全の最上位の規格である IEC 61508 をはじめ、各分野において機能安全規格が発行され、その対応が求められている。鉄道も高い安全性を求められる分野であり、機能安全規格が存在する。

鉄道分野においては機能安全規格の適合を示す際には、セーフティケースという安全性を示す文書を作成しそれを基にアセサーがアセスメントを行う。この安全性を示すセーフティケースという文書は鉄道分野の機能安全規格の一つである EN 50129 / IEC 62425 に記載内容が記述されている。それによると計画から始まり、検証、妥当性確認と非常に詳細な記述が必要である。そのため、セーフティケース作成には非常に大きなコストが必要となる。また、詳細な記述が必要となるためその内容に漏れ抜けが発生しやすくなるという問題点もある。一方でセーフティケースを基にアセスメントを行うのだが、その内容が規格の要求事項にどのように従っているのか、記述の根拠理解が困難という問題も存在している。

これらの課題を解決するために GSN を用いたセーフティケース作成支援とアセスメント支援の方法を開発した。GSN はヨーク大学の Tim Kelly らによって開発された、保証のために構造化された議論の記述方法である。ゴール指向要求分析方法論 KAOS や Fault Tree Analysis (FTA)と類似した構造を持つ。対象となる機能安全規格を GSN により記述しこれのテンプレート化を行う。また、セーフティケースも規格に沿ってその記述内容をテンプレート化する。これらの二つを連携させることで、開発時には成果物を規格に沿った GSN に配置することでセーフティケースの対応する部分に内容が埋め込まれ、アセスメント時にはセーフティケースに対応する GSN を参照することで規格の対応する部分やその記述の根拠

[1] 事例提供:　独立行政法人 産業技術総合研究所／西日本旅客鉄道株式会社
相馬　大輔 氏、田口　研治 氏、西原　秀明 氏、大岩　寛 氏／矢田部　俊介 氏、森　崇 氏

を明確にすることができ、双方の支援が可能となるのである。

3. 取り組みの目的

鉄道分野においても他の高信頼システムと同様に安全性は非常に重要であり、機能安全規格への適合は大きな課題の一つである。鉄道分野における機能安全規格は統合した鉄道システム全体に対する EN 50126 / IEC 62278 を基に、鉄道通信に関する EN 50159 / IEC 62280、鉄道信号システムに関する EN 50129 / IEC 62425、鉄道ソフトウェアに関する EN 50128 / IEC 62279 さらに、複数のハードウェアに関する物など多数存在している(図 B-2-1)。

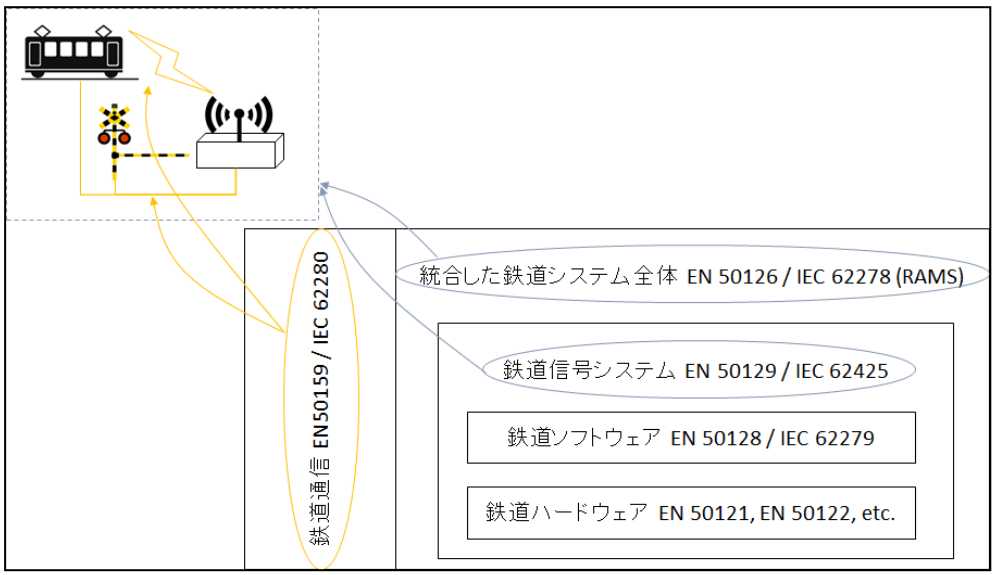


図 B-2-1　鉄道分野の機能安全規格

これらの規格への適合性アセスメントはセーフティケースという安全性を示す文書を作成し、それを用いて行われる。EN 50129 / IEC 62425 においてセーフティケースに記述されなくてはならない内容が規定されており、その内容は安全管理、品質管理、技術的な安全性などの計画、分析、検証と妥当性確認、担当者とその独立性など非常に詳細なものとなっている。そのため、機能安全規格への適合性アセスメントにおいて開発者側、アセサー側の双方に課題がある。

すでに述べたように規格適合性アセスメントに使用するセーフティケースは非常に詳細な記述が必要であり、記述のボリュームも多くなり、セーフティケース作成に非常に大きなコストがかかってしまう。さらに、適合性アセスメントの対象とする規格(以下アセスメント対象規格と呼ぶ)とセーフティケースの記述内容が記述されている規格(IEC 62425)が異なり、互いに参照しているものの詳細にアセスメント対象規格のどの要求事項がセーフティケースのどの記述と関係しているかということが明確でない。そのため、コストがかかってしまうことに加えてセーフティケースの記述内容に抜け漏れが生じやすくなってしまっているので

ある。また、アセスメント対象規格とセーフティケースの関係が不明確であることは規格への適合性評価を行うアセサーにとっても問題となる。アセスメントはセーフティケースを基準に行われるため関係が不明確であると、その記述内容からアセスメント対象規格のどの要求事項を満足しており、どの要求事項が満足されていないかを判断することが難しくなってしまうからである(図 B-2-2)。

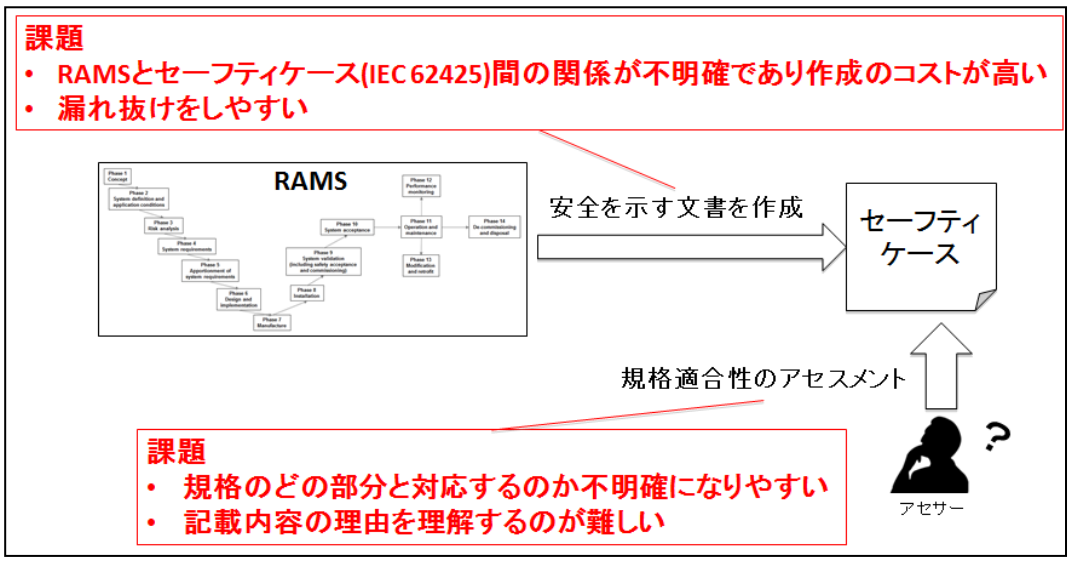


図 B-2-2　現在の課題

以上のような問題点を解決するために、アセスメント対象規格とセーフティケースの記述内容の関係を明確にし、アセスメント対象規格に乗っ取り実施された開発においてセーフティケース作成の支援、規格への適合性アセスメントの支援可能とする方法を開発することを目的とした。

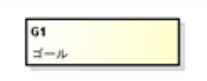
4. 取り組みの対象

本事例ではシステムが機能安全規格への適合性アセスメントを受ける段階を考えており、鉄道分野の安全関連のシステムがその対象となる。適合性アセスメントを受ける対象機能安全規格は EN 50126 / IEC 62278 Railway Applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)であり、セーフティケースは EN 50129 / IEC 62425 Railway applications – Communications, signaling and processing systems – Safety related electronic systems for signaling にしたがって考えている。

本件では目的を達成するために Goal Structuring Notation (GSN)という技術を用いた。GSN はヨーク大学の Tim Kelly らによって開発された、保証のために構造化された議論の記述方法である。ゴール指向要求分析方法論 KAOS や Fault Tree Analysis (FTA)と類似し

た構造を持つ。GSN はゴール、ストラテジー、ソリューション、コンテクスト、アサンプション、ジャスティフィケイショというノードを Supported by、In context of という二つの関連によりつなぎ合わせて議論を構造化する。以下、表 B-2-1 にノードの簡単な説明を、図 B-2-3 に GSN の例を挙げる。

表 B-2-1　GSN のノード(代表的な物)

	**ゴール:システムが満足するべき目標(部分目標)。**
	ストラテジー:論証の方法。ゴールからサブゴールを導く方針。
	コンテキスト:議論の背景や文脈。
	ソリューション:論証をサポートする具体的な証拠。

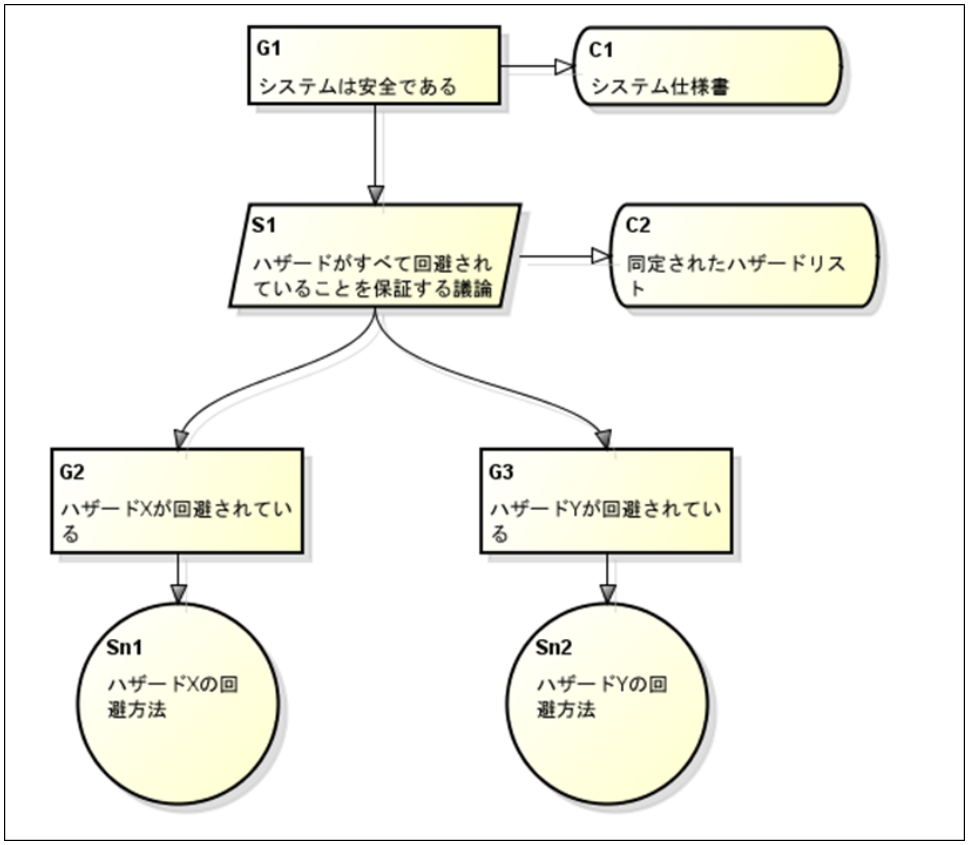


図 B-2-3　GSN 例

まず、IEC 62278 の GSN による記述を行った。これにより IEC 62278 がどのようなどのようにして安全性を担保しているかが明確になる。作成した GSN の各ソリューションやコンテキストなどには IEC 62278 のライフサイクルで作成される成果物(またはその一部が)配置される。作成した GSN を基に成果物(またはその一部)の部分をパラメータ化することで IEC 62278 の GSN テンプレートを作成する。一方で IEC 62425 を基に開発の成果物(またはその一部)の内容を記述する部分をパラメータ化しセーフティケースのテンプレートも作成する。これら二つのテンプレートのパラメータを関連づけることにより IEC 62278 とセーフティケースの関連が明確化し目的を達成可能となる。

取り組みあたり GSN を用いたが、適用に対して次のような考慮点が存在する。

(1)　GSN の適用に対しての準備、トレーニング、スキル

GSN は保証のために構造化された議論の記述方法である。ノードやそれらの関連のつけ方に関しての基本的な学習は GSN Community Standard Version 1.0 や Tim Kelly の博士論文 Arguing Safety – A Systematic Approach to Managing Safety Case などにより可能である。他にも海外では Safety Case 作成、管理、保守に関するトレーニングも実施されている。

また、GSN で使用するノードは非常に単純な図形であるため、様々な Power Point などといった様々なツールにより記述が可能である。しかし、実際はノード間の関係の書き方など記述できること、できないことがきちんと決められている。そのため、GSN の記述の際には専用のツールを使用することが望ましい。現在、日本国内でも有償、無償それぞれの GSN 記述ツールが使用可能である。

(2) GSN 記述の注意点

GSN はノードの内部に説明を書きこむ。この部分は自然言語によって記述されるため非常に表現力が高い。そのため、一つのノードに多くの情報を詰め込んでしまうという問題が起きやすくなる。一つのノードに多くの情報を詰め込んでしまうと何が何に対してサポートしているのか、何が仮定されているのかということが曖昧になり、議論の構造化がなされなくなってしまう。各ノードにはできる限り一つの情報のみ与えそれぞれの関係が曖昧にならないよう注意が必要である。

5. 取り組みの実施

まず初めに IEC 62278 の GSN による記述を行った。IEC 62278 は鉄道分野の機能安全規格の最上位の規格であり、そのライフサイクルは第 1 段階:構想から始まり第 14 段階:廃棄までの 14 段階で構成されている(図 B-2-4)。

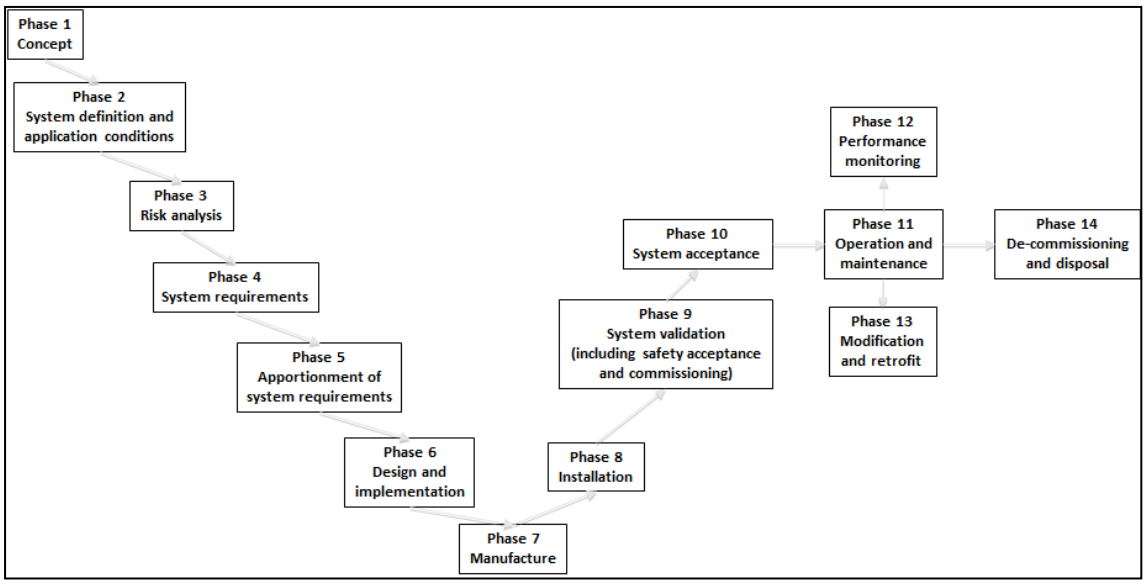


図 B-2-4　IEC 62278 のライフサイクル

また、各段階は

- 目的 ： 段階で達成するべき目的が記述されている
- インプット ： 目的達成のために必要なインプットがなんであるかあげられてい

る

- 要求事項 ： 目的達成のためのタスクや成果物に記載しなくてはならない内容など様々な必要事項が記述されている
- 成果物 ： 実施した結果をどのような成果物としてまとめ、提出するかが述べられている
- 検証 ： 前段階と現段階、または現段階内の成果物の整合性や完全性など段階で検証しなくてはならない事項を挙げている

という項目から成り立っている。

IEC 62278 全体の GSN を記述することは可能であるが、可視性や段階間の関連が見えなくなる可能性があるということから、段階ごとに GSN を作成することとした。本件では特に安全性にとって重要である第 3 段階:リスク分析に対象を絞り、さらにリスク分析とリスクの管理プロセス部分を GSN 化している。

IEC 62278 の第 3 段階:リスク分析はその名前が示す通り、当該システムのハザードを特定しそのリスクを決定する。さらに、リスク分析はこれ以後のライフサイクルの各段階で繰り返し実施されるため、リスクを継続的に管理するプロセスを確立する必要性もある。実際に第 3 段階:リスク分析の目的としては

(1)　当該システムに関わるハザードを特定する
(2)　ハザードの発生につながる事象を特定する
(3)　ハザードに付帯するリスクを明らかにする
(4)　リスクを継続的に管理するプロセスを確立する

となっている。また、リスク分析の結果をまとめたハザードログがここでの成果物となっている(ただし、成果物として名前は与えられていないが過程で作成された分析資料も明確に必要と記述されている)。

GSN は基本的に規格に記述された内容に沿って作成している。ここでは作成した GSN の概略を説明する。GSN のアーキテクチャは図 B-2-5、全体像は図 B-2-6 のとおりである。

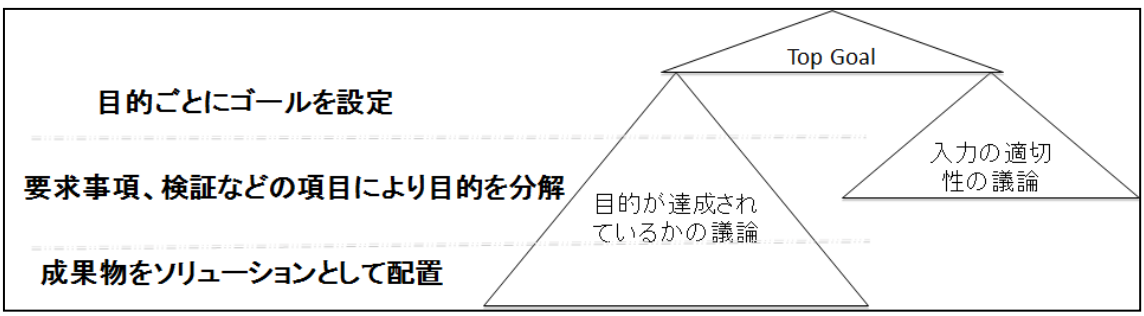


図 B-2-5　GSN のアーキテクチャ

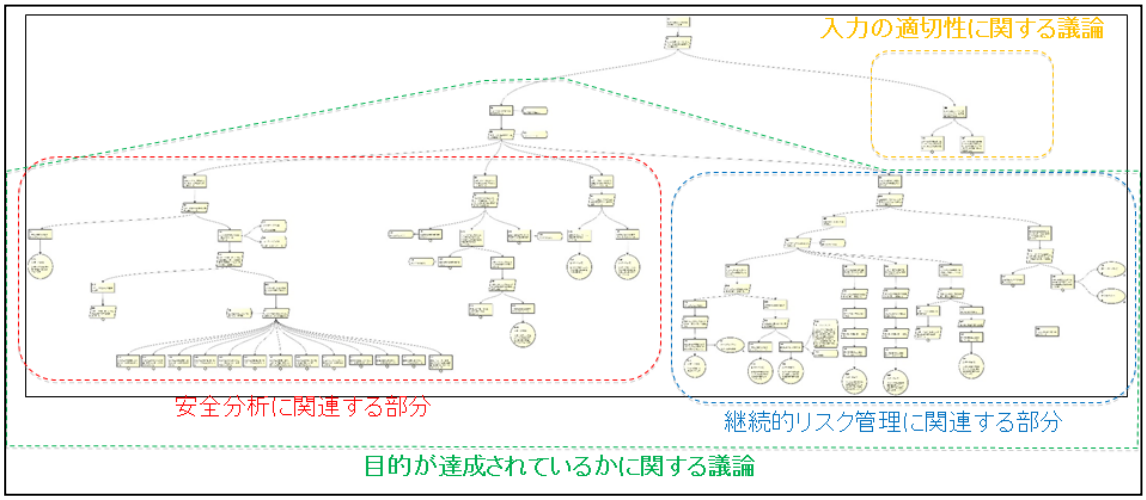


図 B-2-6　GSN-全体像

トップゴールは「リスク分析フェイズが十分実施されている」とし、それをこの段階のインプットが適切であること、目的が達成できていることに分けて議論を展開した(図 B-2-7)。

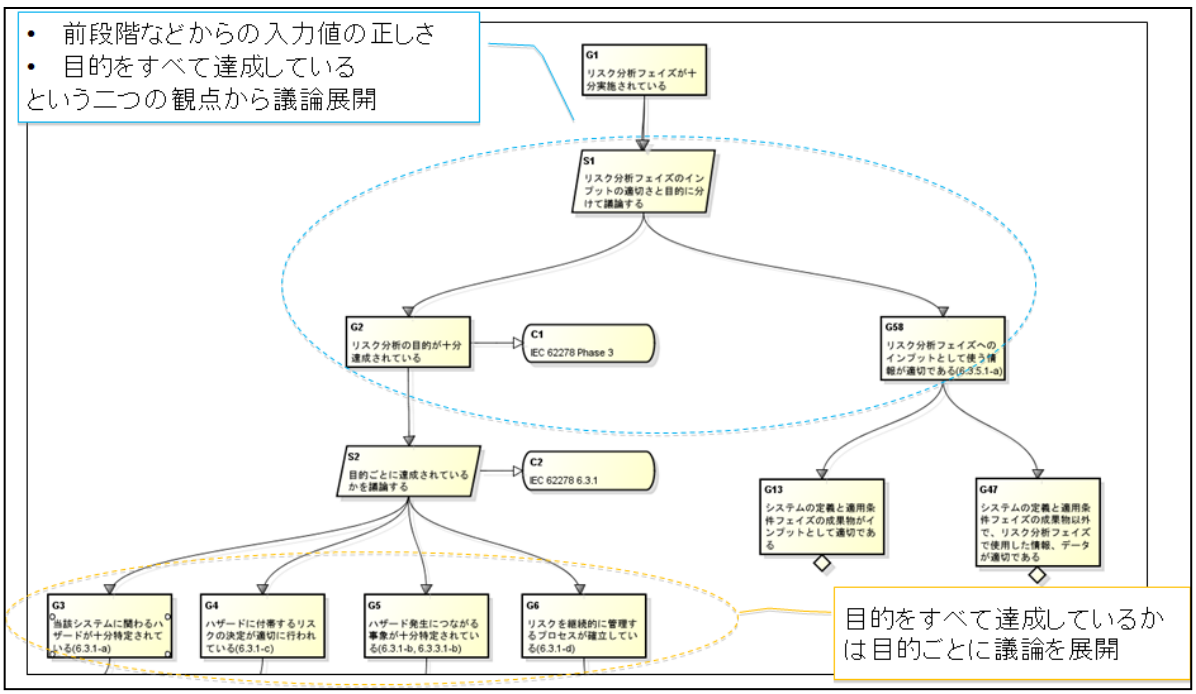


図 B-2-7　GSN-上部構造

さらに、目的が達成されているかは第 3 段階:リスク分析に記述されている 4 つの目的ごとに達成されているかを議論した。

各目的が達成されているかは要求事項や検証に記載されている項目によりサポートされている。例えば、目的 6.3.1-a を達成されているかというゴール(図 B-2-8 中 G3)は検証 6.3.5.1-c に従い議論が展開され、要求事項 6.3.3.3-g(図 B-2-8 中 G7) と要求事項 6.3.3.1-a(図 B-2-8 中 G8) によりサポートされている。これらの議論を支えるソリューションは成果物が配置される。この際、ソリューションは成果物全体となる場合もあるが、成果物のある部分が対象となる場合もある(図 B-2-8 中 Sn10)。また、成果物はコンテキストにも表れることがある。この場合も成果物全体を示す場合と成果物のある部分を示すことがある(図 B-2-8 中 C4, C5)。

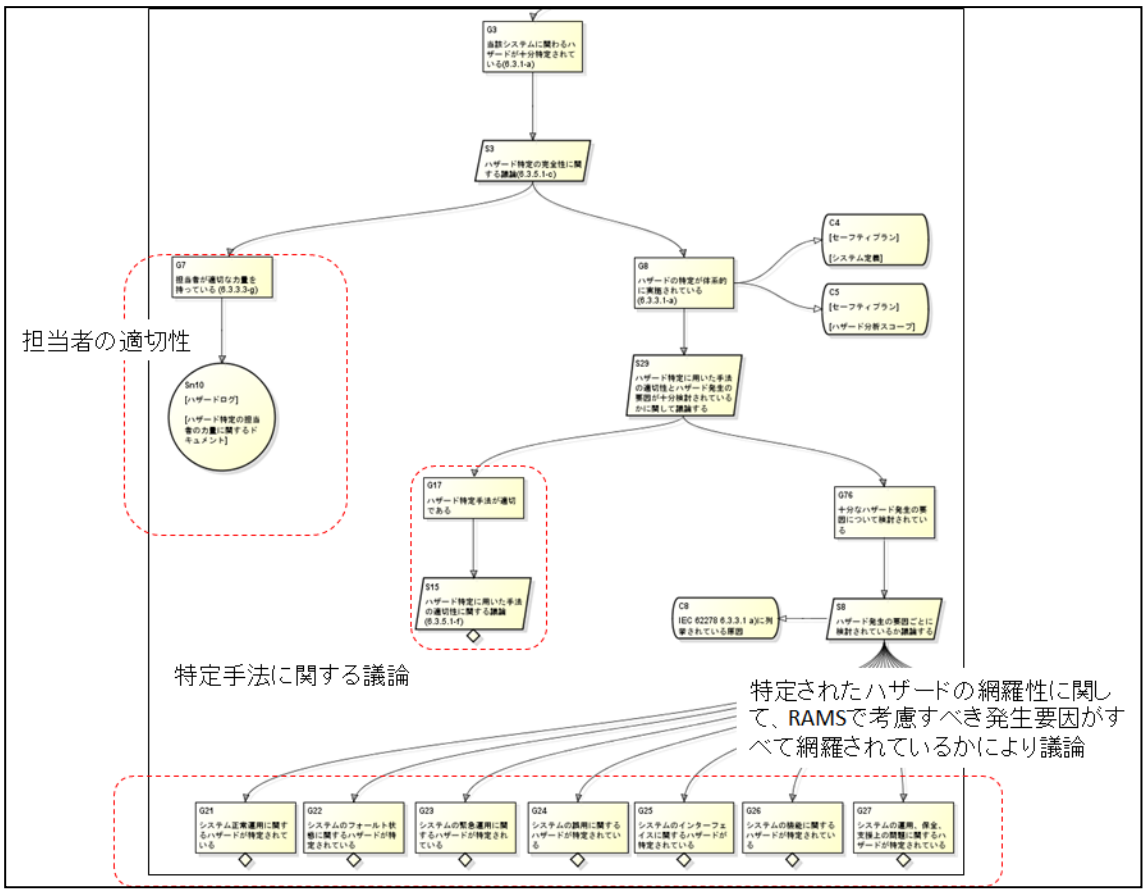


図 B-2-8　GSN-ハザード特定の十分性

最後に作成した GSN を基にそこで使用されている成果物(あるいはその部分)を示しているソリューション、コンテキストに対し、それらにパラメータし RAMS の GSN テンプレートを作成した。成果物とその部分を示す部分があるため、ソリューション、テンプレートにドキュメント名のパラメータとその詳細な部分を示すパラメータを付け加えた。示すものが成果物全体の場合には二つ目の部分であるパラメータを空にすることで表現している(図 B-2-9)。

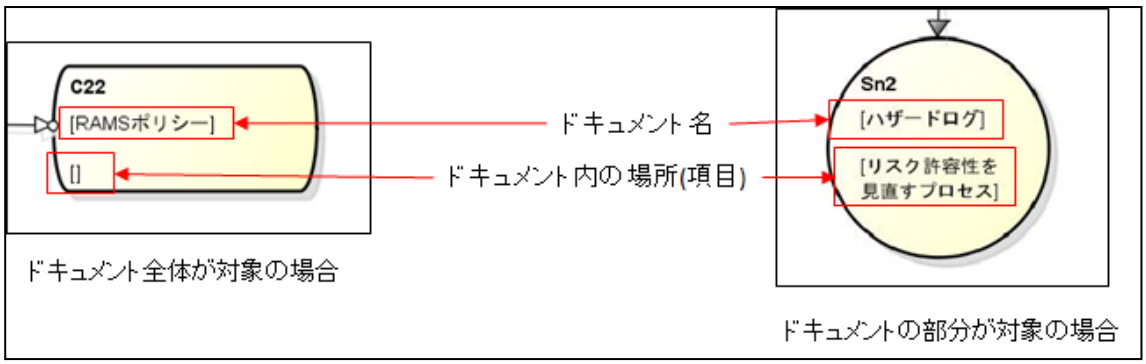


図 B-2-9　GSN のテンプレート化

セーフティケースのテンプレートは IEC 62425 を基に作成した。IEC 62425 はセーフティケースの章立てから始まり、そこに記述すべき内容を詳細に記載している。ただ、記述すべき内容が IEC 62278 のどの段階のどの成果物の内容に当たるかの記載はない。そこで、IEC 62278 と IEC 62425 を比較し、どの成果物(もしくはその部分)がどのセーフティケースの記載内容にあたるか検討を行った。すでに述べたように二つの規格は互いに参照をしている規格であるため用語もほぼ同じものを使用しており、比較はある程度容易に実施できた。一部は IEC 62425 において一般的な記述がされており、IEC 62278 の内容から補足を行った部分もある。

成果物(またはその部分)は[ドキュメント名 ： 記載箇所]という形で記述を行いテンプレートとした。これにより IEC 62278 で作成された成果物の名前と記載箇所をパラメータとして与えることでセーフティケースが完成するのである(図 B-2-10)。

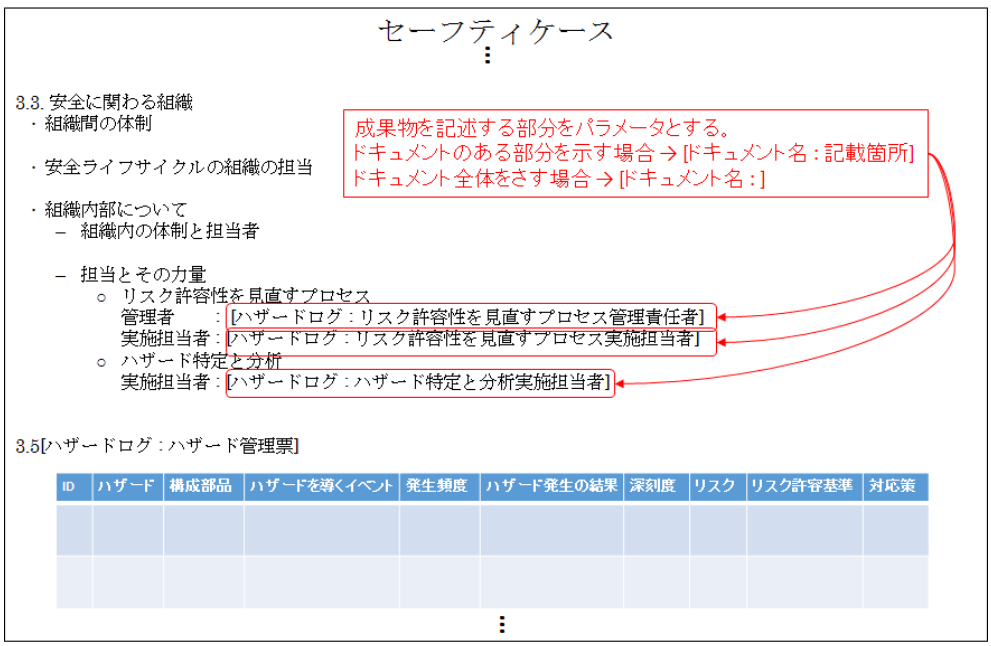


図 B-2-10　セーフティケーステンプレート例

GSN とセーフティケースそれぞれのテンプレートを作成する際に用いたパラメータ(ドキュメント名と記載箇所)の名前は統一したものを使用した。これは同じパラメータ名を用いることにより、GSN とセーフティケースを連携させるためである（図 B-2-11)。

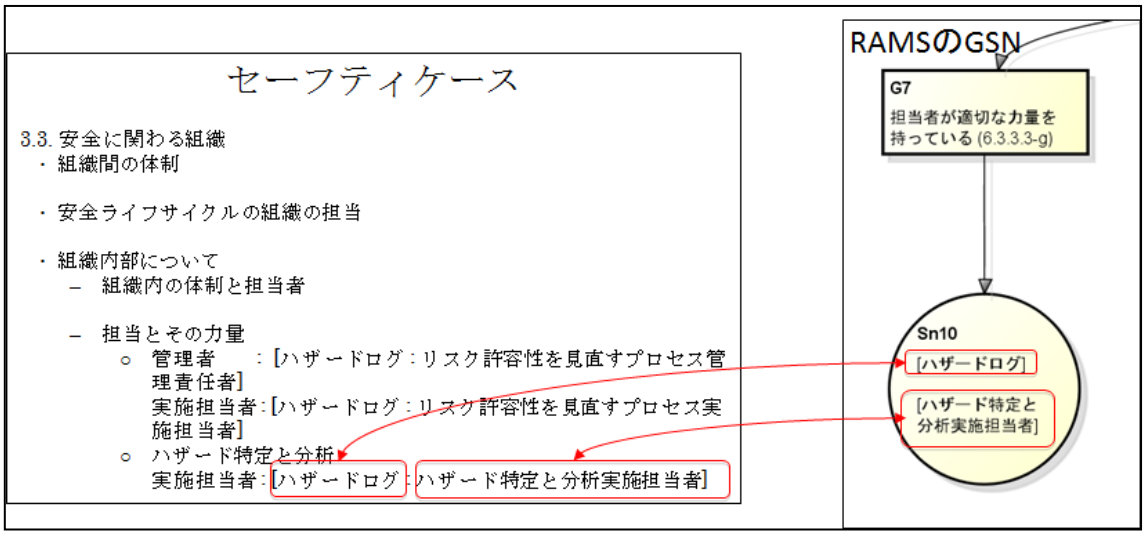


図 B-2-11　GSN とセーフティケースの連携例

6. 達成度の評価

本事例の目的は以下の二つである

（1）　アセスメントのための文書(セーフティケース)作成支援

（2）　規格適合のアセスメントの支援

開発者は IEC 62278 に従い開発を実施する。そこで作成された成果物は IEC 62278 の GSN テンプレートに配置していく。これにより開発工程が IEC 62278 に従い実施されたことを GSN により保証することができる。さらに、この GSN はセーフティケースと連携しているため、GSN が作成されることによりセーフティケースの対応する部分に成果物が登録されることになる。これにより、開発成果物が IEC 62425 のどの部分に対応するか検討する必要なくセーフティケースの作成が行える。また、セーフティケースに抜け漏れがある場合は GSN のソリューションにも抜け漏れが生じているため、容易に気づくことが可能となる(図 B-2-12)。

一方で、セーフティケースの内容がどのような根拠で記述されているか、IEC 62278 のどの要求事項を満たしているのかということはセーフティケースと GSN の関連を見ることにより明確に理解が可能である。これによりアセスメント時にその根拠を IEC 62278 と比較検討するという作業を軽減できる(図 B-2-12)。

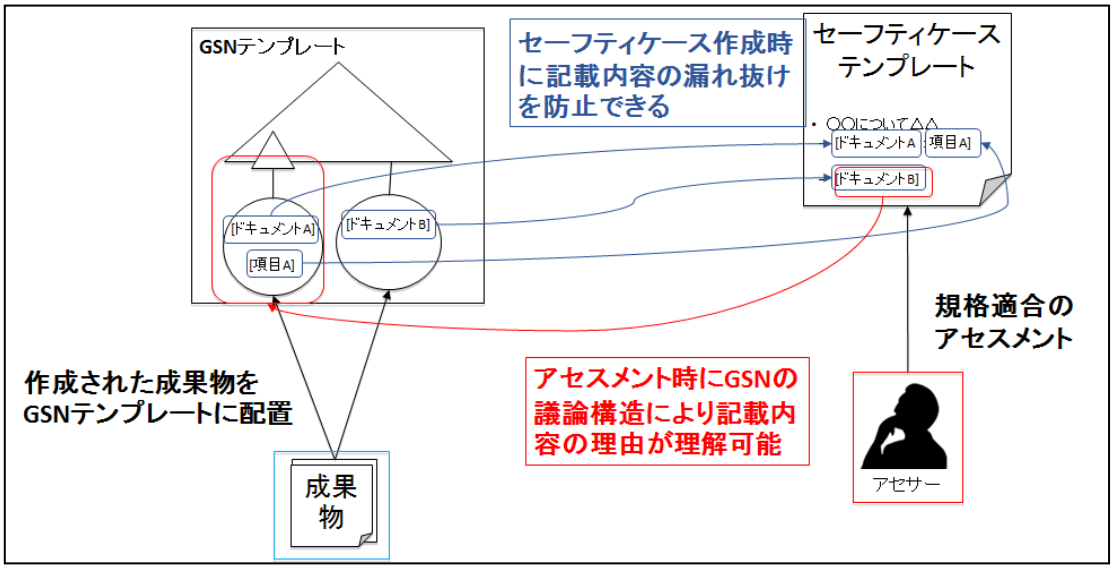


図 B-2-12　セーフティケース作成とアセスメント支援

また、GSN により IEC 62278 を記述することで理解が難しい規格の内容が整理されるという効果もあった。すでに述べたように IEC 62278 は各段階で目的、インプット、要求事項、成果物、検証という項目から成り立っている。しかし、これらの項目に記載されている内容の関係は必ずしも明確にされているとは限らない。実際、目的達成のためにどの要求事項が必要であるか、そのインプットはなんであるか、要求事項の結果はどの成果物に記載されるかといった疑問点は多く存在する。GSN で記述することでこのような項目間の関係が明確になった。図 B-2-13 は、目的と要求事項、検証の関係が明確になった例と、要求事項への入力が明確になった例である。

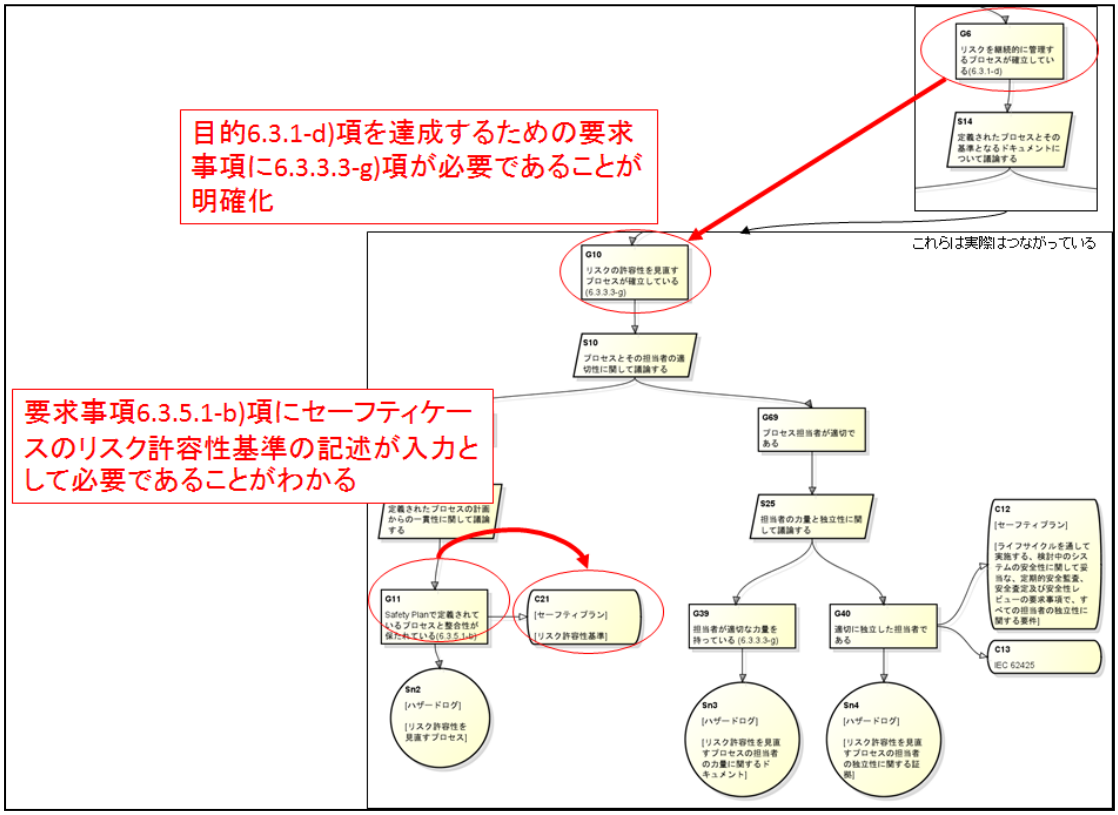


図 B-2-13　記述項目の関係の明確化

7. 今後の取り組みと考察

本事例の取り組みは IEC 62278、IEC 62425 の一部に対して行ったにすぎず、これを両規格全体に対して実施する必要がある。実際の大規模なシステムに対して適用しその効果をはかる必要がある。また、ほかの鉄道規格との連携についても今後実施していく必要があると考えられる。

この取り組みを推し進め「GSN を用いたトレーサビリティ確保の支援」ということも考えている。

トレーサビリティは開発で非常に重要であり、様々な機能安全規格でもトレーサビリティの確保は重要な項目の一つとして挙げられている。昨今、話題として取り上げられること多くなり様々なツールも作成されている。一方で、トレーサビリティの確保は人手によって行われており、どの項目間に関連がありトレースをつけるかは、その作成者でなくてはわからないことが多い。そのため、実施者が限られコストもかかる作業となっている。また、確保したトレースの根拠は作成者しか明確にできないため、トレースの抜け漏れがあったとしてもそれを他の人が発見することは難しい。このような問題点を解決するため GSN を用いてトレースの根拠を示し、トレーサビリティ確保の支援を行うことが課題の一つである。

B－3

非機能要求グレードの大学ポータルサービスへの適用 [1]

1. 概要

名古屋大学では、情報環境マスタープラン2010[1]に従って、情報環境整備のための評価指標の策定を進めている。 本編では、IPA が提案している非機能要求グレード活用シート[2]を用いて、大学ポータルサービスの信頼性指標を定義するとともに、D-Case[3]を用いて信頼性評価指標を確認する方法について検討を実施した結果について報告する。

2. 取組みの目的

2.1. 解決すべき課題

同大学では、情報環境マスタープラン2010 に従って、情報環境整備のための評価指標の策定を進めている。本事例では、IPA が提案している非機能要求グレード活用シートを用いて、大学ポータルサービスの信頼性指標を定義する方法ならびにD-Caseを用いた信頼性評価指標の充足性を保証する方法の具体化と、その適用性の評価が課題である。

同大学の情報環境設備における理念と基本情報・実施計画を全学で策定し、 共有するために名古屋大学情報環境マスタープランが策定された。マスタープランの概念を実現するため、情報環境の達成目標として、 以下に示すような利便性、 安全性および信頼性の3つの観点から、 情報環境の評価指標を設定し、 総合的に評価し、 調和のとれた情報環境の構築を目指すことになっている。

（1） 利便性

開放的かつ機能的な情報環境を構築する観点から以下の評価項目を設定する。

a. ユーザ満足度（ユーザインタフェース、設備の充実度など）
b. 記録する個人活動履歴の必要十分性
c. 大規模計算機環境の維持および速度向上
d. 情報ネットワークの開放性、 運用安定性、 到達性、 容量

（2） 安全性

コンプライアンス（法令）、 セキュリティ（安全保障、 設備、 防衛）、 プライバシー（個人情報保護）のそれぞれの視点から、 利用者にとって安心・安全な情報環境の実現を目指した評価項目を設定する。 システム設計およびシステム運用に対する安全・運用基準を定め、シ

[1] 事例提供： 名古屋大学情報連携統括本部情報戦略室 山本 修一郎 氏

ステム導入者、 情報基盤運用者および、 情報サービス提供者に遵守させる一方、 情報環境に関する倫理教育を行う。他者を攻撃せず、 自己を守るという両面から安心安全をとらえる。むやみに完璧な解を求めず、 他の水準とのバランスをとりながら合理的な解の達成を評価する。

（3） 信頼性

情報環境の継続的な維持のために考慮すべき事項を評価項目とする。情報システム運用時の信頼性概念に従って可用率などを定める。信頼性評価項目として、以下のような項目を規定する必要があるとしている。

a. 各情報基盤、各種サービスの可用率、故障率、保守性等
b. ICTシステム運用者の技術・運用スキルレベル
c. 長期的なシステム運用更新計画

このように、名古屋大学情報環境マスタープランでは、利便性、安全性、信頼性という3つの観点で規定すべき評価指標を例示しているものの、具体的な評価指標の定義については明確になっていないという問題があった。このため、信頼性の観点から非機能要求グレードの活用シートの項目に従って信頼性評価指標を策定することとした。この理由は、信頼性が非機能要求のひとつであること、非機能要求グレードでは、情報システムの非機能要求が我が国の情報システム分野の知見を体系的に整理できていること、したがって、新たに同大学情報環境のためだけに同様の信頼性評価指標を策定するよりも効率的であることである。

一方、懸念事項としては、非機能要求グレード活用シートの評価項目数が多いことから、活用シートを用いた確認作業に時間がかかることが想定された。この課題に対しては、評価対象システムを限定すること、活用使途による信頼性指標の確認作業を進める過程で問題が発生したら直ちに作業を止めて対策をとるという方針を策定することにより、非機能要求グレード活用シートを用いた信頼性評価指標の策定活動を実施することとした。

2.2. 目標

以下で述べる提案手法によって情報環境の信頼性評価を効率的に実施できること

3. 取組みの対象、適用技術・手法、評価・計測

3.1. 対象製品・プロジェクト、適用工程

対象サービス:名古屋大学ポータルサービス。同大学ポータルサイトは図B-3-1のようなWEBサイトである。

図 B-3-1　名古屋大学ポータルサイト

同大学ポータルサイトでは、最初に同大学IDとパスワードを入力することによってログインできるようになっている。同大学IDは、教職員と学生に与えられている。ログインすると、図B-3-1に示すように、同大学ポータルサイトでは、個人ページ、情報連携統括本部からの連絡事項、学生と教職員向けのキャンパス活動支援、主に授業に関する連絡や課題および成績などの管理のための学務支援、防災支援、国際活動支援などのサービスを提供している。大学のあらゆる活動が同大学ポータルサイトを介して支援できるようになっている。

適用工程：同大学ポータルサイトの運用工程における信頼性評価指標の構築と、指標値の評価に基づく信頼性目標の達成度を保証する。

3.2. 方法論

考案した手法では、以下に示すように、①非機能要求グレード、②D-Case、③PDCAサイクルを用いた。ただし、PDCAの適用では、計画から開始するのではなく、現状把握を先行したためチェックから始まるCAPDサイクルとしている。

まず、IPA が提案している非機能要求グレード活用シートを用いて、同大学ポータルサイトの信頼性評価指標を定義する。次いで、D-Caseを用いて、定義した信頼性評価指標が達成されていることを確認する。この結果に基づいて、改善策を立案することによりチェック、改善、計画、実行、からなるCAPDサイクルを反復的に実施する。

以下では、まず非機能要求グレードについて説明する。

非機能要求グレードと活用シート

「非機能要求グレード」は、「非機能要求」についてのユーザと開発者との認識の行き違いや、互いの意図とは異なる理解の状態を防止することを目的とし、重要な項目から段階的に詳細化しながら非機能要求を確認する手法である。非機能要求グレードの効果として、業務への情報システムの影響を明確に把握できること、情報システムの安心・安全な利用や効率的利用ができること、情報システム費用の説明根拠が明確化できることを挙げられている。

非機能要求グレードの内容は、以下のとおりである。

（1） 利用ガイド（解説編）…非機能要求グレードの背景の解説

（2） 利用ガイド（利用編）…非機能要求グレードの利用方法の解説

（3） グレード表…3つの典型モデルシステムとそれに対応する主な非機能要求項目の要求レベル

（4） 項目一覧…非機能要求項目の一覧表

（5） 樹系図…非機能要求項目を6つの大項目ごとに階層的に示した図

（6） 活用シート…プロジェクトに応じてカスタマイズできるように非機能要求グレードの項目一覧をまとめたもの。

このように、非機能要求グレードでは、情報システムのシステム基盤の可用性や拡張性 などの非機能要求項目が活用シートで具体的に明確化されている。

次に、非機能要求グレード活用シートと信頼性評価指標の関係を説明する。

非機能要求グレード活用シートと信頼性評価指標との関係

非機能要求グレードは、対象システムの構築段階で適用することを目的とした非機能要求を発注者と開発者が合意形成を支援するための手法である。したがって、本来は、運用中のシステムの非機能要求を評価することを目的としている訳ではない。しかし、非機能要求グレードでは非機能要求項目を指標値も含めて活用シートとして提供している。

このため、非機能要求グレードの活用事例として、金融系の稼動中システムの再評価作業に活用した事例が紹介されている。この事例では、サービス中のシステムの非機能要求を振り返るため、設計工程で作成したドキュメントを収集し、定義した項目の網羅性・設定したレベルが明確化されていることを確認している。グレード表の重要項目が、ユーザにとっても比較的分かりやすい言葉で定義されているため、システム用語に精通していないユーザ部門の担当者を情報システム部門との認識合わせに利用したとしている。また、利用効果としては、実装時の要求レベルの課題を発見し、将来のリスク回避のための投資やコストダウンの検討の材料とすることができたとしている。この試行事例での非機能要求グレードの利用目的は、ユーザ部門などと社内でコミュニケーションをとりながら、非機能要求に関する暗黙知や非機能要求のレベル感を中心とした評価を行うことと、これらを通じて、稼働中システムのリスク把握とコスト削減を行うことであった。ただし、サービス中のシステムの信頼性評価指標に注目して利用しているわけではない。

本手法では、運用中の情報環境に対する信頼性評価指標として、非機能要求グレードの活用シートを適用することを目的とした。

3.3. 取組みの達成度評価方法

以下の 2 項目によって、取組みの達成度を評価することとした。

（1）　対象サービスの信頼性評価を完了できること

（2）　信頼性評価指標について関係者間で合意できること

4. 取組みの実施、及び実施上の問題、対策・工夫

4.1. プロセス

【準備段階】適用手法の開発

まず、非機能要求グレード活用シートに着目することにより、ポータルサービスの現状を評価する、次いで評価結果に基づいて、対象サービスが信頼性評価指標を満足することを D-Case を用いて確認する手法とした。この理由は、非機能要求グレード活用シートの内容が整理されていて確認し易いことと、適用手法を対象サービスと独立に考案しても、親和性がなければ検討時間を浪費することになるためである。

適用対象サービスが決定している場合には、現状を明らかにした上で将来の目標を明確化することができるので実践的な手法であると考えた。（図 B-3-2）

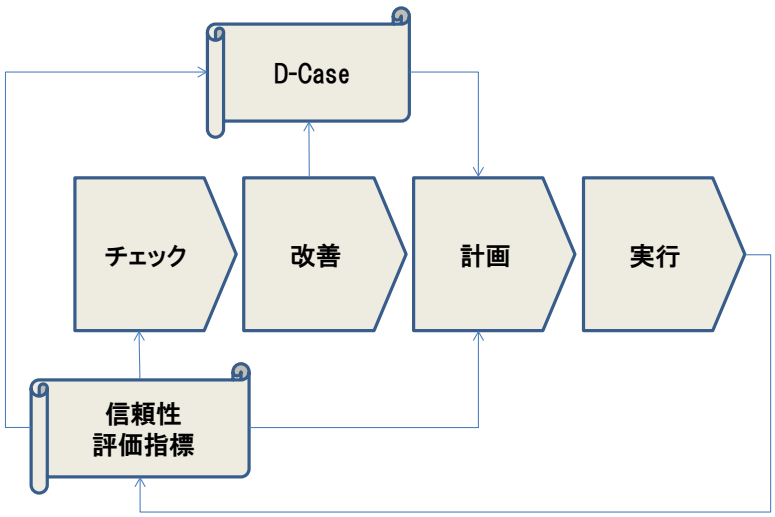


図 B-3-2　D-Case と非機能要求グレードを用いた信頼性評価手法

【実施段階】提案手法の実施

本事例の実施段階では、 非機能要求グレードを用いて同大学ポータルシステムの現状を学部学生が 1 名で調査した。

D-Case の作成では、活用シートの階層構成に基づいて D-Case 階層を手順的に作製する方法を採用した。この段階的な手法については、具体例で説明する。

4.2. 具体的取組み内容、活動

準備段階では、非機能要求グレードと D-Case に基づいて、同大学の情報環境の実情をよく知る有識者が単独で適用手法を開発した。

非機能要求グレードを用いた信頼性評価指標について、担当者との面談によるヒヤリングを、1 週間に1 回1.5 時間程度行い、 合計4 回行った。実施期間は1ヶ月程度であった。

以後、 同大学ポータルシステムを本システムと呼ぶことにする。今回、同大学の情報基盤センターの本システム担当者と現時点での評価指標を調べた。

実験の進め方としては、 非機能要求グレード表の項目をひとつずつ調査していく形で行った。 まずメトリクスを理解した上で、 本システムの具体的な値を調べ、 当てはまるレベルにチェックを入れていった。 本システムは大学構成員だけが利用するので『社会的影響がほとんどないシステム』に分類されていることから、 重要項目で与えられるベース値と比べて大きく異なっていないかについても確認していった。

ここで、 本事例に関連する組織を図B-3-3に示す。 同大学の情報基盤センターは学内の教育活動のための情報基盤として同大学キャンパス情報ネットワークの構築、 運用を行うとともに、 対外的には学術情報ネットワークの東海地区の中核拠点としての役割を担う施設である。 本システムは同大学キャンパス情報ネットワークの一部である。

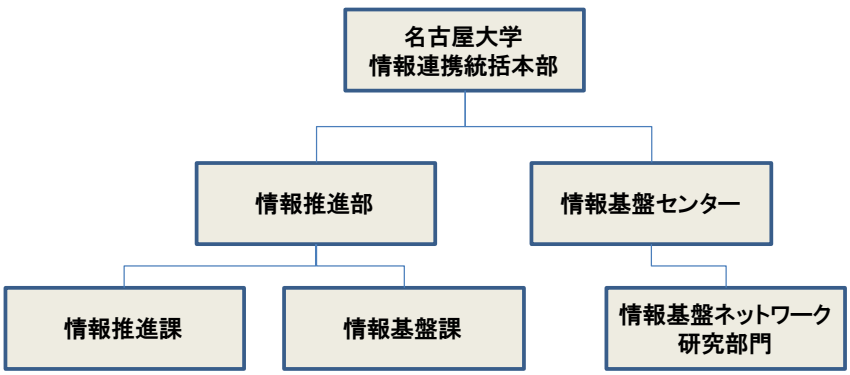


図 B-3-3　対象システムの関係組織

同大学情報連携統括本部の情報推進部には、情報推進課と情報基盤課が設置されている。両課が協力して全学的な情報化の支援業務を担当している。情報基盤課では、同大学のネットワーク、データベース、全国共同利用、計算機システム、マルチメディア教育、e ラーニング、情報管理、ヘルプデスク等の実施、管理・運用及び支援を技術職員が担当している。

情報環境の信頼性評価指標については、情報連携統括本部の情報基盤課の情報系職員が設定・評価することにより、継続的に維持可能な情報環境を構築・整備する。信頼性評価指標

の評価では、非機能要求グレードの活用シートに基づいて作成したエクセルシートを用いて指標値をチェックすることとした。

4.3. 生産物、途中成果物の変更、改善等

実施結果として作成された、評価指標の構成は表B-3-1のようになった。

表 B-3-1　信頼性評価指標

信頼性特性	特性項目	指標数
可用性(8)	運用時間（通常）、 業務継続性、目標復旧水準（通常）、目標復旧水準（大規模災害時）、稼働率、耐障害性、災害対策、回復性	２４
性能・拡張性(7)	通常時の業務量、業務量増大度、保管期間、性能目標値オンライン、性能目標値バッチ、オンラインスループット、バッチスループット	２６
運用・保守性(11)	計画停止、運用負荷削減、運用保守、復旧作業、異常検知対応、運用時間、バックアップ、運用監視、交換用部材の確保、運用環境、運用管理方針	４８
移行性(4)	スケジュール、データ、リハーサル、移行トラブル	１３
セキュリティ(10)	コンプライアンス、セキュリティリスク分析、セキュリティ診断、セキュリティリスク管理、アクセス・利用制限、データの秘匿、不正監視、ネットワーク対策、マルウェア対策、Web 対策	３４
環境・エコロジー(3)	制約条件、システム特性、環境マネージメント	１７
合計	43 項目	１６２

実際の非機能要求グレードの例として、 可用性の項目の一部を表B-3-2に示した。

表 B-3-2　信頼性評価指標の例（部分）

NFR	項目	メトリクス （指標）	指標値（現状）
可用性	運用時間（通常）	運用時間（通常）	24 時間無停止
		計画停止の有無	計画停止有り（運用スケジュールの変更可）
	業務継続性	サービス切替時間	24 時間以上 － コンパイルし直す→１H 比較的小さなこと →10m 程度
		業務継続の要求度	障害時の業務停止を許容する
	目標復旧水準（通常）	RPO（目標復旧地点）	障害発生時点 （日次バックアップ+アーカイブからの復旧）
		RTO（目標復旧時間）	6 時間以内 － ストレージサーバ以外　ストレージサーバは１営業日以上 2、3 日かかるかもしれない
		RLO（目標復旧レベル）	全ての業務
	目標復旧水準（大規模災害時）	システム再開目標	数ヶ月以内に再開 － あまり定まっていないので推測 学内共通基盤によるところが多い
	稼働率	稼働率	99%（1 年間で数時間程度の停止を許容）

次に、非機能要求の定量評価指標をD-Caseを用いて保証した例を図B-3-4に示す。

D-Caseは、システムやサービスの安全性や信頼性を保証するための図式記法である。D-Caseでは、保証すべき内容を主張として矩形で記述する。主張が成立する証拠を円で記述する。上位の主張を立証する証拠がないときに、下位の主張に分解して段階的に保証するために、平行四辺形で、これらの主張間の関係を記述する。

図B-3-4では、「ポータルサービスが可用性を達成している」という主張を、非機能要求グレードで定義されている可用性についての下位特性に分解して説明している。たとえば、可用性の下位特性として「事業継続性が達成されている」や「リカバリ特性が達成されている」と

いう下位特性があることが分かる。さらに、「事業継続性が達成されている」という下位特性が保証するために、非機能要求グレードの指標について分解して説明している。
すなわち、「サービス切り替え時間は24時間以内である」と「サービス中断が許容できる」という2個の指標について、それぞれ証拠が示されている。

　このようにして、D-Caseを用いることで、サービスの定量的な評価指標が達成されていることを保証できるだけでなく、保証結果を分かりやすく体系的に文書化できた。

図 B-3-4　D-Case による信頼性評価指標の保証例

4.4. 途中での主なトピックス

（1）　部門や担当者によるサブシステムの分割について

　本システムは同大学情報基盤センターで構築されるが、情報基盤センター内でも部門に分かれるものが存在する。 このような項目については、 部門からなるサブシステムが独立するよう分割し、 ラベリングしておくことにより評価がしやすくなると考えた。具体的には、情報基盤センターでは情報基盤ネットワーク研究部門があり、ネットワーク関連の業務はこの部門で管理されている。 このことからネットワークに関連する項目を抽出することによってモジュール化を行い、 管理が容易になることが期待される。 加えて、 情報収集の面でも容易である。調査を行って、 分割すべき部分は以下のようである。

_ ネットワーク
_ データベース
_ サーバ設備などの物理的なこと
_ 災害対策など大学として取り組んでいること

(2)　ユーザ、ベンダが異なる項目について

同じ項目内でも、 サブシステムによってメトリクスに含まれる用語が異なる場合がある。 その例がユーザとベンダである。 移行性や運用・保守などの項目で見られる。 これらの項目にあるユーザとベンダとは、 システム側からすると同大学情報基盤センターがユーザかつベンダであり、 ハード側からすると同大学情報基盤センターがユーザでありハードを提供している企業がベンダとなる。 このような場合は、ハードとシステムを分割することによって混乱を防ぐことができる。

具体的な項目を述べると、 移行計画の移行分担作業の項目について、 指標評価の結果、 移行を行うのはベンダである。 しかし、この場合のベンダはシステム側かハード側かによって同じ用語でも異なり、 システム側においては情報基盤センターがベンダとして対応し、 ハード側では保守契約を交わした企業がベンダを担当する。この場合、どちらも同じレベルであるが分かりやすさのため分割すべきと思われる。同じレベルで無い場合はもちろん分割すべきである。 その際には、 ユーザ/ベンダを具体的に明らかにしておく方が良い。

(3)　指標評価の容易性について

今回の実施にあたって、 容易に評価を行うことができた項目とそうでない項目に分類した。 評価が容易に行えたものについても本システム担当者が評価を行うことができた項目と、データベース、 サーバなどその他担当者が評価を行った項目で分類すると、162項目中の150項目が容易に分類できた。他の担当者によって容易に分類できた項目が6項目、その他の項目が6項目であった。

容易に評価できたと判断する基準は、 その項目の主な担当者がメトリクスを理解した後に本システムとしてのなんらかの基準がすぐレスポンスがされること、 何を参照すれば評価基準が分かるか理解していることを定義とする。 これにより、 システム担当者が理解している部分と各担当者が理解している部分、グレードは定まっているがあまり考慮されていないと思われるであろう部分と見ることができる。 容易性が低い場合、 関係者の共通認識が低く、 重要項目であった場合は深刻なエラーが発生する原因にもなりやすいと考えることもできる。 重要項目でなくとも、 システムによっては重要視すべきである項目も存在する可能性もある。 このように指標評価の容易性を踏まえて考察することで非機能要求のディペンダビリティを評価する際にも役に立つと考えている。

(4)　D-Case による信頼性評価指標の確認について

図 B-3-3 から分かるように、信頼性評価指標の定義票の列を D-Case の階層に対応付けることで、D-Case による同大学ポータルの信頼性を客観的かつ容易に確認できるようになった。ここで提示した方法は、D-Case の適用パターンになっている。このように、確認対象に適した D=Case パターンを用意しておくことで、D-Case の現場導入が容易化できることが判明した。

4.5. 人材育成、意識改革等

本事例で紹介したように、適切かつ簡潔な手法を整理しておくことにより、現場での調査とそれに基づくシステム信頼性の確認が容易化できる。このように、適切な手法を活用することで、無理なく人材育成ができるだけでなく、対象システムに対する高信頼化への動機づけを促進できたと考える。実際に、本システムの運用担当者からは、「現状が把握できたことがよかった。この結果を踏まえて今後、改善に向けて検討することを計画したい。」との意見が寄せられた。

5. 達成度の評価、取組みの結果

非機能要求グレードを大学のポータルサイトに適用することによって高信頼性指標を詳細に定義できた。

成果物として大学ポータルサイトの信頼性評価表ならびに、D-Caseによって信頼性評価指標を確認した。これにより、現行サービスの状況を評価できたので、今後の改善策を評価指標に基づいて、目標値を定義する予定である。このように、評価指標を目標値として、定期的に達成度を評価することにより、評価データに基づくPDCAサイクルを実施する予定である。ここで、アクションプランは、以下のようになる。評価指標に対する目標値を定義することが計画(Plan)である。運用時に目標値の達成を監視することが実行(Do)である。対象期間における目標の達成度を確認することにより、達成できた要因と達成できていない要因を分析することが検証(Check)である。検証結果に基づいて、必要な対策を立案することが改善（Action）である。 さらに、改善対策によって達成すべき評価指標の目標値を定義することによって、新たな計画を策定する。

今回の評価では、現行サービスに対する評価指標が明確に定義できていなかったため、上述した PDCA サイクルの検証を実施した。したがって、チェック、改善、計画、実行（CAPD）サイクルと考えることができる。

多くのサービスでは、サービス開発時点と現行サービスでは、継続的なシステム更改のために運用内容が当初の計画と必ずしも一致していないことや、そもそも運用目標が明確になっていない場合がある。このような場合には、本事例で紹介したように、まず標準的な指標を用いて検証することから必要な改善策を明らかにして計画を実行するという CAPD サイクルの方が現実的である。

6. 今後の取組みと考察

今後の課題として、今回策定した信頼性評価指標に基づいて本システムの改善サイクルを実施することが挙げられている。また、同大学の他のサービスについても非機能要求グレード表とD-Caseを用いた信頼性評価指標の調査と確認を実施する予定である。

参考文献

[1] 名古屋大学情報環境マスタープラン 2010、
http://www.icts.nagoya-u.ac.jp/masterplan.html
[2] エンタプライズ系事業/非機能要求グレード、
http://www.ipa.go.jp/sec/softwareengineering/std/ent03-b.html
[3] 松野 裕、 山本 修一郎、 「実践 D-CASE」 978-4-862930-91-0、
ダイテックオンデマンド出版、2013

B－4

冗長構成システム（クラウド等）の耐故障性に対する検証技術 [1]

1. 概要

クラウドを中心とした冗長構成システムで実現される高信頼性システムは、社会インフラの ICT システムとして、なくてはならない存在となっており、トラブルが発生した際の影響度は計り知れない状況となっている。

そのような状況では、トラブル発生に対して即時対応できる運用サービスとする必要があるが、評価・検証手法が十分に確立できていないため、運用に入ってから、重大な障害となるケースがたびたび発生する。

本編では、システムが高度化・複雑化していく中、高信頼性システムを要件定義から運用・維持までのライフサイクルの中、品質を確保して高信頼性を維持していく方法の一つとして、株式会社富士通コンピュータテクノロジーズ／富士通株式会社におけるハードウェアの故障に対するシステムの耐故障性の確認を、検証項目及び確認内容の網羅性を確保しつつ、ハードウェアの疑似故障ツールを用意、効率よく検証する効果的な検証手法の適用事例について紹介する。

2. 取組みの目的

ICT システムは、継続的なサービス提供が求められ、また、一般的なクラウドサービスに代表されるように、不特定で従来では考えられない多数のユーザに対してサービスするため、大規模でかつ高信頼なシステムが要求されている。

ここでは、高信頼性システムへ要求される非機能要求の概要と、設計からテストまでの実現プロセス、そして、検証における課題を述べる。

（1）　高信頼性システムの要求事項

高信頼性システムでは、業務をサービスする際に停止してよい時間（障害保守など）や、障害発生後の対応時間、セキュリティ対策への要求など、「非機能要求」を明確にすることがとても重要である。

これらは、業務に直接関係がないこともあり、一般的に非機能に関する要求は、シ

[1]事例提供：株式会社富士通コンピュータテクノロジーズ／富士通株式会社　表憲一 氏／宮原真次 氏、太田　真司 氏

ステムのユーザ企業においては具体的理解が不十分で、曖昧なまま開発が行われるケースがあった。そのため、システムを構築するベンダーと依頼したユーザ企業にて、あいまいな非機能要求を明確にし、合意形成を支援する目的で、IPA が「非機能要求グレード」を提唱している。

(2)　高信頼性システムを実現するためのプロセス

「非機能要求」は、障害が発生しても業務が継続できるというような「可用性」や障害対応・復旧作業などの「運用・保守性」などであり、システム基盤と呼ばれる、ハードウェア、OS、ミドルウェア製品を組合せて実現している。

また、運用においても、システム運用者によるシステム監視（運用監視サーバ）や運用手順書（トラブルシューティングやエスカレーション等）による対応により、高信頼性システムを維持している。これらの実現イメージを図式化すると、図 B-4-1 のようになる。

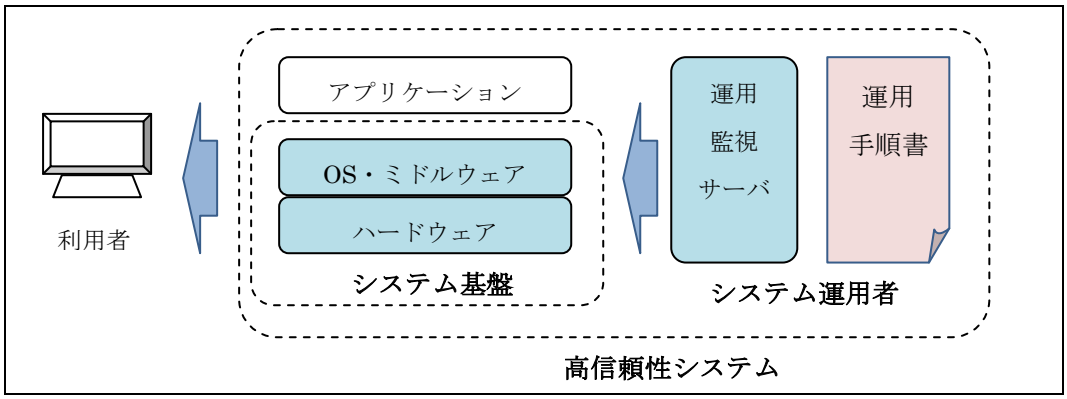


図 B-4-1　高信頼性システムの実現イメージ

このように「非機能要求」が組合せで実現されているため、システム基盤の初期開発（構築）は無論のこと、運用開始後の変更管理（ハード構成変更・増設、OS 等ソフト更新）を含めたライフサイクルにおいて、変更箇所に応じて検証を行い、高信頼性が維持されていることを確認することがとても重要である。

ここでは、システム基盤の初期開発での実現プロセスを図 B-4-2 に示す。

クラウドのように業務継続性が求められるシステムにおいて、システム設計ではサーバ装置を冗長構成にするなどの設計を行い、環境構築ではハードウェアの設置・環境構築等を行い、検証プロセスで、確認・テストを実施していく。

ハードウェアの設置や環境構築は、システム規模が大きく複雑化しているため、確実性が求められる。そのため、ウォータフォール型プロセスで品質を作り込みながら進めている。

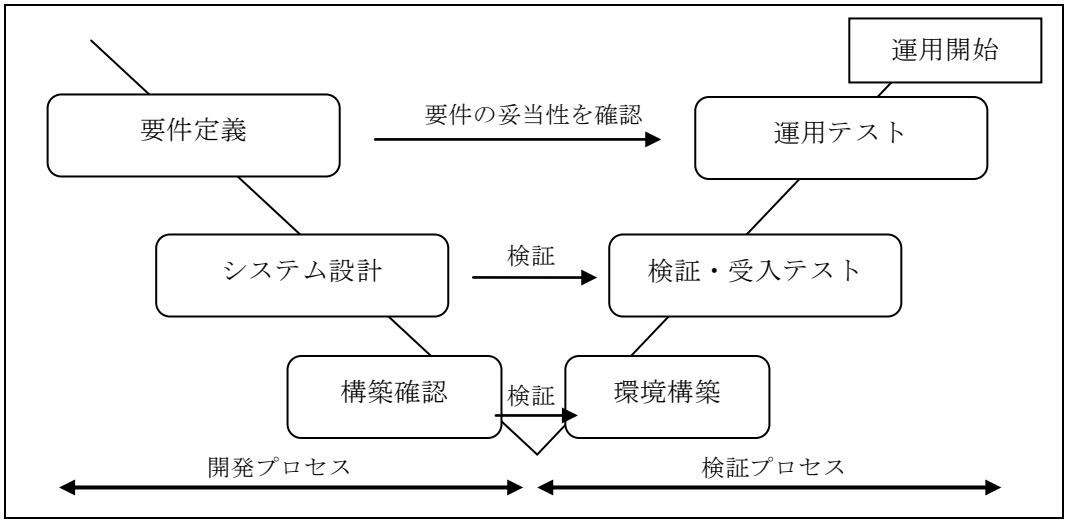


図 B-4-2　システム基盤開発のＶ字モデル

(3)　『非機能要求グレード』を活用した要件定義・システム設計

要件定義プロセスでは、「非機能要求」も明確にしていく必要がある。

一般的に「非機能要求」は、ICT システムのユーザ企業は具体的理解が不十分であるため、IPA が提唱している「非機能要求グレード」を活用し、要件を明確にしていくことが重要である。

「非機能要求グレード」では、まずシステムのグレード要求に応じて、表 B-4-1 に示す 3 種類のモデルシステムに分類される。

表 B-4-2 に示した 5 種類に大別されている非機能要求項目の指標ごとに、モデルシステムに応じたレベルを選択し、要件を明確にしていくことができる。

表 B-4-1　モデルシステムの定義

No.	モデルシステム
1	社会的影響が殆ど無いシステム
2	社会的影響が限定されるシステム
3	社会的影響が極めて大きいシステム

表 B-4-2　非機能要求の大別

No.	大項目
1	可用性
2	性能・拡張性
3	運用・保守性
4	移行性
5	セキュリティ
6	システム環境・エコロジー

このように「非機能要求グレード」を活用し非機能要求を要件定義プロセスにて整理・合意形成を行い、システム設計のインプットとして重要な役目を果たすものとなっている。

システム基盤にて実現される「非機能要求」は、ハードウェア、OS、ミドルウェア製品や運用・保守を考慮して設計しなければならない。例えば、ネットワーク回線が異常によって切断した場合、○秒以内(顧客影響とならない時間内)に代替ネットワーク回線へ切替え、業務を継続させるためには、表 B-4-3 のようなポイントで設計をしている。

表 B-4-3　非機能要求の設計ポイント

No.	分類	設計ポイント
1	ハードウェア	・ネットワーク回線の二重化（冗長化） ・ネットワーク機器の異常監視
2	OS(ドライバ) ・ミドルウェア	・ネットワーク経路異常時の切替え時間の検討 ・異常監視時間の検討
3	運用	・ネットワーク異常監視方法の選択
4	保守	・復旧手順の検討

(4)　耐故障性の検証に対する課題

非機能要求（要件定義）に沿った設計、構築を行い、検証プロセスにて品質を確保しているが、運用開始後に、ハードウェア故障のようなトラブルを起点に、システム設計ミスや復旧手順書の誤り等により、長時間の業務停止に陥るケースが発生している。

このような問題の発生要因として一般的に考えられることを、各プロセスにおいての原因の作り込みと問題検出漏れの観点で整理すると、表 B-4-4 のようになる。

表 B-4-4　各プロセスでの問題の作り込み要因と問題検出漏れ（一般例）

No.	開発プロセス ／検証プロセス	作り込み要因	検出もれ要因
1	要件定義 ／運用テスト	・復旧手順書の作成不備	・実システムでの確認不足 ・確認手段の不足
2	システム設計 ／検証・受入テスト	・冗長構成・監視箇所の考慮不足 ・異常時の動作仕様が入手困難 ・ハード/OS/MW 設定値の考慮不足	・確認観点の不足 ・確認手段の不足
3	環境構築 ／構築確認	・物理結線ミス ・監視設定ミス ・OS/MW 設定ミス	・人手による確認の限界 ・自動チェックが困難

図 B-4-1 に示したように高信頼性システムは、システム基盤や運用監視、運用手順書などの複数の要素で構成されている。これらは、ハードウェアの故障により、上位である OS・ミドルウェアやアプリに影響が波及し、システム運用者による運用監視や運用手順書による対処が行われるため、ハードウェア故障を起点に検証を行うことにより、すべての要素が確認できることが分かる。

また、表 B-4-4 の№1、№2 の検出もれ要因の対処として、検証項目の観点と、確認内容の観点での網羅性を向上させる必要がある。

a.　検証項目の網羅性

従来、非機能要求自体が明確になっておらず、それに伴い検証項目の網羅性を測るすべがなかった。そのため、入力情報として不十分な仕様書等や経験則の積み上げによって、検証がなされており、網羅性が十分とは言えない状況であった。

IPA が提唱する「非機能要求グレード」によって非機能要求が明確となるため、「非機能要求グレード」から検証項目を抽出するアプローチを試みる。

b.　確認内容の網羅性

耐故障性の確認となると、システム基盤の動作（冗長部分の切替えなど）確認が主であった。しかしながら、高信頼性システムの運用において「非機能要求グレード」から考えた場合、運用・保守面や利用者面を検討する必要があり、これらを考慮した確認を試みる。

3. 取組みの対象、適用技術・手法、評価・計測

今回紹介する事例の対象は、非機能要求グレードで定義されている 3 種類のモデルシステムの内、「社会的影響が限定」と「社会的影響が極めて大きい」の 2 種類である。

（1）　検証項目の抽出

高信頼性システム＝稼働率が高いシステムという考えの下、「非機能要求グレード」から検証で確認すべきポイントを抽出する。

今回は、ハードウェア故障の観点からの抽出となるが、セキュリティ観点やソフトウェアバグの観点を考えると、検討すべきポイントが異なるため、ここでは割愛する。

表 B-4-5 に挙げた非機能要求グレード項目の各指標が目標(設計)通りかを検証し、最終的に、検証結果から稼働率を算出し、要件定義で決めた目標となる稼働率であるかを判定することで、高信頼性システムの確認とする。

表 B-4-5　検証すべき非機能要求グレード項目（例）

No.	非機能要求グレード		指標	検証で考慮すべきポイント
	大項目	中項目		
1	可用性	継続性	稼動率	・RTO 等結果から稼働率の確認
2			サービス切替時間	・サービス復旧時間（運用者） ・業務停止時間（利用者） ・業務継続の要求度（単一/二重故障の観点）
3			業務継続の要求度	・単一/二重故障発生時の影響範囲
4			目標復旧時間(RTO)	・障害ポイント別の RTO
5		耐障害性	冗長化(機器/コンポーネント/経路)	・冗長化部分の動作 ・非冗長部分の故障時影響
6		回復性	復旧作業	・作業者、復旧手順の確認
7	運用・保守	運用監視	監視情報	・監視レベルの妥当性（死活/エラー/性能）
8			監視間隔	・障害検知までの時間
9		運用環境	マニュアル準備レベル	・運用、復旧手順の確認

忘れてはいけないのは、コストやシステム特性により、ハードウェアを非冗長構成とした部分に対しての業務への影響である。

システム設計として考慮して設計したものの、OS やミドルウェア等の動作が不明確（ブラックボックス）な部分も多くあり、実際のハードウェア故障により、思いもよらない部分に影響が出るケースもある。障害事例などノウハウがあり、確実に影響範囲が特定でき、システムとしても許容できるのであれば問題はないが、システムとして高信頼性が求められるのであれば、非冗長構成部分の耐障害性の検証は実施すべきと考える。

また、利用者視点で考えた場合、ハードウェア故障にて冗長構成が縮退した状態で、業務継続性をどこまで担保するか、表 B-4-6 のように性能・拡張性の観点で要件を明確にし、検証する必要がある。

表 B-4-6　検証すべき非機能要求グレード項目（例）

No.	非機能要求グレード		指標	検証で考慮すべきポイント
	大項目	中項目		
1	性能	業務	同時アクセス数	・縮退運転時の業務処理量
2	・拡張性	処理量	バッチ処理件数	
3		性能	通常時レスポンス	・縮退運転時の性能影響
		目標値	順守率	

(2)　検証の流れ

検証項目の抽出後、実際にシステム基盤として検証を実施するためには、個々に検証するのではなく、ハードウェア故障を起因とした一つの流れのなかで確認を実施していく必要がある。

図 B-4-3 に示した検証の流れとなるが、これは、実際の故障時のシステム運用者の動きに近いものであり、検証の中で、運用時の異常時の対応を疑似体験するようなイメージである。

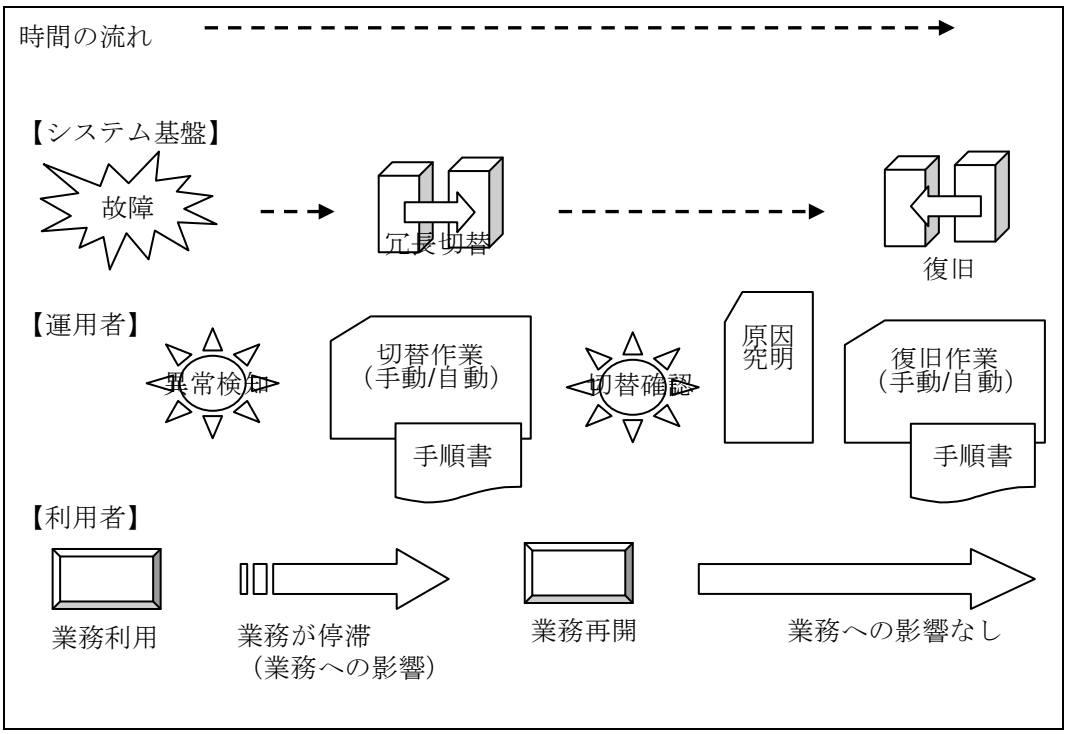


図 B-4-3　検証の流れ

この流れの中で、表 B-4-7 に示した 3 つの視点毎に非機能要求グレードから抽出した検証ポイントをひも付けして、評価を実施するかたちとなる。

表 B-4-7　非機能要求の評価ポイント

No.	視点	評価ポイント（代表的なもの）
1	システム基盤	・冗長化部分の動作 ・単一/二重故障発生時の影響範囲
2	運用者視点	・監視レベルの妥当性 ・障害検知までの時間
3	利用者視点	・業務停止時間 ・縮退運転時の性能影響

(3)　ハードウェア故障の疑似化

検証するためには、起因となるハードウェア故障を発生させる必要があるが、表 B-4-8 のように従来の手法では課題があり、運用前のシステムにおいて耐障害性の検証を実施するには、とても抵抗感があり、実施に及ばない場合が多くあった。

表 B-4-8　従来の手法

No.	手法	課題
1	ケーブル挿抜	LAN 経路異常として、実際に接続しているケーブルを抜いて異常を発生させる。ケーブル操作が伴うため、ハード故障の原因となったり、ケーブルの戻し間違いにより設計とは異なる構成となる場合がある。
2	強制的ハード操作	サーバ異常として強制的にメイン電源を落としたり、HDD を抜いたりして異常を発生させる。保証外の操作のため、ハード故障の原因となる可能性がある。
3	試験器	インタフェース測定器・試験器等を使用し、異常動作を発生させる。ハードウェア装置に試験器を接続するため一時的に構成変更が必要で、運用と異なる接続形態となる。 また、試験器の専門的利用スキルを要求される。

この課題を解決するため、ハードウェア故障をソフトで疑似的に発生させる手法を検討。表 B-4-9 のような要件を定義し、ツール化を実現した。

表 B-4-9　新たな手法への要件

No.	項目	要件概要
1	ハード構成変更なし	ハードの構成変更がなしで実行できる
2	専門スキルの排除	ハードウェア機器（サーバ、ストレージ、ネットワーク）個々の専門スキルなしで、操作が可能
3	疑似的な故障レベル	冗長構成の縮退・切替え機能が動作する疑似的な故障を実現する。

4. 取組みの実施、及び実施上の問題、対策・工夫

実際に検証する際には、検証対象のシステム構成から検証すべき非機能要求グレード項目の確認ポイントをひも付けしていく必要がある。

(1)　ハードウェア故障個所の抽出

システム基盤の検証観点から、システム構成の冗長構成部分の洗い出しを行う。

図 B-4-4 のシステム構成（概略図）をベースに説明する。

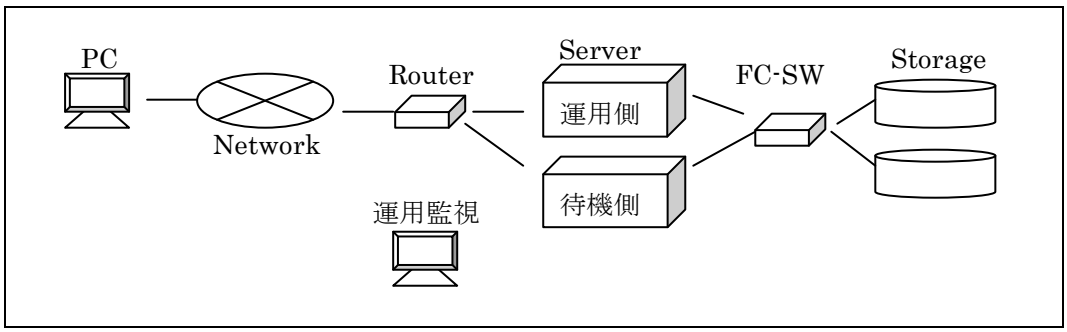


図 B-4-4　事例のシステム構成（概略図）

洗い出しは以下の手順で行い、疑似故障発生ツールでの実現性を確認した。

a.　システム構成上のシングル構成部分、冗長構成部分の抽出

b.　冗長構成の切替え動作の契機（異常状態）の抽出

抽出結果を整理すると表 B-4-10 のようになる。

表 B-4-10　ハードウェア故障の洗い出し

No.	装置	冗長構成タイプ	異常状態
1	Server	運用・待機構成	・サーバ内異常 ・Router 間経路異常 ・FC-SW 間経路異常
2	Storage	ミラーリング構成	・ストレージ異常 ・FC-SW 間異常
3	FC-SW	SW 二重化、 マルチパス構成	・FC-SW 異常 ・Server 間経路異常 ・Storage 間経路異常
4	Router	シングル構成	・Router 異常 ・Network 間異常 ・Server 間異常

(2)　疑似故障発生ツールの準備

これから疑似故障発生ツールでの実現性の確認となる。

図 B-4-5～図 B-4-7 は、故障を契機とした冗長機能の振る舞い例となっている。

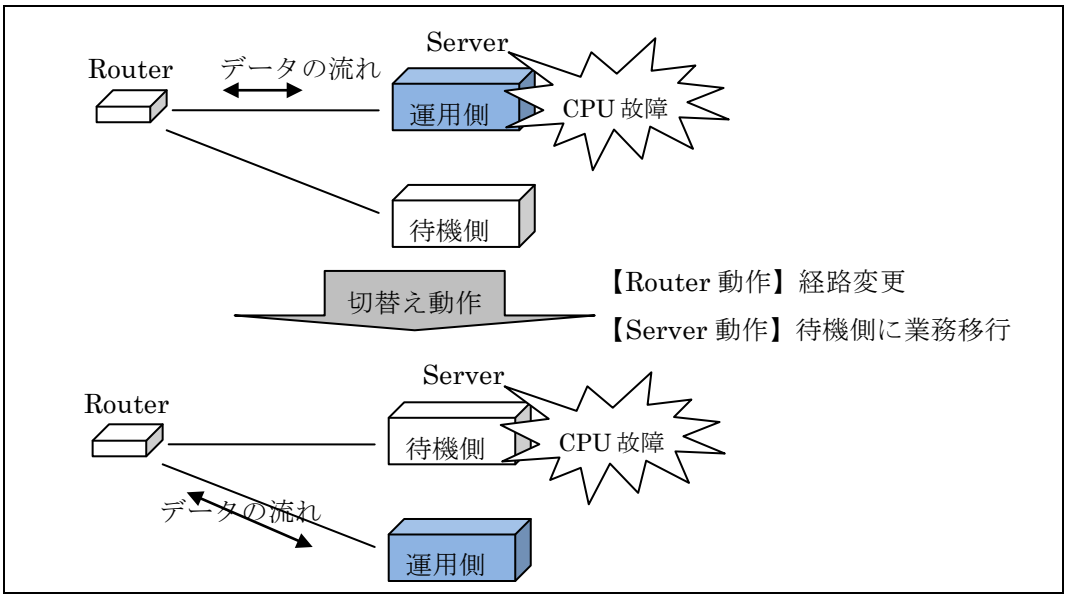


図 B-4-5　サーバ切替動作の確認例

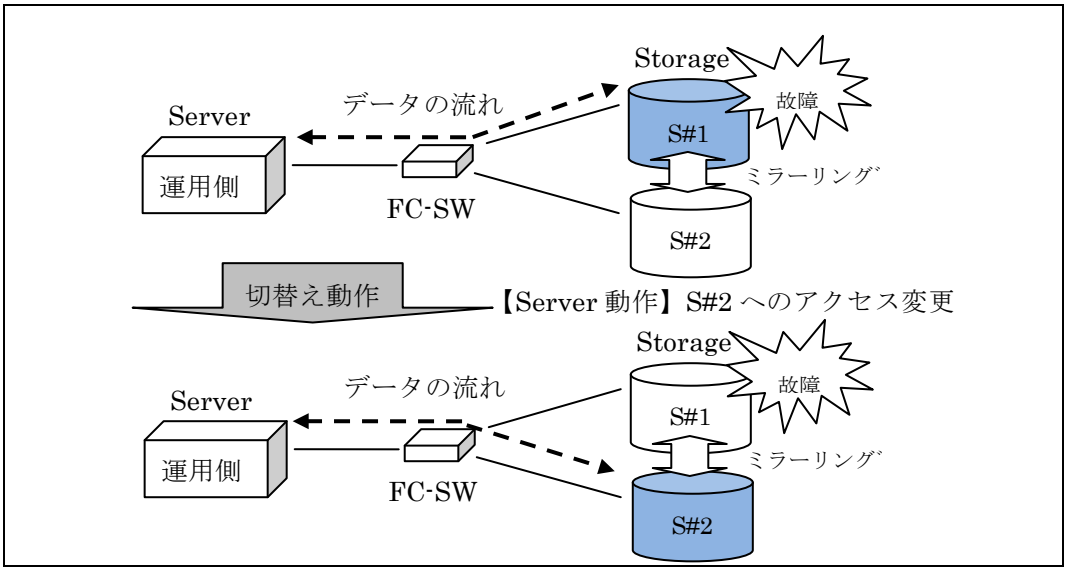


図 B-4-6　ストレージの確認例（ストレージ故障）

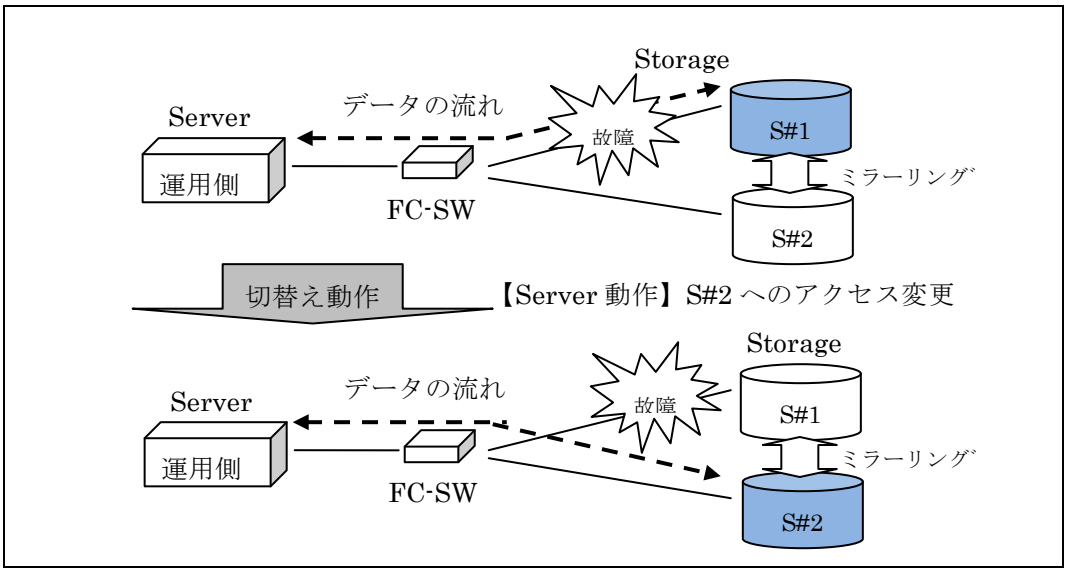


図 B-4-7　ストレージの確認例（FC 経路故障）

本システム構成において、疑似故障発生ツールで用意する疑似故障は表 B-4-11 の通りである。

なお、故障発生後には復旧作業を検証する必要があるため、復旧するための機能も備える。

これにより、異常時の検証の流れがスムーズに行える。

表 B-4-11　疑似故障一覧

No.	装置	疑似故障	復旧機能
1	Server	CPU 故障	パワーオン復旧
2	Storage	HDD 故障	HDD 復旧
3	FC-SW	FC ポート故障	FC ポート復旧
4	Router	LAN ポート故障	LAN ポート復旧

（3）　検証作業

検証は、3 つの視点（システム基盤、運用、利用者）で故障発生前後、復旧時における非機能要件事項を確認していく。

非機能要求において、サービス切替時間や目標復旧時間などの時間的要素があるため、各装置の時間を同期させ、時間を含めたログ採取等の考慮をしておくことが大切である。

整理すると、表 B-4-12 のようになる。

表 B-4-12　検証確認ポイント

時系列	システム基盤	運用	利用者
検証前	・正常状態であることの確認	・システムが正常であることを把握できること	
故障発生時	（ハード故障） ・冗長化部分の切替時間の確認	・障害箇所の特定がすぐに可能か ・システム切替えが発生したことを把握できるか ・異常通知までの時間が目標時間以内か	・業務への影響度合いは妥当か（業務停止、レスポンス低下）
異常状態 （縮退運転）		・手順書にて状態を確認できるか ・故障個所以外に異常が通知されていないか	・業務への影響度合いは妥当か（業務停止、レスポンス低下）
復旧作業	（ハード復旧）	・手順通り復旧できるか	・業務への影響はないか

5. 達成度の評価、取組み結果

ハードウェア故障の観点から検証項目を抽出し、疑似故障発生ツールを適用した検証事例を 2 例ほど紹介する。

(1)　物品取引システムの事例

事例システムの概要を以下に示す（表 B-4-13）

表 B-4-13　物品取引システムの概要

No.	項目	概要
1	非機能要求レベル	社会的影響が限定
2	検証実施工程	運用テスト
3	システム基盤概要	サーバ冗長構成（自動切替え）
4	システム基盤規模	サーバ複数台
5	アプリケーション	物品取引システム（クライアント・サーバ構成）
6	運用	運用監視サーバでの一括監視
7	保守	復旧手順書を準備中

本事例では、クライアント複数台から取引業務を実行中にサーバに疑似故障を発生させ、自動切替えによる待機サーバへの切替えを実施した。

本システムでは、故障が発生してから自動切替えが完了し取引業務が再開されるまでを「サービス切替時間」として定義し、目標値を設けていた。(表 B-4-14)

表 B-4-14　サービス切替時間

No.	非機能要求グレード		指標	目標値	実測値
	大項目	中項目			
1	可用性	継続性	サービス切替時間	３０分以内	１５分

検証の結果、目標値以内でサービスが切替わることが確認できた。

さらに、今回の検証でサービス停止時間を実体験することで、要求定義で設定したサービス切替時間の目標値では、実運用において影響が出ることが分かり、目標値の見直しを行うこととなった。

(2)　環境貸出サービス(IaaS)の事例

事例システムの概要を以下に示す。(表 B-4-15)

表 B-4-15　環境貸出サービス（IaaS）の概要

No.	項目	概要
1	非機能要求レベル	社会的影響が極めて大きい
2	検証実施工程	検証・受入テスト
3	システム基盤概要	サーバ、ストレージ、ネットワーク冗長構成 （自動切替え）
4	システム基盤規模	各装置毎が数十台
5	アプリケーション	仮想サーバ環境
6	運用	運用監視サーバでの一括監視
7	保守	復旧手順書

本事例では、受入テストにて異常系試験として、各冗長構成部分について疑似故障を発生させ、自動切替動作の確認、復旧手順書に従った作業確認を実施した。

システム基盤の装置台数が多く、構成的に複雑であるため、運用中に故障が発生した場合、長時間ダウンとなる問題を検出している。

以下が検出した問題事例である。（表 B-4-16）

表 B-4-16　問題事例

No.	分類	事例
1	設計ミス	非冗長構成部分の故障により業務へ影響する
2	環境構築ミス	設定ミスにより冗長切替え処理が動作せず
3	手順書ミス	復旧手順書通りに作業を行っても復旧できない（手順不足）

6. 今後の取組みと考察

今回の取り組みにより、「非機能要求グレード」を活用することにより、検証項目と確認内容の網羅性を確保しつつ、疑似故障ツールを用意したことにより、効率的に耐故障性の検証が可能となった。これにより、従来は実際の運用に入ってからでなければ見つけることができなかったハードウェアの故障を起因としたトラブル要因を未然に検出し、対策を講じることにより、運用品質の向上へ貢献ができた。

さらに、システムのライフサイクルにおいて、高信頼なシステムを維持していくためには、異常系試験に要する時間が課題となり、また、実際に起こりえるハード故障と疑似故障のギャップをどう埋めていくか、など、更なる取組みが必要である。

(1)　異常系試験の省力化、期間短縮化

クラウドシステムに代表されるような、利用者の増加に合わせて、システムを増設するようなシステムにおいて、環境構築から運用開始までの期間を短縮したいが、環

境構築ミス等の懸念があるため、異常系試験を省力化し、確実に短期間で実施できる取組みが必要である。

今回、疑似故障発生をソフト化したことにより、自動化が容易となっている。また、結果の妥当性を自動的に判断できる仕組みを検討中で、これらを実現することにより、検証の流れがすべて自動化することができる見込みである。

(2)　疑似故障の充実

今回の取組みで用意した疑似故障は、ハードが故障して動作が停止するような事象を実現しているが、実際の故障においては、中途半端に故障し、動作が不安定となるケースもある。

そのため、疑似故障パターンの充実が必要となっている。

また、高信頼性システムの開発・運用の現場において、OSS 活用やアジャイルや DevOps などの考え方の取込みが盛んに行われるようになっており、本事例で紹介したウォータフォール型プロセスでの品質作り込みでは困難なケースがでている。この流れにあった品質を担保するプロセスを検討し、対応していく必要がある。

B－5

単体ランダムテスト実行／可視化ツール “Jvis”の適用事例 [1]

1. 概要

テスト工程は、ソフトウェア開発全体の工数の中で 50%を占めると言われ、非常に多くのコストを要する[1]。また、ソフトウェアの大規模化、複雑化、短納期化に伴い、ソフトウェアを十分に検証することが難しくなっている[2]。このような背景から、テスト工程のコスト削減やテスト効率の向上を目的とした幅広い研究が行われている。

一方、Web サービスの開発やスマートフォンのアプリケーション開発など、環境の変化がめまぐるしく行われ易い領域では、短いスパンで実装して評価することを何度も繰り返し行うアジャイル開発を採用していることが多い。評価を何度も繰り返す必要があるため、アジャイル開発においてもテスト効率の向上、特に単体テストの効率向上が求められている。

テスト工程は、単体テスト、結合テスト、システムテスト、受け入れテストの 4 つの工程で構成される。単体テストは、テスト工程の最も初期に実施され、ソフトウェアにおける最小単位であるモジュールに焦点を当てたテストである[3]。この時点で多くの欠陥を検出できれば、後工程での手戻りを防ぐことができる。コスト削減と品質向上の両面において、単体テストは重要な工程である。

また、テスト工程の最も初期に実施される単体テストは、検証すべきテスト項目が多く存在する。単体テストを手作業で行う場合、全てのテスト項目を検証するためには膨大な時間を要する。この問題を解決するために、単体テストを自動化する場合がある。しかし、テストを自動化した場合、多くのテストデータを自動で実行できる一方で、同時に大量の実行結果が出力されてしまう。大量に羅列された実行結果や自然言語で記述された実行ログから、テストの実施状況を理解することは容易ではなく、自動化した単体テストの効率化を阻害する要因となっている。

本編では、この問題に対する解として、宮崎大学で研究開発された、単体テストを自動化し、かつ、テスト実施状況のリアルタイムな可視化を同時に行う単体テスト可視化ツール Jvis(Tool for Java programs to visualize unit testing)の事例を紹介する。

Jvis が最も有効な適用対象となる継続的インテグレーションによる開発環境では、単体テストの自動化は重要な取り組みであり、複数のツールが提供されている。しかし、単体テス

[1] 事例提供：宮崎大学　工学教育研究部　片山　徹郎 氏

トの自動化とリアルタイムな可視化を、それぞれ別のツールを複数組み合わせて実施する必要があるなど、利用するには課題が多い。また、継続的インテグレーションにおいては、Jvis が提供する単体テストが開発者のデバック支援でなく、テストカテゴリとしての回帰テストを実現することで、信頼性を確保することが目的であることも重要な開発テーマである。

Jvis によって、単体テストの効率化を図りつつ、テスト実施状況のリアルタイムな可視化により、テスト実施状況把握の効率化を図る。

Jvis は、テスト対象コードに対して命令文および分岐の網羅状況を取得するために必要な処理を追記したカバレッジ計測コードを生成する。また、テスト対象コードに対するテストデータを自動生成し、カバレッジ計測コードに与えることによって、テストを自動実施する。カバレッジ計測コードの実行によって取得したテスト対象コードの網羅状況を元に、テスト実施状況の可視化をリアルタイムに行う。テスト終了後、実行結果を HTML(Hyper Text Markup Language)形式のファイルに出力する。

2. 単体テスト可視化ツール Jvis

単体テスト可視化ツール Jvis の機能、構成等について説明する。

2.1. 機能

単体テスト可視化ツール Jvis は、主に 3 つの機能を有する。

(1) テストデータの自動生成及びテストの自動実行

テスト対象コードに対して、ガウス分布[7]に基づいたテストデータを自動生成し、これを入力することによってテストを自動実施する。テスト実施に際して、ユーザはテストコードを準備する必要がない。

(2) C0 及び C1 の計測

テスト実施によって、テスト対象コードの C0(命令網羅)及び C1(分岐網羅) [3]を計測する。このうち C1 をテストの終了基準に用いている。C1 の値が 100%に達すると、テストを自動的に終了する。

(3) テスト実施状況のリアルタイムな可視化

テストされた命令文のハイライト及び各行の実行回数の表示を随時行うことによって、テスト実施状況をリアルタイムに可視化する。

Jvis の外観および各 UI(User Interface)の概要を図 B-5-1 に示す。

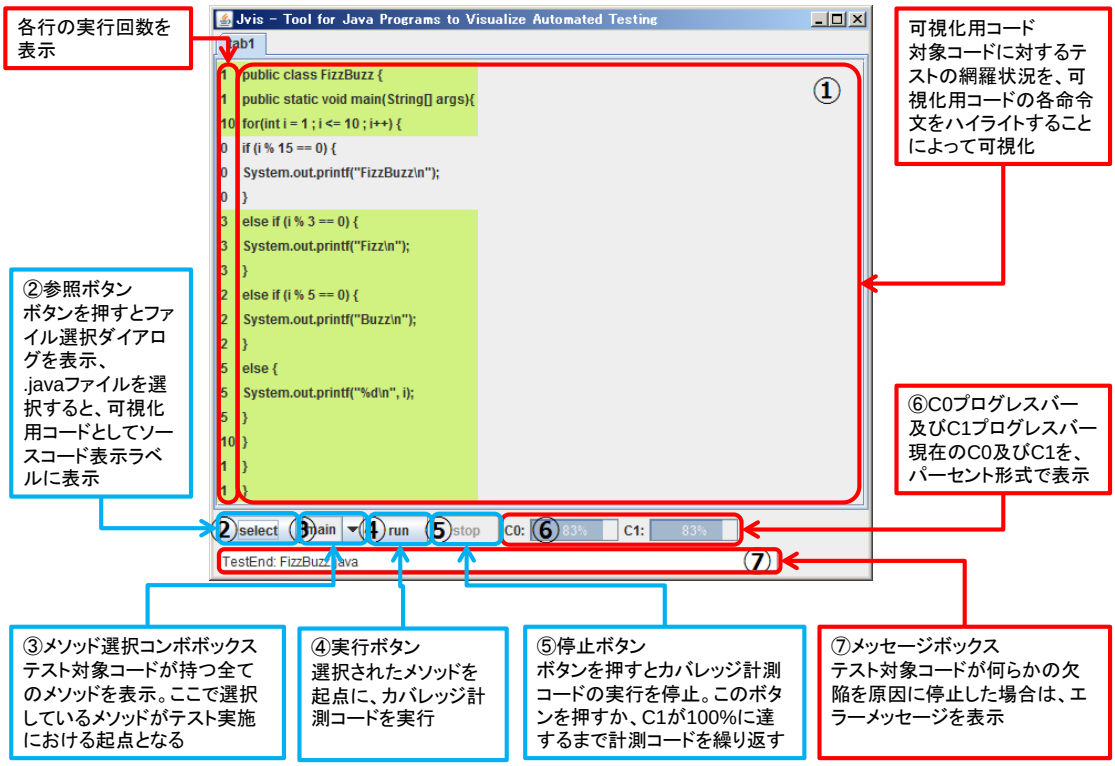


図 B-5-1　単体テスト可視化ツール Jvis の外観

2.2. 構造および処理

Jvis の全体構成を、図 B-5-2 に示す。Jvis は、解析部、カバレッジ計測コード生成部、テスト実施部、テストデータ生成部、終了処理部の 5 つで構成される。

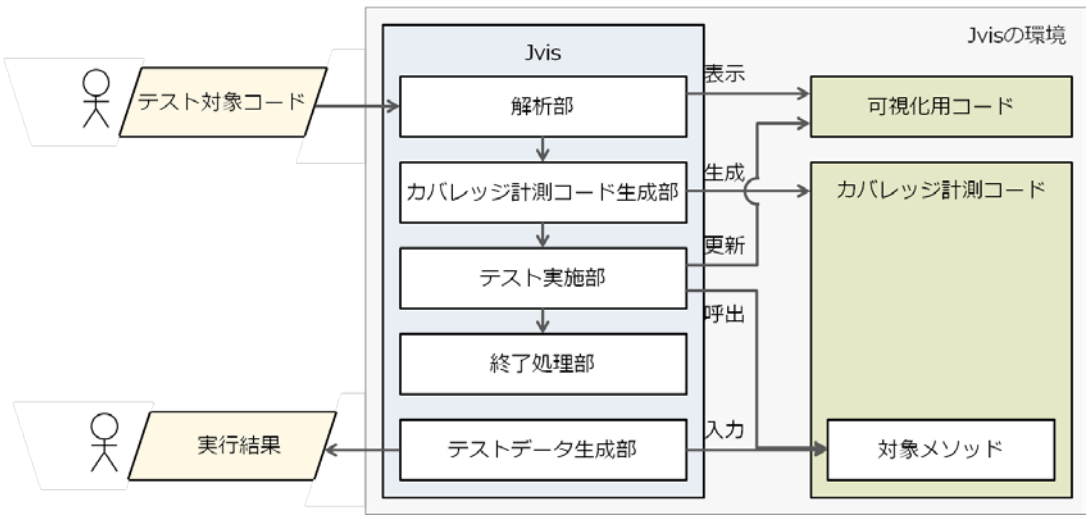


図 B-5-2　単体テスト可視化ツール Jvis の全体構成

(1)　解析部

解析部は、ユーザが選択したテスト対象コードを読み込み、正規表現を用いたパターンマッチングによる解析を行い、可視化用コードを作成する。可視化用コードとは、C1 に基づいたテストの実施状況をユーザに伝えるために、ソースコード表示ラベルに表示されるコードである。テスト対象コードに対して、false 方向の分岐を持たない分岐命令、即ち else 文を持たない if 文に対しては else{} を、default 文を持たない switch case 文に対しては default: を挿入している。

解析と同時に、テスト対象コードからクラス名及び引数を含むメソッド名の抽出を行う。抽出したクラス名は、テスト実施部がテスト対象コードを呼び出す際に参照し、メソッド名は、メソッド選択コンボボックスに一覧として表示する。

さらに、テスト対象コード中に記述されている数値、文字および文字列を抽出する。抽出した数値、文字および文字列は、テストデータ生成部がテストデータを生成する際に参照する。

(2)　カバレッジ計測コード生成部

解析部の処理が終了すると、カバレッジ計測コード生成部に移行する。

カバレッジ計測コードとは、可視化用コードに対して、命令文及び分岐の網羅状況を取得するために必要な処理を追記したコードである。

カバレッジ計測コード生成部では、解析部で作成した可視化用コードに対して、1 行ずつ正規表現を用いたパターンマッチを行い、各パターンに対応した処理を施す。

a. 命令文に対するカウンタの挿入

   カウンタの挿入によって文法エラーが発生することのない全ての命令文が対象である。

   原則として各命令文の直後に、挿入したカウンタが到達不能コードになってしまう場合には命令文の直前に、カウンタを挿入する。

b. 分岐命令に対するカウンタの挿入

   全ての分岐命令が対象である。各分岐命令の直後にカウンタを挿入する。

c. 戻り値の記録

   全ての return 文が対象である。各 return 文の直前に戻り値の記録を行う代入文を挿入する。

命令文及び分岐命令に挿入したカウンタにより、網羅状況の取得及び各行の実行回数の記録を行う。各カウンタの初期値は 0 とし、命令文及び分岐命令を実行することによって、該当行に対応する配列要素をインクリメントする。各カウンタは、文法上の制約に応じて、命令文の直後または直前に挿入するか、いずれも不可能な場合は挿入しない。カウンタを挿入しない命令文については、前後の命令文の網羅状況から判断して、実行回数の更新及び背景のハイライト処理を行う。

また、すべての return 文に対して戻り値の記録を行う代入文を挿入する。記録した

戻り値は、終了処理部によって実行結果を出力する際に参照する。

(3)　テスト実施部

カバレッジ計測コードの生成が終了すると、テスト実施部に移行する。

テスト実施部では、ユーザが選択するメソッドを起点に、カバレッジ計測コードを実行する。引数を持つメソッドを呼び出す場合、テストデータ生成部によって引数の型に対応したテストデータを生成し、対象メソッドに対して入力する。また、カバレッジ計測コードの実行によって取得した命令文及び分岐の網羅状況を元に、Jvis のソースコード表示ラベルに表示された可視化用コードの該当行を随時ハイライトする。網羅した命令文は緑色に、何らかの欠陥が原因でプログラムが停止した場合はテストを停止した上で停止箇所の命令文を赤色に、それぞれハイライトする。

各行の左端に実行回数を随時表示する。回数が増すに従い、回数表示の背景色が段階的に濃くなる。

カバレッジ計測コードの実行が 1 回完了する度に C1 の値を確認し、C1 の値が 100%に達したらテストを終了する。100%に達しない場合でも、停止ボタンを押すことによって終了できる。

(4)　テストデータ生成部

テストデータ生成部では、テスト実施部で呼び出すメソッドが持つ引数の型に応じて、以下の平均値及び分散を持つガウス分布に基づいたテストデータを生成する。

μ = テスト対象コード中に記述されている数値及び文字、文字列

$\sigma 2 = 2.0$

平均値は、テスト対象コードに記述されている全ての数値を対象とし、文字及び文字列はシングルまたはダブルクォーテーションで囲まれているものを対象とする。文字については、文字コードに基づいて数値化し、これをガウス分布における平均値としている。文字列については、1 文字ずつ区切り、ガウス分布に基づいてそれぞれのテストデータを得た後、それらを連結する。

例として、文字列"ABC"に対するテストデータ生成のメカニズムを、図 B-5-3 に示す。

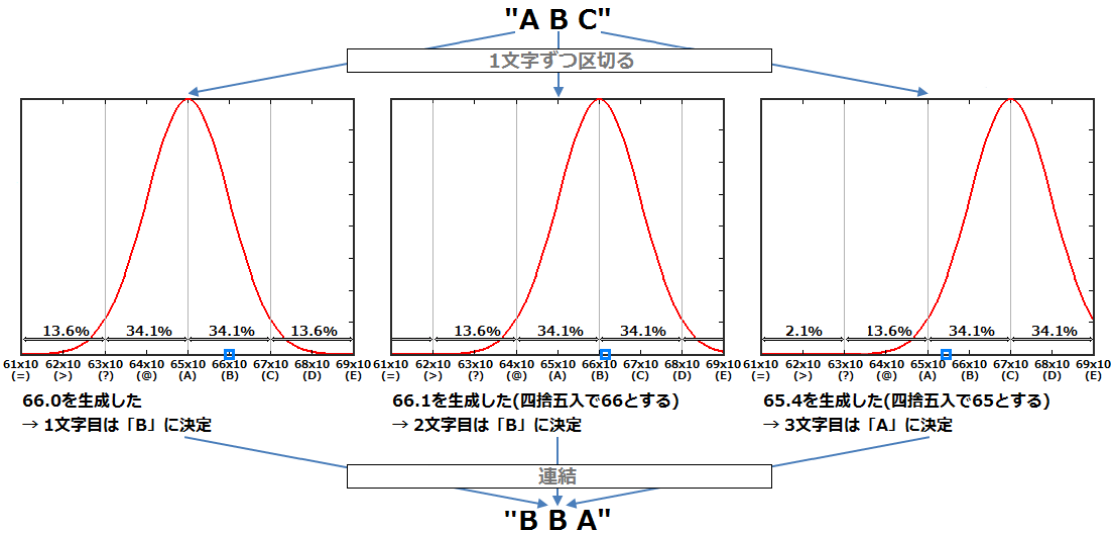


図 B-5-3　文字列"ABC"に対するテストデータ生成のメカニズム

同一の引数に対して複数の平均値を抽出した場合は、その中から 1 つを無作為に選択し、それを平均値とする。テスト対象コード中に平均値の対象となる数値、文字および文字列が存在しない場合は、各データ型に対して設定した最小値～最大値の範囲から一様乱数を生成する。各データ型に対する乱数の生成範囲を表 B-5-1 に示す。

表 B-5-1　各データ型に対する乱数の生成範囲

データ型	乱数の生成範囲
Int	-32768～32767
long	-2147483648～2147483647
float	32768.000～32767.000
double	2147483648.000～2147483647.000
byte	0～255
char	'0'～'9', 'a'～'z', 'A'～'Z'
String	'0'～'9', 'a'～'z', 'A'～'Z'で構成した 1～10 文字の文字列
boolean	true 又は false

現在、テストデータ生成部が生成できるテストデータの型には制限があり、プリミティブ型にしか対応していない。

(5)　終了処理部

テスト実施部が終了すると、終了処理部に移行する。

終了処理部では、実行結果を HTML 形式のファイルに出力する。可視化用コードをハイライトした網羅状況と各行の実行回数、C0 及び C1 の値を、メインページ[index.html]に出力する。また、各メソッドに対するテストデータと、各テストデータに対応する戻り値を、サブページ[メソッド名(行数).html]に出力する。サブページは、テスト対象コードが持つメソッドの数だけ生成する。

3. 取り組みに対する評価（動作検証）、適用の要点

Jvis が正しく動作することを検証するため、4 パターンの Java プログラム「在庫管理プログラム」を適用した。動作確認済みの在庫管理プログラム(正常版)を元に、3 件のバリエーションを作成し、想定通りの結果が得られることを確認した。

a. 正常版
在庫データの読込みと、注文に対する在庫確認の機能を持つ 30 行の Java プログラムである

b. 無限ループを含む在庫管理プログラム
正常版に対して、4 行目の変数 i のインクリメント処理を削除する。while 文の条件式に含まれる変数 i の値が変化しないため、無限ループが発生する。

c. 到達不能コードを含む在庫管理プログラム
正常版に対して、4 行目の while 文の条件式が満たされることがないように条件式を変更する。while 文内のコードは到達不能コードとなる。これにより、在庫の充足又は不足、商品コードの有無にかかわらず、Inventory_check メソッドは戻り値として 0 を返す。

d. 実行時エラーを含む在庫管理プログラム
正常版に対して、4 行目の while 文の条件式を変更し、変数 i が配列の要素数を超えて、5 行目の配列参照時に、実行時エラーの 1 つである IndexOutOfBoundsException が発生するようにする。

3.1. 正常な在庫管理プログラムによる動作検証

在庫管理プログラムの一部である Inventory_check メソッドを、図 B-5-4 に示す。Inventory_check メソッドは、注文に対する在庫の不足又は充足を確認し、対応する値を戻り値として返す。

```
public static int Inventory_check(int number, int amount){
  String[] Inventory = null;
  int i=0;
  while(i < Inventory_load().size()){
    Inventory=((String)Inventory_load().get(i)).split(",",0);
    i++;
    if(Inventory[0].equals(String.valueOf(number))){
      if(Integer.parseInt(Inventory[2])-amount < 0){
        return -1;                  //在庫不足
      }
      else{
        return 1;                   //在庫充足
      }
    }
    else{}
  }
  return 0;                         //商品コードなし
}
```

図 B-5-4　在庫管理プログラムの一部(Inventory check メソッド)

在庫管理プログラムを元に、解析部が生成した可視化用コードを図 B-5-5 に示す。C1 に基づいたテスト実施状況を可視化するため、false 方向の分岐を持たない if 文に対して、空の else 文を挿入している。

```
public static int Inventory_check(int number, int amount){
  String[] Inventory = null;
  int i=0;
  while(i < Inventory_load().size()){
    Inventory=((String)Inventory_load().get(i)).split(",",0);
    i++;
    if(Inventory[0].equals(String.valueOf(number))){
      if(Integer.parseInt(Inventory[2])-amount < 0){
        return -1;                  //在庫不足
      }
      else{
        return 1;                   //在庫充足
      }
    }
  }
  return 0;                         //商品コードなし
}
```

図 B-5-5　在庫管理プログラムの一部(Inventory check メソッド)の可視化用コード

可視化用コードを元に、カバレッジ計測コード生成部が生成したカバレッジ計測コードの一部を図 B-5-6 に示す。

```
public static int Inventory_check(int number, int amount){Jvis.statement[3]++;
   String[] Inventory = null; Jvis.statement[4]++;
      int i=0; Jvis.statement[5]++;
      while(i < Inventory_load().size()){ Jvis.statement[6]++; Jvis.branch[6]++;
         Inventory=((String)Inventory_load().get(i)).split(",",0); Jvis.statement[7]++;
         i++; Jvis.statement[8]++;
         if(Inventory[0].equals(String.valueOf(number))){ Jvis.statement[9]++;
Jvis.branch[9]++;
            if(Integer.parseInt(Inventory[2])-amount < 0){ Jvis.statement[10]++;
Jvis.branch[10]++;
               Jvis.statement[11]++;
Jvis.returnStorage[Jvis.NumOfRuns][11][Jvis.NumOfEle++]=Jvis.visualizationCode.get(11);
return -1;
            }
            else{ Jvis.statement[13]++; Jvis.branch[13]++;
               Jvis.statement[14]++;
Jvis.returnStorage[Jvis.NumOfRuns][14][Jvis.NumOfEle++]=Jvis.visualizationCode.get(14);
return 1;
         }
      }
      else{Jvis.branch[17]++;}
   } Jvis.branch[18]++;
   Jvis.statement[19]++;
Jvis.returnStorage[Jvis.NumOfRuns][19][Jvis.NumOfEle++]=Jvis.visualizationCode.get(19);
return 0;
}
```

図 B-5-6　在庫管理プログラムの一部(Inventory Check メソッド)のカバレッジ計測コード

Inventory_check メソッドを起点に在庫管理プログラムに対するテストを実施し、終了した後の Jvis の外観を図 B-5-7 に示す。テスト対象コードをハイライトし、命令文ごとの実行回数を示すことによって、テスト実施状況をリアルタイムに可視化できることを確認した。

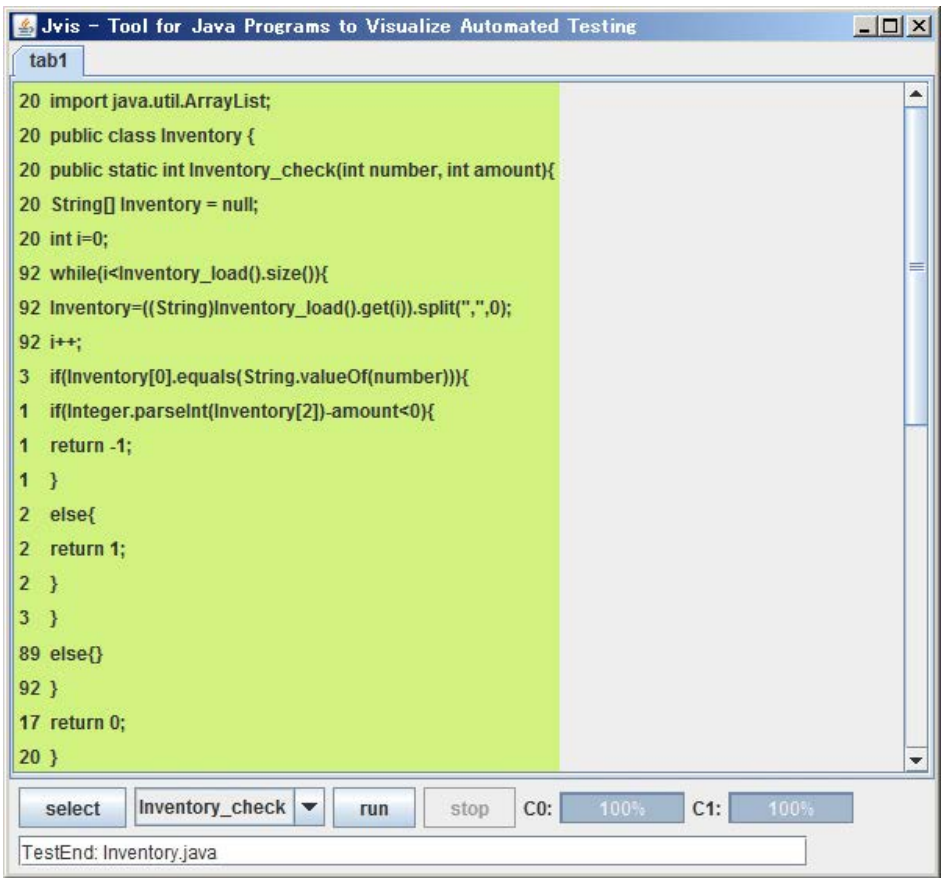


図 B-5-7　正常な在庫管理プログラム適用後の Jvis の外観

正常版の在庫管理プログラムは、テストの終了条件である C1 を 100%満たすことができたため、カバレッジ計測コードの実行は自動的に停止した。全ての行がハイライトされ、全ての命令文及び分岐を網羅できたことを示している。

3.2.　無限ループを含む在庫管理プログラムによる動作検証

無限ループを含む在庫管理プログラムのカバレッジ計測コードを 1 分間実行し続けた Jvis の外観を、図 B-5-8 に示す。無限ループが発生している様子を可視化できることを確認した。

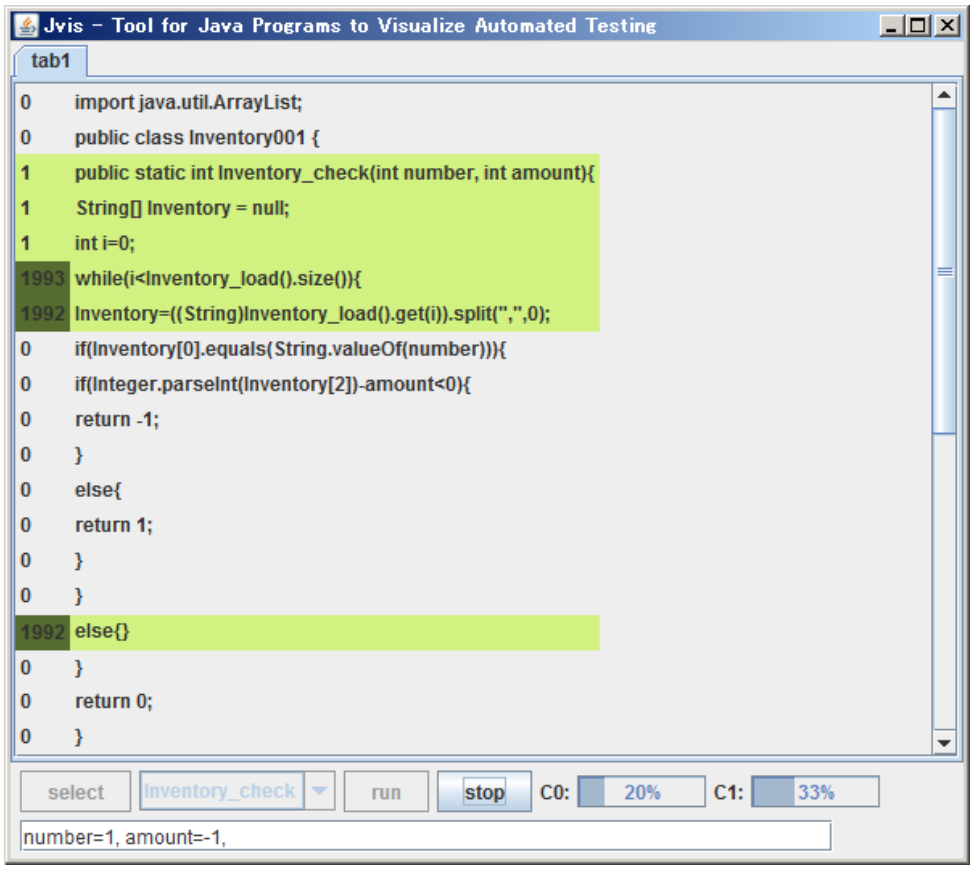


図 B-5-8　無限ループを含む在庫確認プログラムを 1 分間実行し続けた Jvis の外観

Inventory_check メソッドは 1 度しか呼び出していないものの、無限ループが起こっている命令文(6、7 および 16 行目)の実行回数のみが増え続け、その箇所の背景色が濃くなっている。正常版と比べて、明らかに異常な振る舞いが起こっていることを示している。

3.3.　到達不能コードを含む在庫管理プログラムによる動作検証

到達不能コードを含む在庫管理プログラムのカバレッジ計測コードを 1 分間実行し続けた Jvis の外観を図 B-5-9 に示す。到達不能コードをテスト実施中に検出できることを確認した。

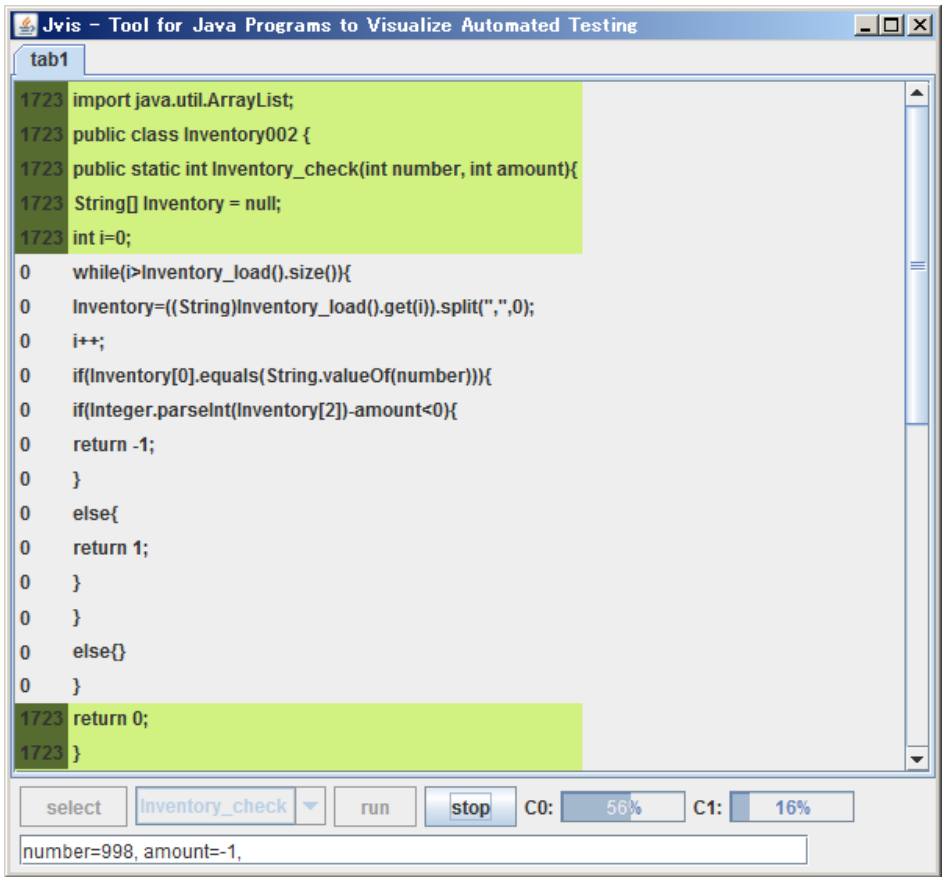


図 B-5-9　到達不能コードを含む在庫確認プログラムを 1 分間実行し続けた Jvis の外観

while 文の中、すなわち到達不能コードはハイライトせず、現在の C0 及び C1 の値を示すプログレスバーは、それぞれ 56%と 16%で頭打ちになっている。実行結果を確認したところ、有効な商品コードを引数として入力しているにもかかわらず、全ての戻り値が return 0;であった。

3.4.　実行時エラーを含む在庫管理プログラムによる動作検証

実行時エラーを含む在庫管理プログラムを Jvis に適用し、実行時エラーの検出によってテストを停止した後の Jvis の外観を、図 B-5-10 に示す。エラーを検出した命令文を赤色でハイライトし、メッセージボックスにエラーに関する情報を出力することを確認した。

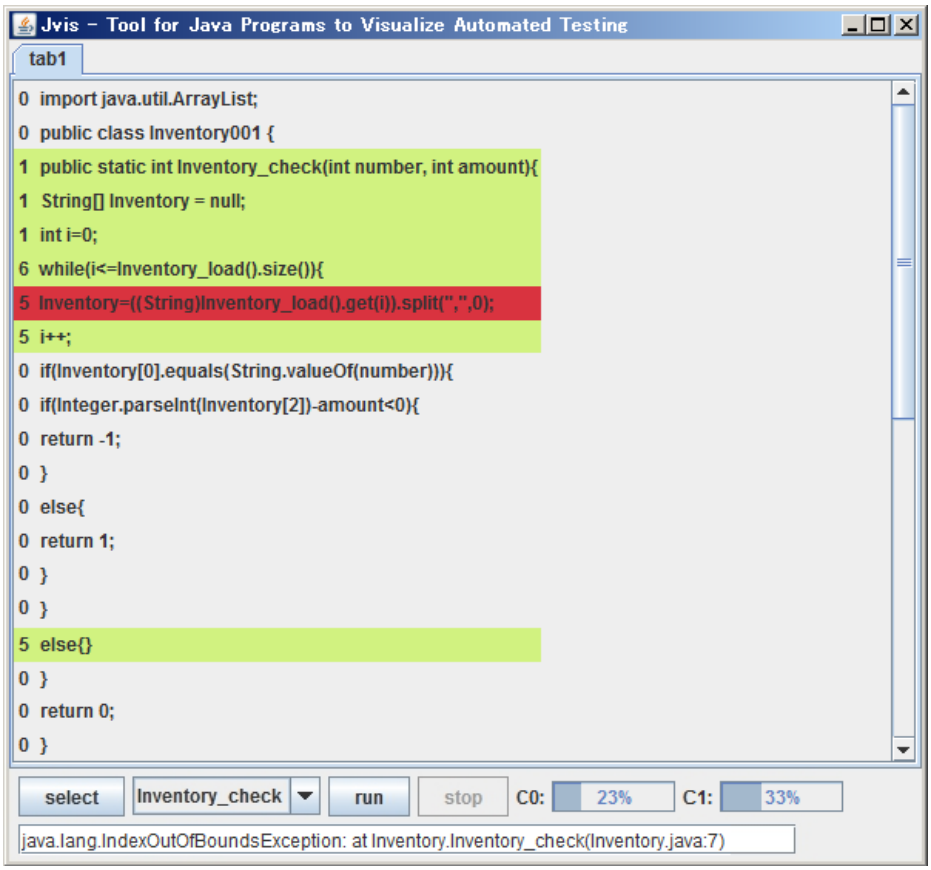


図 B-5-10　実行時エラーの検出によってテストを停止した後の Jvis の外観

実行時エラーによってプログラムは正常に終了しなかったが、エラー検出によるテスト停止までの可視化処理は正常に実施できている。

実行時エラーを検出した 7 行目の命令文を赤色でハイライトしているが、エラーの直接の原因である 6 行目をハイライトすることはできない。メッセージボックスには、IndexOutOfBoundsException を検出したこと、検出した行数など、エラーに関する情報が表示されている。

4. 考察

本事例では、テスト実施状況の理解および確認に費やす手間の削減を目的として、単体テスト可視化ツール Jvis を紹介した。Jvis は、テスト対象コードに対してカバレッジを基準とした自動テストを実施する。C1 をテストの終了基準とし、テスト対象コードが C1 を 100%満たすことによって自動的に単体テストを終了する。このため、実装終了後すぐに、テスト実施が可能であり、短いスパンで評価を何度も繰り返し行うアジャイル開発においても、適用可能である。

また、命令文および分岐の網羅状況を取得するカバレッジ計測コードを生成し、実行する

ことによって、テスト実施状況のリアルタイムな可視化を実現する。リアルタイムな可視化機能によって、現在のテスト実施状況をテスト実施中に確認し、把握することが可能である。テスト実施後、実行結果を HTML 形式のファイルに出力する。

4.1. Jvis の問題点

単体テスト可視化ツール Jvis の問題点を、以下に示す。

(1) プログラムの計装による実行時間のずれ

テスト実施において、Jvis が実際に実行するプログラムは、テスト対象コードを計装したカバレッジ計測コードである。テスト対象コードとカバレッジ計測コードの論理動作は同じだが、実行時間にずれが生じる。時間的制約の厳しいプログラムにおいては、実行時間のずれによって、正常な動作が行われない可能性がある。

(2) テストデータ生成における制限

現在、Jvis が生成できるテストデータの型には制限があり、プリミティブ型にしか対応していない。Java 言語において使用頻度の高いオブジェクト型のテストデータは生成することができない。このため、適用可能なプログラムに制限がある。

(3) 不十分な解析性能

プログラムの解析手法として、正規表現を用いたパターンマッチを用いているため、プログラムの多様な記述に対応できていない。例えば、同一行に複数の命令文を記述したプログラムを Jvis に適用した場合、正確なカバレッジを取得することができない。命令文ごとに可視化処理を実施するには、ユーザによるプログラムの記述を制限し、命令文ごとに改行する必要がある。

(4) 期待値との照合にかかる手間

Jvis が入力するテストデータに対する戻り値を、期待値と照合することができない。テストデータに対する戻り値の正誤については、ユーザによる実行結果の確認が必要である。テストデータの数が多くなった場合、実行結果の確認には手間を要する。

(1) に関しては、命令文および分岐の網羅状況を取得する手段として、テスト対象コードを計装する手法以外にも、バイトコードを計装する手法や JVMTI(Java Virtual Machine Tool Interface) [8]を使用する手法がある[9]が、どの手法を採用してもこの問題を解決することはできない。プログラムの実行時情報を取得する際に、避けることのできない問題である。

(2) に関しては、既存の研究であるシンボリック実行の概念を用いることによって解決できる。シンボリック実行とは、具体的なテストデータに代わり、象徴的なデータをプログラムに与える手法のことである。この手法は、Java PathFinder[10]などのテスト自動実行ツールによって、実用化されている。本事例においては、同様の手法を導入するに至っておらず、今後の課題である。

(3) に関しては、構文解析器の導入を考えている。

構文解析器による字句レベルの解析や、プログラミング言語の構文に則った解析を行うことによって、同一行に複数の命令文を記述したプログラムを適用した場合でも、適切な改行処理や計装が可能になると考えている。

（4）に関しては、テストデータの再利用による解決策を考えている。

Jvis は、テスト実施時に入力したテストデータと、テストデータに対する戻り値を、HTML 形式のファイルとして保存する。2 回目以降のテストあるいは回帰テスト[3]実施時に、初回に入力したテストデータを再利用し、また、初回に取得した戻り値を期待値として利用すれば、テストデータに対する戻り値と、期待値との比較が可能になると考えている。

4.2. テスト実施に関する考察

Jvis は、ガウス分布に基づいたテストデータの自動生成を行う。テスト対象コード中に記述されている数値および文字、文字列を平均値とし、これに近いテストデータを、短時間で大量に生成および入力する。適用例で示した 30 行の在庫管理プログラムに対しては、テスト終了条件を満たすまでの 20 個のテストデータを、約 1 秒間で生成および入力している。適用例で示した在庫管理プログラムに対して、完全に無作為なテストデータ生成手法を用いてテストを実施した場合、条件式を満たすテストデータを生成することが難しく、テスト終了条件を満たすまでに多くの時間を要する。また、プログラム中の入力条件又は分岐条件式に範囲指定がある場合、「以下」と「未満」の取り違えやコーディングにおける「≧」と「＞」の記述ミスなどの誤りが起こりやすい。ガウス分布に基づいたテストデータの自動生成手法を採用することによって、完全に無作為なテストデータ生成手法に比べて、同値分割に基づく境界値[3]周辺の値を検証できる可能性が高く、単体テストを効率化できる。

テストデータ生成に関する研究事例として、シンボリック実行やモデル検査、ランダムテストの概念を用いたテストデータ生成手法が提案されている。シンボリック実行およびモデル検査を用いたテストデータ生成ツールとして代表的なものに、CUTE[11]がある。CUTE は、プログラムに対するモデルを生成し、モデルに対するシンボリック実行を行うことによって、効率的なパス網羅を実現する。また、ランダムテストを用いたテストデータ生成ツールには、Randoop[12]がある。Randoop は、テストデータの入力によって得られるフィードバックを元に、効率的なテストデータを生成する。それぞれ、効率的なテストデータを生成する一方、CUTE の利用には、テストの"深さ"を指定する必要があり、ユーザはプログラムの構造を把握している必要がある。また、Randoop については、冗長性を避ける振る舞いからテストデータの生成数を抑制してしまうというデメリットがある。しかし、Jvis が採用するガウス分布に基づいたテストデータ生成手法は、テストデータを生成する過程に無作為な要素が多く、これらの研究と比較すると効率性で劣る。一方で、Jvis は、テストデータ生成機能とテスト実施状況の可視化機能を同時に実現することによって、自動テストによる大量のテストデータが、テスト対象コード中のどこをどれだけ網羅したのかを視覚的に示すことができる。

なお、Jvis は C0 および C1 に基づいたテストを実施しているため、到達不能コードの発見や網羅した命令文および分岐方向を少なくとも 1 回は実行できることを検証できる。また、動的なテストを実施することから、NullPointerException や NumberFormatException 、ArrayIndexOutOfBoundsException など、実行時のエラーを検出することができる。一方、実行時にテスト対象コード中の各パラメータが確保しているデータの正誤については、Jvis では検証することができない。Jvis は、多くのテストデータ入力によるテストを目的としている。このため、各パラメータが確保しているデータを確認するには、プログラムに対して別途デバッグを実施し、検証する必要がある。

4.3. 可視化機能に関する考察

Jvis は、テスト実施によるテスト対象コードの網羅状況をリアルタイムに可視化する。現在、テスト実施状況を可視化する幾つかのツール[4] [5] [6]が提案されているが、Jvis のようにテスト実施状況をリアルタイムに可視化するツールは存在しない。カバレッジ測定ツールとして、Java Coverage Validator[13]がある。Java Coverage Validator は、プログラムに対する引数を手動で与えることによって、実行された命令文を随時ハイライトする。

なお、Java Coverage Validator におけるテスト対象コードの網羅状況の取得には、JVMTI が使用されている。実行箇所をリアルタイムに可視化する点は、Jvis と同じ機能であると言えるが、Java Coverage Validator はデバッグを目的としたツールであり、テストを目的とした Jvis とはカテゴリが異なる。

テスト実施状況を可視化するツールとして著名なものに、EclEmma(Emma for Eclipse) [6] がある。EclEmma は、テスト対象コードのカバレッジを計測し、網羅状況を可視化する Eclipse[14]のプラグインである。テストの実施には、各テストケースを記述したテストコードの準備が必要である。テストコードの実行によって網羅した命令文は緑色に、幾つかある分岐方向のうち 1 つ以上を網羅した分岐命令は黄色に、網羅していない命令文は赤色にハイライトする。なお、EclEmma におけるテスト対象コードの網羅状況の取得は、バイトコードの計装により行われる。EclEmma による可視化処理実施後の Eclipse コードエディタのスクリーンショットを、図 B-5-11 に示す。EclEmma の可視化処理は、全てのテストが実施された後に行われる。このため、テスト実施中のテスト対象コードの振る舞いを確認することができない。また、何らかの欠陥を原因にテスト対象コードの実行が正常に終了しない場合には、可視化処理が正常に実施されない場合がある。一方、Jvis は、命令文および分岐の網羅状況を取得するカバレッジ計測コードを生成することによって、テスト実施における網羅状況を、テスト実施中にリアルタイムに可視化する。このため、テスト実施状況の把握を効率化できる。

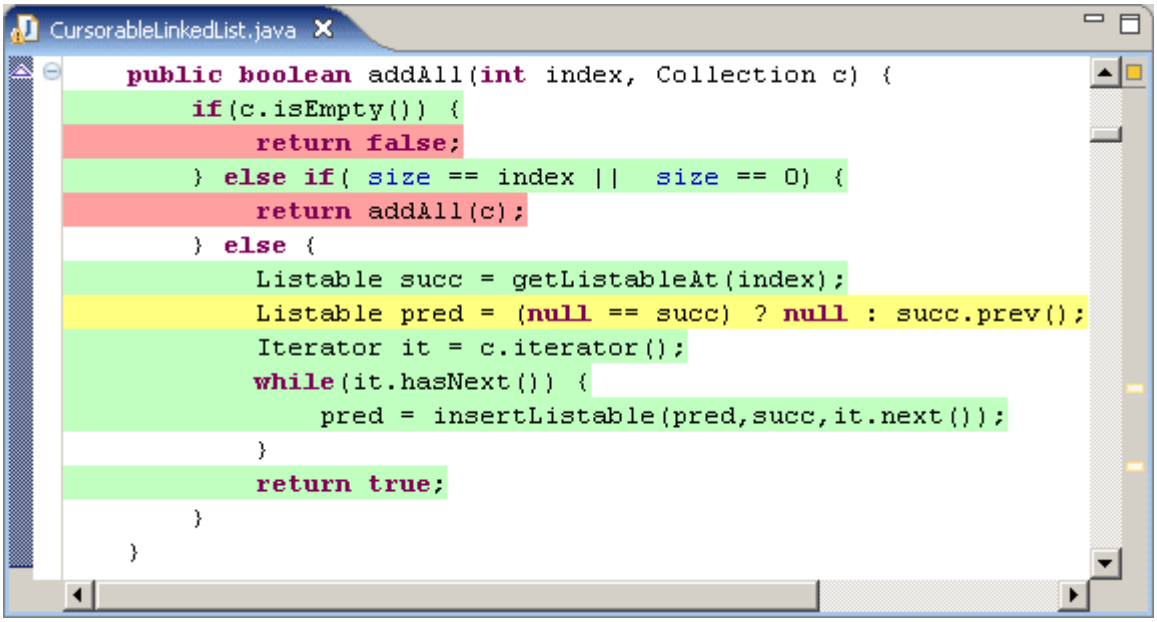


図 B-5-11　EclEmma による可視化処理実施後の Eclipse コードエディタの
スクリーンショット

5. おわりに

本事例では、単体テスト可視化ツール Jvis を紹介した。Jvis は、単体テストの自動化を行い、テスト実施状況のリアルタイムな可視化を行う。具体的には、カバレッジ計測コードを生成し、これに自動生成したテストデータを入力し単体テストを自動実施する。実装終了後すぐに、テスト実施が可能であるため、Web サービスの開発やスマートフォンのアプリケーション開発などで行われる、短いスパンで実装して評価することを何度も繰り返し行うアジャイル開発においても、単体テストの効率向上が見込め、ひいては、開発全体のコスト削減が見込める。

また、カバレッジ計測コードを用いて、命令文および分岐の網羅状況を随時取得しテスト実施状況をリアルタイムに可視化する。Jvis によって、単体テストの効率化を図りつつ、テスト実施状況のリアルタイムな可視化により、テスト実施状況把握の効率化を図る。

以下に、今後の課題を挙げる。

・可視性の向上

本提案手法では、可視化用コードをハイライトすることによって、C1 に基づいたテストの可視化を実現している。分岐や繰り返し処理における流れの可視性を向上するため、フローチャートなど、よりグラフィカルな可視化手法を用いることを検討する必要がある。

・実用性の向上

現在、テストデータ生成部が生成するテストデータの型には制限がある。特に、Java

言語によるプログラミングでは、プリミティブ型以外のデータ型を用いる頻度が極めて高い。また、テスト対象コードの解析に正規表現を用いたパターンマッチを行っているが、これではプログラムの多様な記述に対応できない。様々な構造や振る舞いを持つプログラムを適用し、問題点を洗い出した上で、自動生成するテストデータの型拡張や構文解析器の導入など、実用性の向上を行う必要がある。

・有用性についての実証実験

評価実験に使用した在庫管理プログラムは、プログラム中に具体的な数値や文字を含んでいるため、ガウス分布に基づいたテストデータの自動生成手法でも、C0 および C1 カバレッジを容易に満たすことができる。今後、被験者を用いた実証実験において、より実用的かつ大規模なプログラムに適用することによって、リアルタイムな可視化手法の有用性についての評価が予定されている。

参考文献

[1] E. Kit: Software Testing in the RealWorld: Improving the Process, Addison-Wesley Professional, 1995.
[2] R. L. Glass: Facts and fallacies of software engineering, Addison-Wesley, 2003.
[3] ソフトウェアテスト技術者資格認定の運営組織: ISTQB テスト技術者資格制度 Foundation Level シラバス 日本語版 Version 2011.J02, http://jstqb.jp/dl/JSTQB-SyllabusFoundation_Version2011.J02.pdf, 2011.
[4] V. Roubtsov: EMMA - a free Java code coverage tool, http://emma.sourceforge.net/, 2006.
[5] M. Doliner: Cobertura - A code coverage utility for Java, http://cobertura.sourceforge.net/, 2014.
[6] M. R. Hoffmann: EclEmma - Java Code Coverage for Eclipse, http://www.eclemma.org/, 2012.
[7] C. F. Gauss: Disquisitiones arithmeticae, Yale University Press, 1965.
[8] Oracle: JavaTM Virtual Machine Tool Interface Reference Draft 0.30.27, http://docs.oracle.com/javase/jp/1.5.0/guide/jvmti/jvmti.html, 2004.
[9] M. Shahid, S. Ibrahim: An Evaluation of Test Coverage Tools in Software Testing", Proc. International Conference on Telecommunication Technology and Applications 2011, 216-222, 2011.
[10] NASA Ames Research Center: JPF, http://babelfish.arc.nasa.gov/trac/jpf/, 2007.
[11] K. Sen, D. Marinov, and G. Agha: CUTE: a concolic unit testing engine for C, Proc. 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 263-272, 2005.
[12] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball: Feedback-directed random test generation, Proc. 29th International Conference on Software Engineering, 75-84, 2007.
[13] Software Verification Ltd.: Java Coverage Validator, http://www.softwareverify.com/java-coverage.php, 2012.
[14] Eclipse Foundation: Eclipse, http://www.eclipse.org/, 2014.

B－6

要求仕様明確化のための仕様記述技術(USDM)活用事例[1]

1. 概要

本編では、業務方法の変化に伴い、一層求められる様になった、要求の明確化を実現するための手法を適用した事例を紹介する。

自動車メーカーの A 社は、国内サプライヤとシステム開発を行う場合に、仕様の詳細を打合せ等で決めていく、通称「擦り合わせ開発」が進められていた。しかし、近年の情勢の変化から、海外企業を含んだ複数のサプライヤと取引するケースが増え、国内サプライヤと同様の詳細度合の要求仕様のままでは、今までのような開発ができない状況に陥っている。これらに対応するには、従来の属人的な業務の方式からプロセス型の業務方式への移行が必須となっている。当然ながら、業務方式が変化しても品質は維持されなければならず、これらの課題を解決する仮説として、要求（要求仕様）を明確化することを定義した。要求（要求仕様）の明確化とは、高いスキルを保有している技術者間で暗黙的に取り交わされる要件を仕様として文書化することである。要求が明確化されることで、属人性に依存してきた技術を明らかにでき、既存以外のサプライヤとの取引を可能にするとともに、サプライヤとの業務の分担を明確にすることができると考えた。本事例では、これら要求（要求仕様）の明確化を実現する手段として USDM[2] (Universal Specification Describing Manner)とテスト設計技術を利用する試みを行っている。また、要求仕様の記述の明確化と併せて、要求仕様を体系的に整理している。この体系的な整理には検証のテスト設計技術を利用した。

2. 取組みの目的

自動車業界では、ドイツ自動車工業会や AUTOSAR が策定しているモジュールやプラットホームの共通仕様をベースに、ハードやソフトを複数サプライヤのものと組み合わせて構成する流れになっている。その結果、自動車メーカーは取引サプライヤの数が増え、海外サプライヤとの取引も増える傾向にある。海外サプライヤとの取引では、既存の系列サプライヤと進められてきた大まかな要求に対して後から詳細要求および追加・変更要求を行う開発方式が通用せず、結果として要求（要求仕様）を明確にすることが必要となる。また海外サ

---

[1] 事例提供：株式会社ベリサーブ　冬川　健一 氏

[2] USDM(Universal Specification Describing Manner)は、システムクリエイツの清水吉男氏により考案された仕様記述の方式である。「仕様は基本的に要求の中の動詞にある」の考えに基づいて、要求仕様を階層構造で表現する。

プライヤなどからソフトウェアのみ購入してくる場合は、内部構造が不明なためブラックボックステストの検証で動作確認をする仕組みも必要となり、調達する購入品の評価基準も要求（要求仕様）が大きく関係する。A 社でも搭載するシステムユニットによっては、すでに大手のサプライヤが商品として保有しているものを購入する場合もある。この取組みの目的は、属人的、暗黙的で曖昧な仕様を排除して、第 3 者にも理解できる要求を仕様として文書化することで、要求（要求仕様）が明確になり、部品化されるソフトウェアの調達と受入を円滑にすることにある。また、仕様が文書化されることで、調達の選択肢も増やすことができる。

紹介する事例の取組み範囲を開発モデルの全体像(図 B-6-1 を参照)を示して明確にする。A 社の場合は開発・検証プロセスに W モデルを採用し定義している。W モデルは、上位の開発工程でテストエンジニアが参画し、開発工程に対応するテスト設計を実施するモデルである。V モデルに比べて、テスト設計を早期に実施することで、不具合の検出を早め、手戻りを少なくすることができる。このプロジェクトでは製造・開発（検証を含む）で、複数のサプライヤが参加して各々が担当する領域をカバーしている。

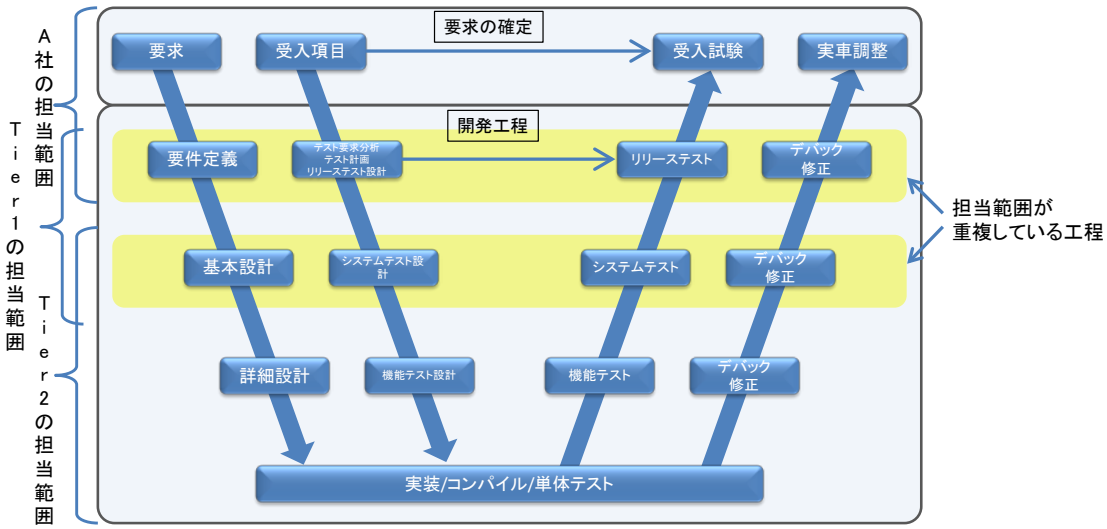


図 B-6-1　自動車メーカー、サプライヤの関係（役割の分担）

製品の要求・最終確認を実施する自動車メーカーに対して、要件定義から詳細設計とその検証を担当する Tier1 グループと、基本設計以降で実装をメインとする Tier2 グループのサプライヤが、それぞれ存在する。各々のサプライヤがカバーする領域は、重複するとともに役割が分担されている。要求元となる A 社および各サプライヤ間でのコミュニケーションがプロジェクトに大きく影響する。同じ Tier1 グループであっても、サプライヤごとに要求や受入の基準が異なると、工程下位のサプライヤに影響することは容易に想像できる。

従来の「擦り合わせ開発」では、開発工程が進むと徐々に要求（要求仕様）が定まっていく。そのため仕様確定の合意形成は、開発を行いながら進むので、担当者間で暗黙的な合意

形成を経て、仕様書に反映されないものが発生してくる。また、要求が定まっていない状態でも開発に着手できる反面で、要求が決まった時点で進行中の作成された仕様書と整合性を確認するなどの手戻り作業が必要となる。この整合性の作業が疎かになると、暗黙的で曖昧な仕様書ができあがる。これらの仕様書では、調達条件である要求（要求仕様）が明確に定義されていないために、広くサプライヤから調達することはできない。調達を実現するためには明確な要求（要求仕様）の作成が必要となる。

従来の「擦り合わせ開発」の元での要求（要求仕様）の作成および要件定義では、次のような課題が顕在化している。

- 暗黙的な要件定義が許容されている

  超上流工程での要求および受入項目は要求に合致した受入試験のみでなく、工程上では 1 レベル上流の車両レベルで実車調整も暗黙的に含まれている。これらに対応するために、本来では対応範囲でない要件定義で、暗黙的に車両レベルでの受入試験を考慮した記述が必要となっている。【図 B-6-2：図中の①】
- W モデルでは定義されていない、車両レベルでの実車調整などが存在する

  「図 B-6-1　自動車メーカー、サプライヤの関係（役割の分担）」には実車調整と記されているが､超上流工程の要求に対応する受入試験と車両レベルの実車調整作業は、現実には体系化されていない。W モデルで定義されていない作業があると、自動車メーカーとサプライヤとの役割と責任の切り分けが困難になる。【図 B-6-2：図中の②】
- 要求（要求仕様）の粒度が定義されていない

  要求（要求仕様）はサプライヤに提示されるが、各サプライヤに出すべき要求の範囲やレベルの詳細が定義されていない。このようにサプライヤに出すべき要求（要求仕様）の粒度が不揃いであるため、それぞれに対応する受入試験の検証項目も粒度を揃えることができず、品質を安定的に確保することが難しい。【図 B-6-2：図中の③】

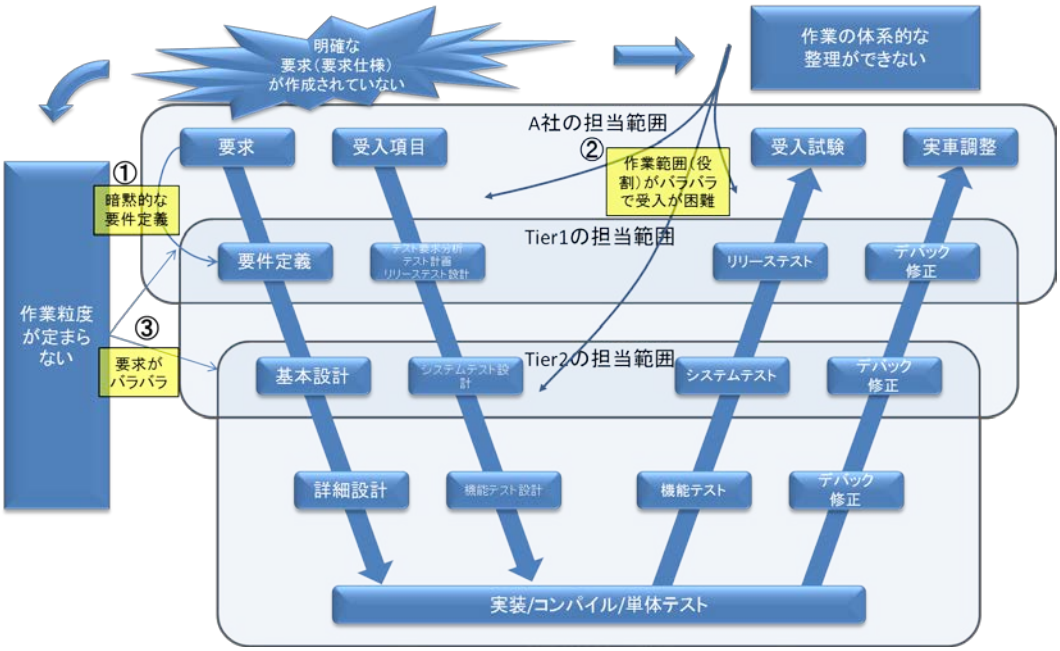


図 B-6-2　明確な要求(要求仕様)が作成されていないための作業粒度の課題

・　システムの検証など、テスト項目の設計が属人的なスキルに依存してしまう。

　システムの検証や車両関連の検証項目抽出が属人的で、経験値（技術者の知見）によって行われており、定型化された方法・技法が無い。ただし経験値は、長年かけて蓄積された内容であるため多くの漏れがあるとは考えにくいが、漏れが少ないことを確認する方法が確立されておらず、無駄な項目が存在する可能性がある。(図 B-6-3)

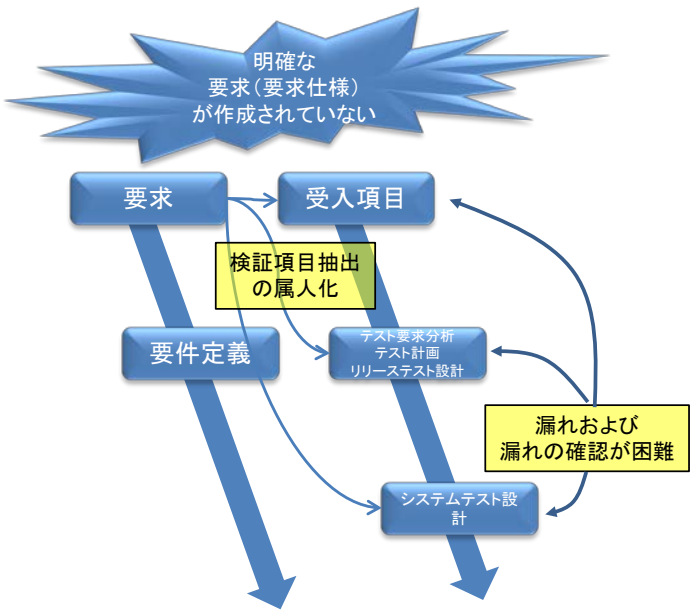


図 B-6-3　明確な要求(要求仕様)が作成されていないための検証の課題

・　追跡性（トレーサビリティ）が確保できない

　要求（要求仕様）が明確に定義されていないため、要件定義～システムの受入検証まで、W モデルに準じたトレーサビリティが確保されていない。トレーサビリティ（追跡性）が確保されていないことで、システムの開発状況把握や安全規格の認証取得の対応が困難になっている。(図 B-6-4)

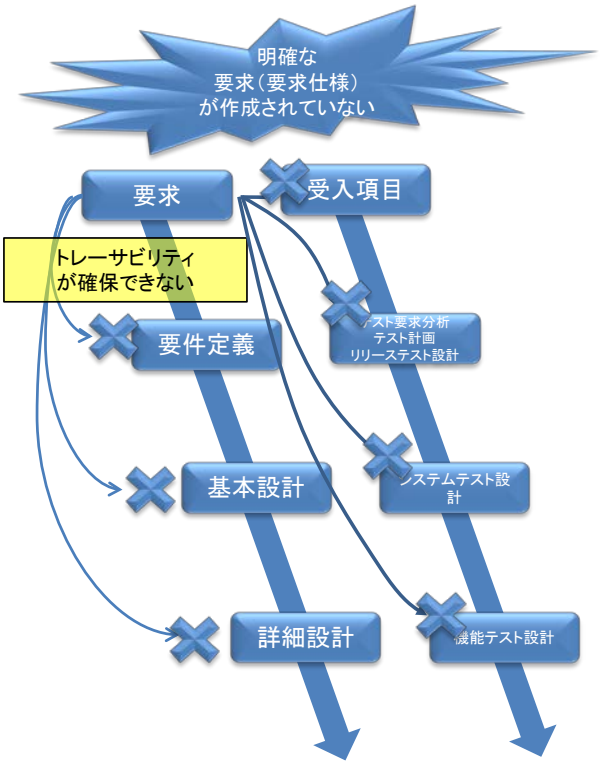


図 B-6-4　明確な要求（要求仕様）が作成されていないための追跡性の課題

前述した各々の課題は独立した課題でなく、相互に関連しており、後述の取組みを進める中で連携した課題対策が求められている。

3. 取組みの対象、適用技術・手法、評価・計測

今回の事例は、車両挙動安定化の各種ブレーキ機能の開発と検証が対象で、自動車メーカーで技術開発したものを、サプライヤに設計～製造を委託しているものである。W モデルを各プロセスで誰が担当するかという観点で分割すると、要求仕様は自動車メーカーで作成し、サプライヤが要件と機能仕様を定義して、設計～システム検証したものを納入、自動車メーカーは受入検証と実車による検証を実施している。また、ブレーキの基本制御機能は別部署が担当するなど、機能による作業分担も実施されているため、分担した役割も明確にする必要があった。本事例では、ブレーキ機能の診断機能やランプ動作、車種による諸元違いの影

響の確認を担当している。

本事例の取組みでは、下記のような技術の適用を実施した。

・ USDM 記述形式の適用によって要求仕様を文書化し明確にする

要求仕様書を USDM 記述形式により作成する。USDM は、要求仕様書の記述方式である。USDM の特徴は、次の通りである。(後述の「図 B-6-6　**要求仕様記述例**」も参照のこと)

a. 要求を数段（通常は 2～3）の階層構造で定義する

b. 定義された要求に対して、その要求が必要な理由を記述する。理由を併記することで、解釈の取り違えを抑止することができる。

c. 階層化した最下層には、要求の仕様（振る舞い）を記述する。

d. 要求および仕様（振る舞い）には、ID を割り付ける。

USDM を適用することで、要求および要求仕様に記述内容と質が統一されるとともに、ID による管理が可能となる。

要求仕様書を明確にすると同時に受入検証項目を設定することで、受入基準を明確にできる。これらにテスト設計技術を利用することで、テスト設計の特徴である網羅性を高め、要求項目の漏れの防止も行える。

・ W モデル（V&V ステップ）を適用して、開発と検証業務のフローと役割分担を明確にする

開発と検証業務のフローと役割分担を明確にするドキュメントガイドを作成する。検証要求分析～検証項目作成を手順化する。手順化には、次の内容が含まれる。

a. システムレベルや車両レベルでの検証観点を分類した

b. 検証設計のテンプレートや適用するテスト技法を整備した

上記の取組みにより、「システムの検証の属人化」「暗黙的な要件定義」「要求（要求仕様）の粒度が未定義」「車両レベルでの W モデルでの要求と検証の明確化」の課題に対応した。

顕在化している課題で対応できていない「追跡性（トレーサビリティ）の確保」は、次の仕組みを創り対応する。

・ トレーサビリティの仕組みを導入する(図 B-6-5)

市販のトレーサビリティツールを活用し、USDM で作成した要求仕様や、それに紐付く検証項目を入力して管理した。

課題に対する取組みは、単純化している。これらは、継続的に取組みを浸透させるために、複雑な処理は排除して、基本となる部分のみ対応するようにしている。基本となる課題が解決できれば、スパイラルアップで詳細な部分を進めることができると考える。

また、この取組みの課題達成に対する評価および計測は、ゴール目標を定量的な指標として設定が難しいため、取組み成果にて評価している。

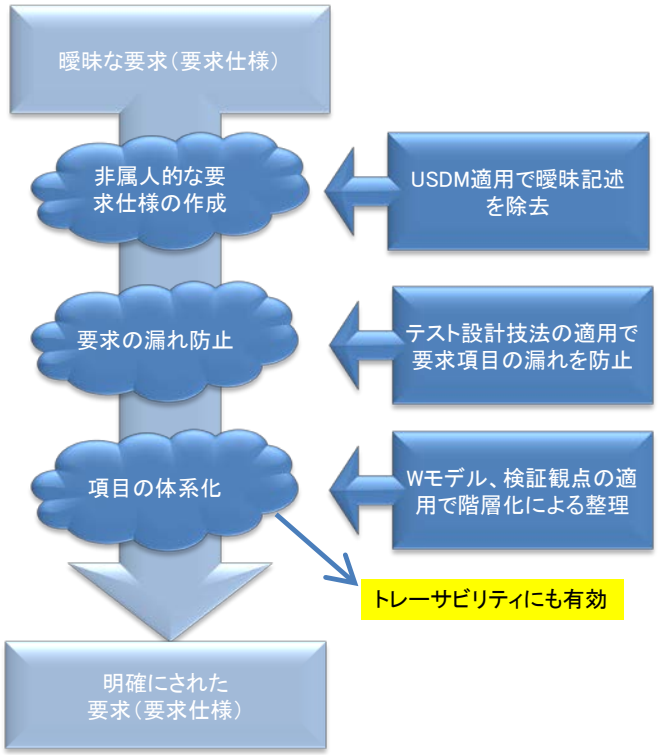


図 B-6-5　明確な要求（要求仕様）の作成のための対応

4. 取組みの実施、及び実施上の問題、対策・工夫

取組みの実施にあたり、大きく 2 つの工夫と課題が存在した。一つは、誰が作成しても同じ品質の USDM を記述するための工夫。もう一つは、USDM で記述することによる効果を確認するために、要求を熟知している技術者でなく、他の技術者が作成すること、である。それぞれの工夫と課題および解決方法を説明する。

4.1. USDM 記述のための適用ガイドラインの作成

要求（要求仕様）を USDM で記述するにあたり、今後の展開として USDM 作成を他のプロジェクトにも適用できるように、USDM 記述の品質を一定レベルに確保するためのガイドラインを作成した。最低限の USDM で記述する方法は教育する必要があるが、作業担当者に依存した記述内容にならないために、次のような USDM 記述のためのガイドラインの作成を実施した。

- ガイドラインの基本的なルール
  - EXCEL フォーマットにて記載した

(1) 本来であれば、構成管理を前提としたツールなどを利用した方が望ましく思われるが、ツールの習得期間や習得度への抵抗を考慮して、親しみやすい EXCEL

で作成する書式を採用した。

- 要求事項は USDM 記法を用いて記載した
  - USDM は要求を仕様化する手法で、下記の通りに整理を行った。
    - ✓ 要求・要求番号・理由・説明・仕様・仕様番号・仕様チェックボックスで構成する
    - ✓ 「要求」に「理由」をつける
    - ✓ 「仕様」に対して整合やトレース確認するための「チェックボックス」をつける
    - ✓ 「仕様」は「要求」に含まれる”動詞”および”目的語”で表現する（～を～する）

要求仕様記述例は、図 B-6-6 を参照のこと。

手動作動	要求	S-REQ010	（要求内容を記載） （例）AAAA機能の作動状態・故障状態に応じたランプ点灯要求をおこなう
		理由	（要求の理由について記載） （例）AAAA機能の作動状態・故障状態をドライバーに通知するため
		説明	（補足説明があれば記載）
	＜正常系＞		
	（赤）要件合意 （黄）機能仕様書確認 （青）受入試験結果 （緑）実車テスト結果	S-REQ010-N01	（要求内容をさらに仕様レベルまでに分割した要求仕様を記載） （例）AAAA機能が正常に動作している状態では、AAAA作動灯を点灯する
		S-REQ010-N02	（例）AAAA機能が正常に停止している状態では、AAAA作動灯を消灯する
	＜異常系＞		
		S-REQ010-E01	（例）AAAA機能が正常に動作中に、故障が発生したときは、AAAA作動灯を点滅する

図 B-6-6　要求仕様記述例

次に USDM で記述する要求（要求仕様）書の書式を作成し、統一された成果物が作成できるようにした。

USDM で記述する要求（要求仕様）書を次のようにガイドラインに記述した。

- ガイドラインの章構成と各章の概要
  - 表紙
    - ドキュメント名・ドキュメントバージョンについて記載
  - 改訂履歴
    - ドキュメント名・レビュー履歴・改版履歴について記載
  - 目次
    - 章番号・各章のタイトル・記載個所へのリンクについて記載

- １.はじめに
  - ✧　目的・適用範囲・用語規定・参照資料・ドキュメント内の記載ルールについて記載
- ２.機能コンセプト
  - ✧　機能概要・要求概要・ユーザ特性について記載
- ３.インターフェース
  - ✧　システム構成・各種インターフェースについて記載
- ４.機能要求
  - ✧　要求する機能詳細について記載
- ５.非機能要求
  - ✧　要求するソフトウェア品質特性について記載
- ６. 使用環境条件
  - ✧　要求する使用環境条件について記載
- ７. 開発責任の所在
  - ✧　開発の役割責任について記載
- ８.Appendix
  - ✧　ドキュメントの補足説明について記載

ガイドラインや記述ルールは、その構成自体がトレーサビリティを意識している。したがって､適切に項目を入力することで､文書自身がトレーサビリティの確保をすることとなる。また、ガイドライン自体は、あえて文書数を減らすことで読み手および適用を行う技術者の負担を軽減した。

4.2. 第３者による要求（要求仕様）書を作成するポイント

最も特徴的な取組みは要求仕様書の記述方法を USDM で記述することである。特に経験値を保有している現場の技術者でなく、第 3 者が作成していることである。第 3 者が作成することにより、USDM で記述することによる効果を確認することができる。

まず、知見や経験の少ない第 3 者が USDM で記述することができた理由は、テスト設計技術を応用したからであると考えられる。従来から独立性の高い第３者のテスト技術者がシステムテスト以降を担当することが多い。これは、開発担当者にかかる納期や品質課題などのバイアスを第３者のテスト技術者が作成することで低く抑えることができ、適切にテストを実施できるためである。また、テスト設計では網羅性が重要な項目であるため、これを活かすことで要求の漏れを防止することに貢献する。これらの特徴を活かし、第３者のテスト技術者による要求（要求仕様）の作成を実施した。これらは、テスト設計技術が要求仕様などの文書化をするために適していることを示した。作成された仕様の妥当性の確認作業が、並行して行われている。これらのテスト設計技術の工夫の一部を紹介する。

今回は、多くのテストのプロジェクトで利用されるテスト設計技術である分類ツリー法（classification tree method）を利用して、要求仕様を層別構造で仕様書の作成を実施している。第 3 者のテスト技術者が要求仕様書を作成するためには、これらのテスト設計技術を応用することが適している。テスト設計では、要求が“振る舞い＝機能”であることに着眼し、抽象的な振る舞い記述を分解不可能な階層まで展開することで、明確な要求項目として抽出することが可能になる。（図 B-6-7）

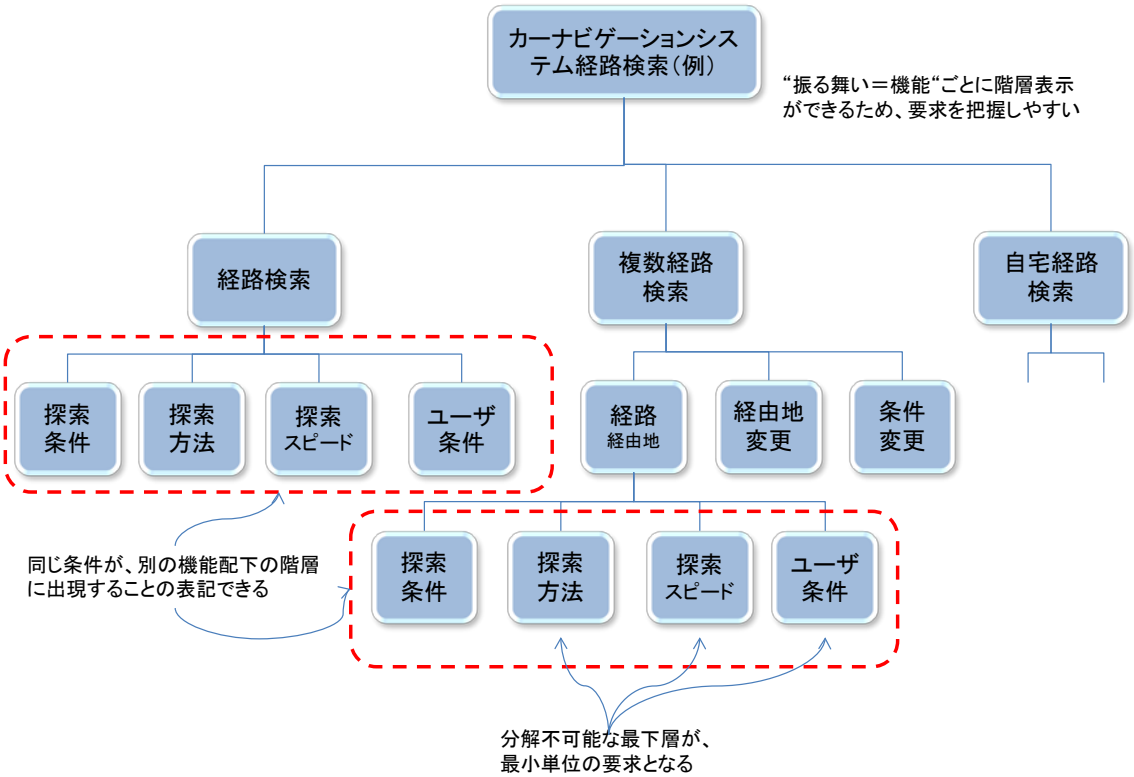


図 B-6-7　参考－分類ツリー法（classification tree method）

ここの要求（要求仕様）を明確にすることは、USDM の記述方式で担保されるべきであるが、要求仕様を作成しようとしたとき、要求を層別に分解しながら機能からサブ機能に落とし込まれていく。

これらの分解技法は明確に定義されていないことが多い。定義されていない作業は、属人的になる。この状態では、USDM での記述が正しいとしても、実装されたソフトウェアがこれを満足するシステムであるか、受入試験が困難になる。したがって USDM で記述する各要求（要求仕様）も統一した階層構造で管理されるべきである。

分類ツリー法を用いたクラシフィケーション木構造で分解すると、層別のグループ化がされるのと同時に記述された要求の関係が明示できる。独立した構造の要求であっても、属性が関連する他の要求との強弱（因果）関係を示すことができ、次工程以降の設計での観点（レ

ビュー、注意点）になる。特に、他の組織（サプライヤ）に引き渡す場合の制約・前提事項のカバレージ（網羅性）やトレーサビリティにも有効となる。

その他にも要求（要求仕様）にテスト技術を利用するメリットがある。一般的に開発仕様が“振る舞い＝機能”を実現するための処理方法を記述するのに対し、テスト設計では処理に対して入出力、特に期待値に着目してテスト仕様を記述する。これらの期待値を定義する時に設計仕様書から読み取れない場合は、経験的に処理条件などの考慮が漏れている場合が多い。また、テスト設計技術では、テストで確認する項目とテスト対象に与える条件に分類して整理する。テストで確認する項目は対象となるパラメータや要素を示し、テスト対象に与える条件はその値となる。

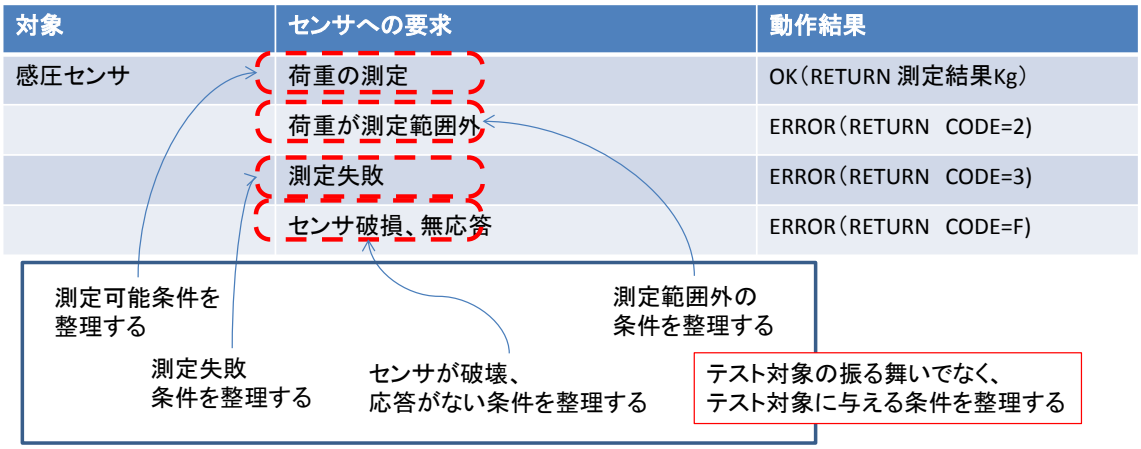


図 B-6-8　テスト条件の整理

例えば、テスト対象が感圧センサの場合、測定する対象となる荷重と測定できるデータやその範囲などの数値に分類できる。これらの分類によって感圧センサを構成する要素が整理できる。（図 B-6-8）また、追加や変更の要求があった場合に、測定する事象の変更か、与えるデータなどの変更かによって影響範囲を特定し易くすることにも役立てることができる。

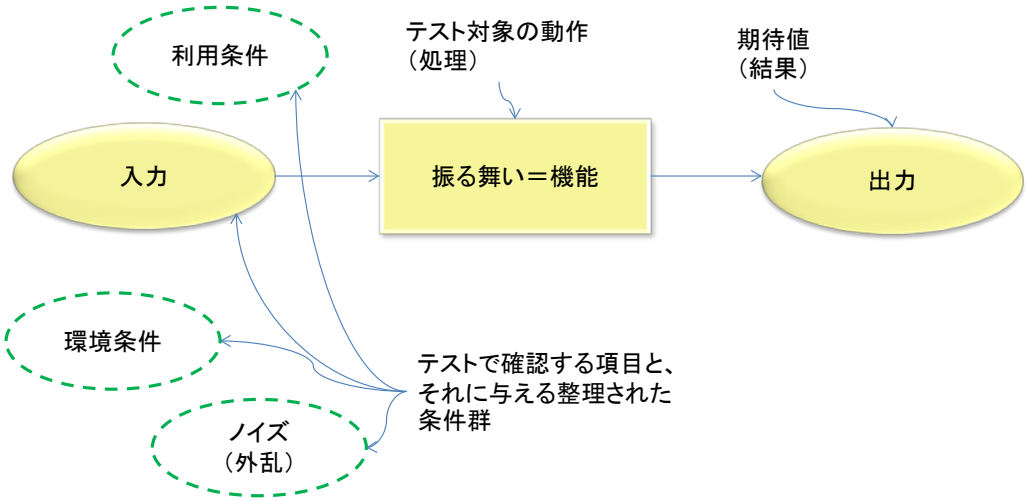


図 B-6-9　テスト項目の分類

このような取組みでは、既存のプロセスに対して文書作成の業務が増加し、現業の技術者に負担増を強いることが多くなり、現場の技術者に受け入れられることが難しい。また、スキルを保有している技術者は、暗黙的な要求を記述することは難しい。これらはプロセス改善などの取組みに似ている。この取組みでは、プロジェクトに影響を与えないように現業の技術者に負担をかけないことを要求されている。そのために、この取組み事例では要求仕様の整理と作成を第３者のテスト技術者が実施していることも特徴となっている。第３者であるテスト技術者が既存技術をまとめ、要求（要求仕様）を作成することも困難と考えられていた。しかしテスト技術者は、要求（要求仕様）や設計仕様からテスト設計を実施する過程で仕様に不明点などがある場合は、設計者に対して質問票を取り交わすなど、適切な処理を行い確認を行う。これらのコミュニケーション能力が長けている技術者が多い。テスト技術者のコミュニケーション能力の例としては、不具合のステータス管理などがあげられる。テスト技術者が不具合を発見したときは、事象や発生手順および環境条件などを開発者にレポートして、適切に修正または状況が完了するまで対応状況を管理する。このときに開発者とは、仕様の課題や納期、技術的な問題など、随時情報を交換し、発見した不具合を対応完了の状態まで持っていく。このような開発者とのコミュニケーションは、テスト技術者の重要なスキルである。不具合を「不明な要求（要求仕様）」に置き換えることが可能であったため、第３者であっても要求（要求仕様）の作成が可能であったと考えられる。

5. 達成度の評価、取組みの結果

この取組みによって要求（要求仕様）を明確にするだけでなく、要求項目を体系的に整理することで、要求項目間の関連性や要求と要求仕様および受入試験とのトレーサビリティが確保できることが分かった。これらは、要求に対する仕様変更に対しての影響範囲や変更リ

スクの特定などにも有効に利用できると考える。また記述すべき項目と内容についてのガイドラインを作成することで、広く利用が可能になると考える。特に、要求仕様を把握している技術者でなく第 3 者が要求仕様の作成を担当していることで、暗黙的な技術が明確にできるようになり、また要求（要求仕様）が体系的に整理することに貢献できている。要求（要求仕様）の体系的な整理は、層別に分類することで階層的に整理され、階層構造や属性分類を実施することで詳細化した仕様間の関連性も明確にすることができ、仕様変更や派生開発時に影響分析を行うのに有効となる。

取組み成果を整理すると、下記の課題がほぼ達成できた。

- USDM を利用した要求（要求仕様）を記述するためのガイドラインを作成することで、安定した品質で作成が可能となった。
- 要求に対応する受入基準が明確になり、受入試験の項目作成で品質向上が図れた。
- テスト技術を利用して体系的に整理することで、トレーサビリティなど影響分析に利用が可能となった。
- W モデルでの担当する役割分担を明確にすることができた。
- 第 3 者による実施により、既存の技術者に少ない負担で行えた。

ただし、これらは小規模な範囲での試験的な取組みであり、適用範囲を拡大して利用するためには更なる試みや課題への対応が必要と考える。

6. 今後の取組みと考察

この取組みは、自動車メーカーの A 社のみで完結できない。製造・開発を担当する Tier1 グループおよび Tier2 グループも同様な処理ができていなければ意味をなさない。従って、この取組み自身を手順化して、製造・開発の全ての担当へ展開する必要がある。

今後の取組みには下記のような課題が存在する。

- IEEE 830 （ IEEE Recommended Practice for Software Requirements Specifications）の活用

  対外的に利用を図るためにソフトウェア要求仕様の定義に IEEE 830 の活用を検討しているが、IEEE830 は概念の定義が多く、利用するには作成テンプレートやガイドラインなどの作成要項を整備する必要がある
- 要求～要求仕様の仕組み化の実施

  超上流領域の要求から要求仕様の作成は、USDM の利用やガイドラインの作成することで課題に対して達成したが、W モデルでの作業標準との対応ができていない。今後は、仕組みとして作業フレームへの展開をする必要がある。
- これらの仕組みを自動車メーカーだけでなく、サプライヤへの展開

  本取組みでは、自動車メーカーだけでなく、サプライヤとの関係にも影響する。これらに対応し、まずは Tier1 グループに対して、同技術を展開する必要がある。

参考文献

[1]「派生開発」を成功させるプロセス改善の技術と極意、技術評論社、2007、清水吉男氏

B－7

形式手法を用いたセキュリティ検証 [1]

1. 概要

近年のシステム・ソフトウェア開発における、セキュリティ要件の増大とその保証を確立しなければならない時代において、従来手法であるレビューだけでは十分な保証が難しくなってきている。セキュリティ分野においても高品質、高信頼化が求められるようになり、セキュリティが担保されていることを保証するための客観的な方法として形式手法に注目が集まってきた現状がある。情報セキュリティを評価するための国際規格であるコモンクライテリア(ISO/IEC 15408)等においても、高レベルのセキュリティ保証として形式手法の適用が求められている。

本事例は、『平成 24 年度戦略的基盤技術高度化支援事業　－形式手法を活用した組込みセキュリティ技術の確立と、安全・安心な CPS 社会を支える無線通信ミドルウェアの開発－』事業において、組込み機器におけるセキュリティの高品質、高信頼化へ貢献するため、形式手法を用いてセキュリティ機能の有効性を検証した事例である。

選択した形式手法は Event-B(Rodin プラットフォーム)であり、集合論を用いた定理証明やモデル検査が可能となっている[1]。

2. 取組みの目的

2.1. 背景

昨今、スマートハウスや車載システムなどがネットワーク連携を行い、実社会と IT との融合が目覚ましく発展してきている。CPS(Cyber-Physical System)社会への関心と普及が高まる中、組込み機器の役割も変化してきている。今まで組込み機器は単独で動作することが前提となっており、ネットワーク連携におけるセキュリティ対策が施されてこなかったが、今後、組込み機器がネットワーク連携していくことが予想され、組込み機器が処理する情報や組込み機器自体にもセキュリティ対策を施す必要性が出てきた。

2.2. 解決すべき課題

このような社会環境の中で、組込み機器を介したソフトウェアのセキュリティ検証上、以下のような課題がある。セキュリティ技術への関心が高まっていくことに対して、情報セキュリティに関する情報は十分に展開されているが、組込み機器のネットワーク連携における利用形態や、保護すべき資産、リスク評価など、組込み機器特有のセキュリティ要件に関し

[1] 事例提供：アーク・システム・ソリューションズ株式会社 研究開発チーム　池田 和博 氏

ては情報が少ないのが現状である。そのため、組込み機器が満たすべきセキュリティ要件を導出するとともに、従来手法であるレビューや脅威分析以外の上流工程における分析・検証手段を活用した取組みを強力に推進していく必要がある。

2.3. 目標

組込み機器上でセキュリティが担保されていることを保証するために、最近注目を集めている客観的な方法の形式手法が有効であることを具体的に検証することを目標とする。

検証の観点は以下の通りである。

- セキュリティ機能やセキュリティ脅威を形式記述することで、要求仕様に記述された自然言語による曖昧性を排除できること
- 数学的な証明によって客観的な観点からセキュリティの品質を保証できること

特に、組込み機器におけるセキュリティ特性の観点から検証に必要な項目の抽出や、検証観点の明確化に主眼を置いた。

3. 取組みの対象と適用技術・手法

形式手法（今回利用する Event-B）を適用する時のセキュリティ機能やセキュリティ脅威の定義や定理証明による検証方法の体系化を整理し、開発現場での適用方法を考察する。

3.1. 対象製品・プロジェクト、適用工程

本検証は、宅内における組込み機器が無線ネットワーク連携を制御する無線通信ミドルウェアの開発事業において実施した。無線通信ミドルウェアは宅内機器と宅内に設置するゲートウェイに組み込まれ、宅内ネットワークへのアクセスコントロールを行う。宅内機器はネットワーク参加の認証をゲートウェイに対して行い、ゲートウェイに認証されることでネットワークへ参加できる。同じ宅内機器間の通信は必ずゲートウェイを介して通信することでアクセスコントロールを行う仕様である。（図 B-7-1）

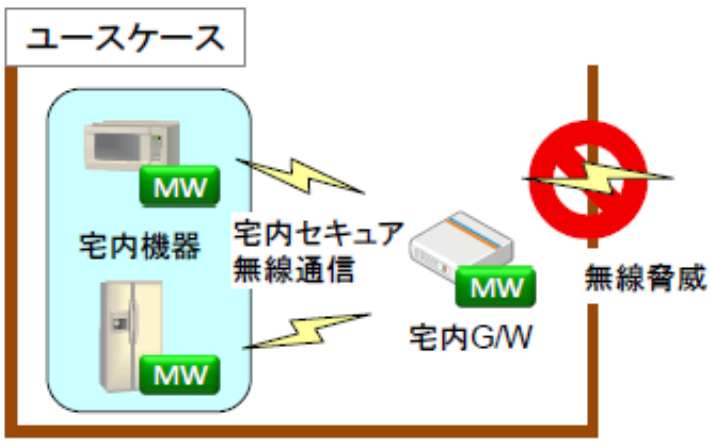


図 B-7-1　検証環境

評価対象（TOE：Target of Evaluation）はミドルウェア及びそれを搭載する端末とアクセスポイントを含むネットワーク環境であり、開発対象（TOD：Target of Development）はミドルウェアである。（図 B-7-2、図 B-7-3）

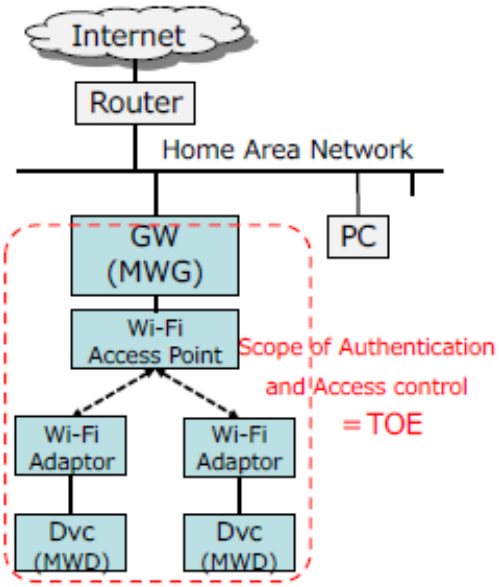


図 B-7-2　ネットワーク構成

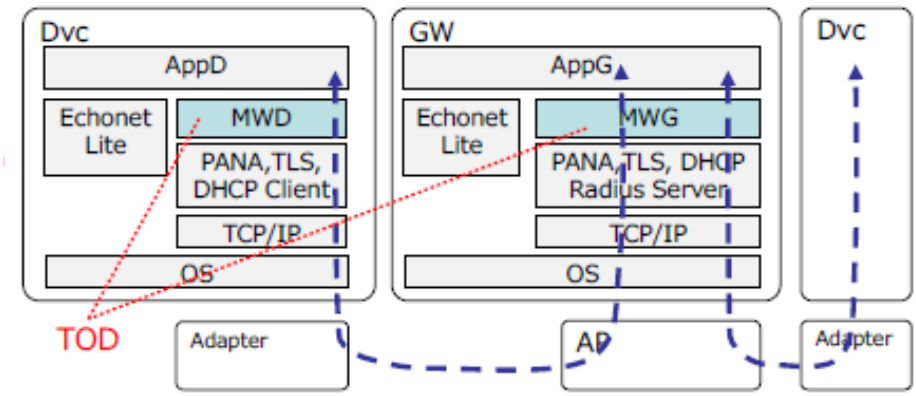


図 B-7-3　ソフトウェア構成

無線通信ミドルウェアのセキュリティ機能が記載された要求仕様やセキュリティ要件をインプット材料として、開発の上流工程において検証方法（Event-B）を用いて形式記述(モデル化)を行い、セキュリティ機能の有効性を証明によって保証する、もしくは脆弱性を検出する、ということを実施した。脆弱性が検出された場合には、脆弱性情報を要求仕様へフィードバックし、セキュリティ要件及び要求仕様の修正を行った。

3.2. 検証手法（Event-B）

形式記述言語である Event-B は、集合論を用いた定理証明により検証を行う手法である。元は B-Method から派生した形式言語であり、上流工程での検証に向いている[2]。これは、Event-B が高い表現力を持ち、具体的には水平リファインメントを用いて抽象的な表現から段階的に詳細化を行うことができるためである。

水平リファインメントとは上位モデルとの整合性を検証しながら、リファインメントにより機能の追加・上書き、または分割などを行うことができるため、要求仕様を段階的に表現していく方法として有効である。

Event-B を記述するためのツールとして、Rodin プラットフォーム（以下 Rodin と略）を用いることを選択した。Rodin は無償で提供されており、Eclipse ベースの Event-B 記述に特化したツールであり、証明器による証明課題の自動生成及び自動証明が可能なツールである。また、そのツールのプラグイン機能（ProB）を利用することにより、Event-B モデルの可視化やモデル検査を行うことも可能である。

3.3. 検証の観点

要求仕様を形式仕様で記述する際に、定理証明が完了することが必ずしもセキュリティ機能を保証していることにはつながらないため、そのモデルが何を検証しているのかを明確化する必要がある。そのため、無線通信ミドルウェアの保護すべき資産とセキュリティ特性を抽出し、特性ごとに記述・検証を行った。

Event-B を用いたセキュリティ検証の観点として、無線通信ミドルウェアの要求仕様を抽象化し、形式記述を行う。その抽象モデルに水平リファインメントを用いて仕様の拡張や環境要因を追加し、段階的に詳細化していくことで仕様やセキュリティ機能をモデル化していく。そして、モデル化された仕様に対してセキュリティ脅威となる攻撃モデルを導入することにより、セキュリティ機能が有効に機能しているかを定理証明により検証する。

導入する攻撃モデルは、通信ネットワークの暗号プロトコルを分析する際に一般的に使われている Dolev-Yao モデルを参考に、攻撃モデルを作成する。Dolev-Yao モデルは以下が前提となる。

- 暗号系は完全に安全な状態である（絶対に破られることがない）
- 攻撃者は通信をフルコントロールしている（メッセージの傍受・削除や生成ができる）

また、セキュリティ機能がどの程度の攻撃に対してまで有効に機能しているかを、無線通信ミドルウェアに対して行う攻撃者の能力を水平リファインメントによって弱い状態から

徐々に攻撃能力を最大限に表現していくことで、セキュリティレベルを明確化した。

4. 取組みの実施、及び実施上の問題、対策・工夫

開発チームの体制は、無線通信ミドルウェアの要求仕様を作成するチーム（要求仕様チーム）と、その要求仕様を検証するチーム（検証チーム）に分かれて作業を実施した。

4.1. セキュリティ要件の導出

要求仕様チームは、宅内で利用される組込み機器が無線ネットワーク連携したことを想定して、組込み機器特有のセキュリティ脅威及びセキュリティ特性を導出した。情報セキュリティにおける特性の観点から機密性、完全性、可用性の 3 つ。さらに、組込み機器の場合、利用者と組込み機器の認証を考慮する必要があり、利用者によるサービス利用の否認も考慮する必要があることから、認証と否認防止の 2 つを挙げた。

（1） 機密性(Confidentiality)：情報へのアクセスを認められた者だけが、その情報にアクセスできることを確保する。

（2） 完全性(Integrity)：情報が破壊、改ざん、消去されていない状態を確保する。

（3） 可用性(Availability)：情報へのアクセスを認められた者だけが、必要時に中断することなく、情報及び関連資産にアクセスできる状態を確保する。

（4） 認証(Authentication)：利用者の身元が保証されている。利用を許可された機器以外は利用することができない。

（5） 否認防止(Non-repudiation)：利用者はアクションを実行した後、それを実行したことを否認できない。

そして、無線ネットワーク連携を行う組込み機器が保護すべき資産、セキュリティ特性、セキュリティ脅威の観点から、以下のセキュリティ要件を導出した。（表 B-7-1）

表 B-7-1　開発対象のセキュリティ要件

(CC：Common Criteria(ISO/IEC15408)のセキュリティ機能要件の分類)

資産	C.I.A.N.A	脅威	対策方針	CC
通信データ	機密性	盗聴	通信データの暗号化	FCS
		なりすまし	通信相手の特定	FDP
		不正アクセス	認証機能、アクセス制御	FIA
		機器情報の読取	情報削除機能	FDP
	可用性	ただのり	認証機能	FIA,FDP
		DoS攻撃	IPS機能	FRU,FTA,FTP
	完全性	偽中継局	認証機能、電子証明書	FIA,FDP
	否認防止	サービス利用否認	ログ機能	FAU,FCO
鍵情報	機密性	解析	再生成・無効化、強度	FCS
	認証	漏洩	ライフタイムの制限	FIA

上記のプロセスを踏まえて、開発対象システムである無線通信ミドルウェアが満たすべきセキュリティ要件及びセキュリティ脅威への対策方針を抽出し、以下のような要求仕様を作成した。無線通信ミドルウェアは宅内ネットワークへ参加するための認証機能とアクセスコントロール機能を提供する。

【要求仕様：無線通信ミドルウェアの主なセキュリティ機能】

- 正規の宅内機器、ゲートウェイのみが宅内ネットワークに参加できる。
- 不正なゲートウェイによる宅内機器の操作や、不正な機器が宅内ネットワークへ参加できないようにする。

4.2. セキュリティ検証の実施

検証チームは、要求仕様及びセキュリティ要件を元に、Event-B を用いて無線通信ミドルウェアのセキュリティ機能が有効に機能しているか検証した。

Event-B を用いて検証する際、以下の手順で実施した。（図 B-7-4）

（1） Event-B 適用の前提の定義

a. 検証の対象範囲、検証レベル(目標)の設定

b. システムの仕様・環境を含めたシステム全体の振舞いの想定

c. 保護資産と保証(証明)すべきセキュリティプロパティの定義

（2） Event-B のモデリング実施

a. 仕様・セキュリティ機能の抽象化(モデリング)

b. リファインメントによるシステムの仕様・環境の表現

c. セキュリティ脅威のモデル化(攻撃者モデル)

（3） Event-B モデルの妥当性確認

a. Event-B モデルの記述・式の確認

b. 要求仕様・振舞いの想定と Event-B モデルの整合性の確認

（4） 定理証明及びモデル検査によるセキュリティ検証

a. ツールによる自動証明と手動証明

b. ツールで証明できなかった未証明課題の検証（モデル検査）

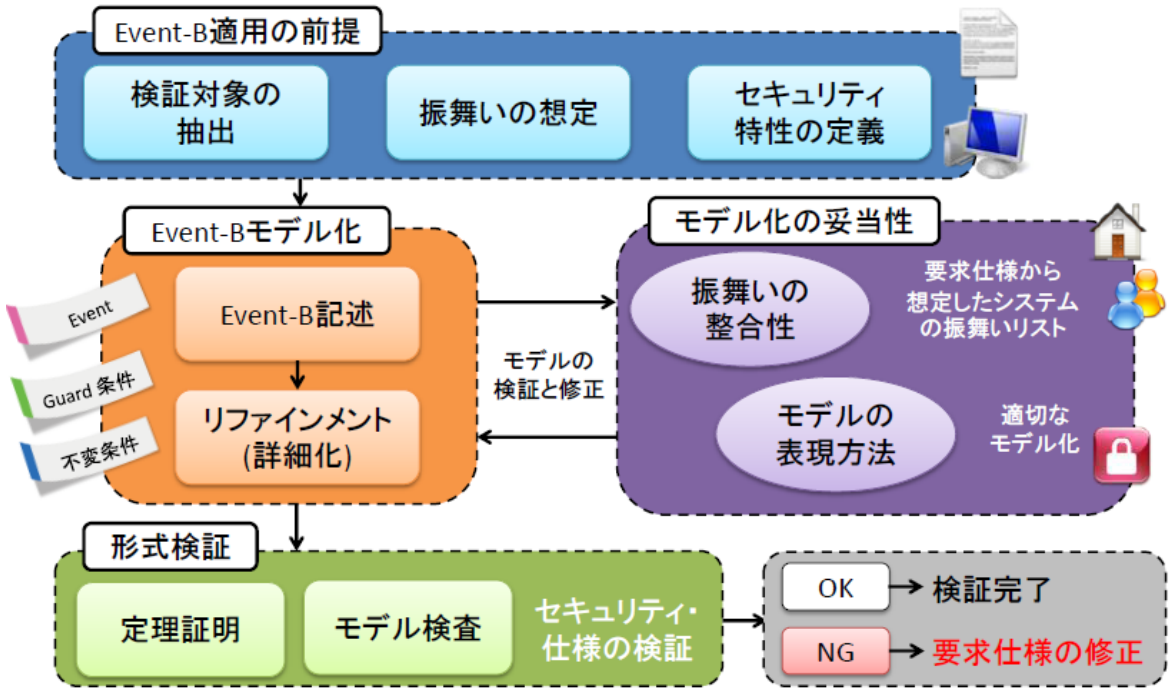


図 B-7-4　Event-B の適用関係

4.2.1. Event-B 適用の前提の定義

（1）検証の対象、検証レベル(目標)の設定

本検証では対象を､運用フェーズにおける通信プロトコルのセキュリティ機能とした。セキュリティ要件を分析・検証する観点では、運用フェーズ以外の開発フェーズや廃棄フェーズも検証の対象となるが、要求仕様の検証も含め、運用時におけるユーザー操作やシステム挙動がユーザーやシステムに与える影響を検証することに的を絞った。

検証対象のシステムにおいて、セキュリティ要件として検証すべき項目を抽出し、優先順位をつけた。対象システムが保証すべき項目に的を絞った形でモデリングすることとした。

コモンクライテリア(ISO/IEC 15408)[3]では評価保証レベルに応じて形式手法の適用方法も変わるので、実開発においては顧客や開発現場からの要求があり、それに応じた目標の設定を行う必要がある。従って、検証対象は目標とする検証レベルに応じて変更され、優先度の高い項目から検証していくことが重要である。

適用した形式手法（Event-B）は再利用性が高く、すでに証明したことを前提として新たなモデルに適用することが可能であり、順次モデリング規模を大きくしていくことも可能であることから、より対象を絞り込んで小さなモデルから始めて、徐々に詳細化や追加などを行う方がモデル自体の妥当性確認を行う上でも効率の良いことが分かった。

（2）システムの仕様・環境を含めたシステム全体の振舞いの想定

この段階で作成するシステム振舞いの想定は要求仕様から想定できるすべてを抽出する。本検証では、システムがある処理を行った場合に、次にどのような状態へと遷移するか、またはどう処理すべきかを抽出した。また、処理だけでなく、対象システムを構成する環境も想定し、無線通信ミドルウェアが搭載される宅内機器・ゲートウェイを操作するユーザー、ネットワーク上の通信データもシステムを構成する要素として捉えて抽出した。

Event-B による形式仕様記述を行い、他者がレビューする際に、モデルの内容が何を表現しているかわからないことが必ず問題となった。Event-B は高い表現力を持つことから、記述者は自分の考えを表現しているつもりでも記述者によって表現方法に差異が生じるため、他者は記述内容を一見しただけでは理解できない場合が多かった。また、Event-B モデル全体で仕様やセキュリティ要件をどの程度表しているかの妥当性を確認する際にも、必要な情報が要求仕様だけでは過不足があるのか判断できなかった。

これに対して、Event-B モデルの妥当性を確認するために、要求仕様からシステムがどのような振舞いとなるかを想定することが重要であるとわかった。

(3) 保護資産と保証(証明)すべきセキュリティプロパティの定義

Event-B による形式仕様記述を実施する前に、自然言語によって保護資産とセキュリティ特性の関係を明確にした。セキュリティ特性は要求仕様チームが導出した 5 つの必須項目(機密性・完全性・可用性・認証・否認防止)を用いて、無線通信ミドルウェアの保護資産とセキュリティ特性ごとに保証したい要件を「セキュリティプロパティ」として定義した。セキュリティプロパティは Event-B の Invariant(不変条件)に記述する対象であり、記述したモデルによって Invariant を違反していないか証明が求められる。その証明が完了することでセキュリティプロパティが保証されていることを示す。

さらに、セキュリティ脅威によって定義したセキュリティプロパティが破られるかどうかを形式検証することで、脆弱性の検出やセキュリティ機能の有効性を保証することにつながるので、無線通信ミドルウェアの要求仕様から保護資産に対しての攻撃インタフェース／攻撃アクションを想定した。(表 B-7-2) そして、要求仕様から作成したセキュリティプロパティと比較して、定義内容の修正・追加を行った。

表 B-7-2　保護資産に対するセキュリティ脅威の一覧

保護資産	セキュリティ特性	攻撃エージェント	攻撃インタフェース／攻撃アクション
宅内機器、GWのネットワーク利用(通信機能)	認証	第3者	正規の機器・GW以外の端末を通信に参加させる。
利用者またはメーカー、サービス提供利用者の秘密情報	機密性	第3者	宅内機器とGWの間でやり取りされる通信データを盗聴する。
機器の機能(サービス利用)を満たすために必要な情報	完全性	第3者	宅内機器とGWの間でやり取りされる通信データを偽造、または改ざんする。
機器内部の保存情報	機密性	第3者	機器に格納された情報を読み取る。

4.2.2. Event-B のモデリング実施

(1) 仕様・セキュリティ機能の抽象化(モデリング)

Event-B でセキュリティ検証を行う場合、初期モデルはセキュリティが保証されている最善の形を定義することが重要である。この初期モデルが正しくモデル化できていなければ、以降のリファインメントモデルにおける検証も意味がない。初期モデルで検証したことを詳細化による機能拡張、セキュリティ脅威の導入によっても証明することができれば初期モデルで表現したセキュリティプロパティを保証することができていると言える。つまり、システムや構成する環境を含めたシステムの本質的な性質やセキュリティ機能の正しさを抜き出す必要がある。

保証したいセキュリティプロパティは「4.2.1Event-B 適用の前提の定義 (3)」で定義した内容を Invariant(不変条件)に記載し、Event-B モデルとして要求仕様やシステムの振舞いの観点から、セキュリティプロパティが保証される形での最善となるモデルを初期モデルとして記述した。

しかし、実際には Event-B に限らず形式手法における抽象化は非常に重要であると同時に難しいものであった。抽象化と言うにはモデルの構成要素が過剰となることや、システムの特性を表していないものなど、抽象的な表現ができていなかった。そのため、要求仕様から検証するセキュリティ機能のすべてを抽象化するのではなく、セキュリティ機能の関連性や順位付けを行い、簡潔なモデルを作成することから始めた。

また、検証観点であるセキュリティプロパティは、論理式を使用した「○○は△△である/ではない」といった特定の条件下において論理が正しいか否かを明確にする式に置き換えた。そのため、保護資産は変数（○○）として表現し、セキュリティプロパティから変数が満たすべき状態変化（△△）を Invariant に定義した。

(2) リファインメントによるシステムの仕様・環境の表現

簡潔に表現した初期モデルの抽象的な表現から、より仕様に近い形にモデル化するためリファインメントによる詳細化を行った。

Event-B の特徴であるリファインメントによるイベントの追加や条件の強化を行うことで、初期モデルとの整合性を検証できる。そのため、初期モデルで定義する変数の関係性は抽象表現であることも踏まえて、なるべく包括的な表現となるように定義し、イベントのガード条件にて変数の制御を行うようにモデル化した。それにより、リファインメントしたイベントのガード条件を強化することで、より表現したい仕様に近い形にモデリングしていくことができた。

詳細化によって初期モデルとの整合性がないことにより証明が完了しない場合、モデルの記述に問題があるか、仕様に問題があるかのどちらかとなる。この判断については後述する「4.2.3Event-B モデルの妥当性確認」にて言及する。

(3) セキュリティ脅威のモデル化

想定した攻撃インタフェースや攻撃アクションをリファインメントにより導入した。

セキュリティ検証の観点として、攻撃モデルにより初期モデルで定義した Invariant に影響があるかを検証することが重要な要素である。(図 B-7-5)

本検証では、攻撃モデルとして Dolev-Yao モデルを採用し、攻撃者が通信をフルコントロール状態している前提で攻撃アクションをモデル化した。

攻撃モデルはリファインメントを利用して攻撃者の能力を段階的な強さで表現した。リファインメントの際に、攻撃者の能力を強化することによって脆弱性が検出された場合、どの程度の攻撃には耐えられるかを判断することができる。

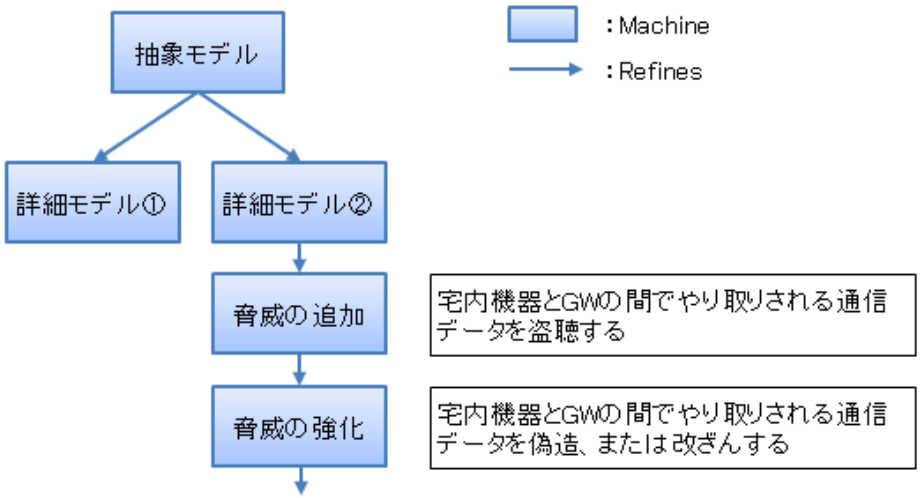


図 B-7-5　リファインメントによる攻撃モデルの導入例

4.2.3. Event-B モデルの妥当性確認

(1) Event-B モデルの記述・式の確認

Event-B は集合論を基礎として、関係を用いて表現することが一般的である。モデルの妥当性確認の際、特に気を付けた点は、仕様やセキュリティ機能を表現している変数の関係（関数）が想定していた制約や条件を正しく表現しているかを確認することである。なぜならば、変数の関係性が正しく定義できていなければ、セキュリティプロパティの証明が想定とは異なる結果になるからである。

目視以外の確認方法として、Rodin のプラグインツールである ProB を使用した。ProB は Event-B モデルのアニメーション実行が可能であり、定義したイベントの実行によって変数がどのように変化していくかを確認することができる。それにより変数が想定した状態変化を行っているか各イベントの実行前と実行後の変数の値を確認した。また、定理証明が完了しない箇所について、モデルの記述に問題があるか、仕様に問題があるかの判断基準にも利用した。モデルの記述に問題がある場合、概ねイベントの実行が不可能な状態になる(デッドロックの発生など)。それ以外にも、変数の定義範囲を超えるような状態になる場合もあり、変数の定義内容が問題であることもアニメーション実行

により判明した。仕様に問題がある場合には、セキュリティプロパティを違反することになり、セキュリティ機能の脆弱性を検出したことになる。

(2) 要求仕様・振舞いの想定と Event-B モデルの整合性の確認

システムの処理や環境を含めたシステムの振舞いが正しくモデル化されていることを確認するのは上述した通り重要である。そのため、Event-B モデルが要求仕様や振舞いを表現できているか確認する方法は「4.2.1Event-B 適用の前提の定義 (2)」で定義した振舞いの想定である。その想定と Event-B モデルで整合性の確認を行った。

本検証では、振舞いを定義する際に各項目を ID で管理し、Event-B モデル作成時にコメント部分に同じ ID を付与することで、他の者がレビューし易いよう記述した。同様に、セキュリティプロパティの定義も ID で管理し、イベントのガード条件やアクションだけでなく Invariant も対比できるようにした。これによって、Event-B で仕様をどのように表現したかを自然言語で認識できることから、記述したモデルが要求仕様や想定された振舞いを表現できているか確認する上でも役に立った。

モデルの妥当性確認は Event-B モデルの記述時に同時進行で行った。Event-B は MACHINE と CONTEXT から構成されており、振舞いを表現する MACHINE の記述後に妥当性を確認し、問題があれば適時修正を行い再度確認する手順とした。

1 つの MACHINE ごとに実施したのは、抽象モデルが検証したいことを正しく表現できていない場合にモデルとしての意味がないので、MACHINE ごとに確認する必要があるためである。また、Rodin ツールで記述したモデルがリファインメントされている場合、抽象モデルの修正内容が詳細モデルに自動反映されないという問題もある。リファインメント後に問題が発覚した場合、抽象モデルと詳細モデルの両方を修正しなくてはならないため、作業の効率化を図るためリファインメント前に抽象モデルの確認を行うことで、手戻りを軽減することができた。

4.2.4. 定理証明及びモデル検査によるセキュリティ検証

(1) ツールによる自動証明と手動証明

Rodin ツールによって自動生成された証明課題は、ツールによる自動証明が実施され、自動で証明できなかった証明課題は手動で証明を行う必要がある。基本的にはツールの証明性能によって大半の証明課題は自動的に証明が完了した。

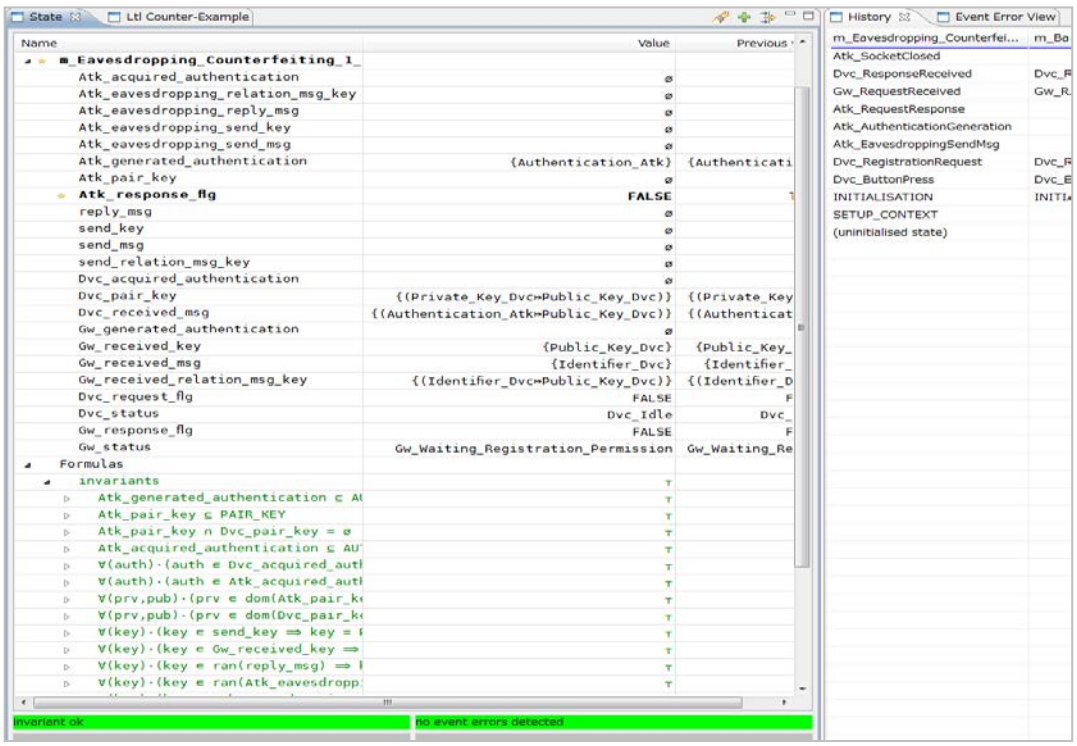
証明が完了しない項目は対話証明（手動による証明）により、証明仮定を与えて証明が完了するように促す。証明仮定を導出するためにツールを用いて証明課題の分割を行い、証明できる部分と証明できない部分に分離して、何が証明できていないかを判断した。証明できない部分に対して、変数がどのような状態変化を行うかを Invariant に追加して証明を完了へと導いた。これは Technical Invariant と呼ばれ、Invariant に記述したことで証明課題は増加するが、その証明課題が完了すると、それを他の証明の仮定として利用することができる。上記を実施しても未証明の項目が残る場合がある。それ

は、セキュリティ機能の脆弱性の可能性があるため、モデル検査によるセキュリティ検証を実施した。

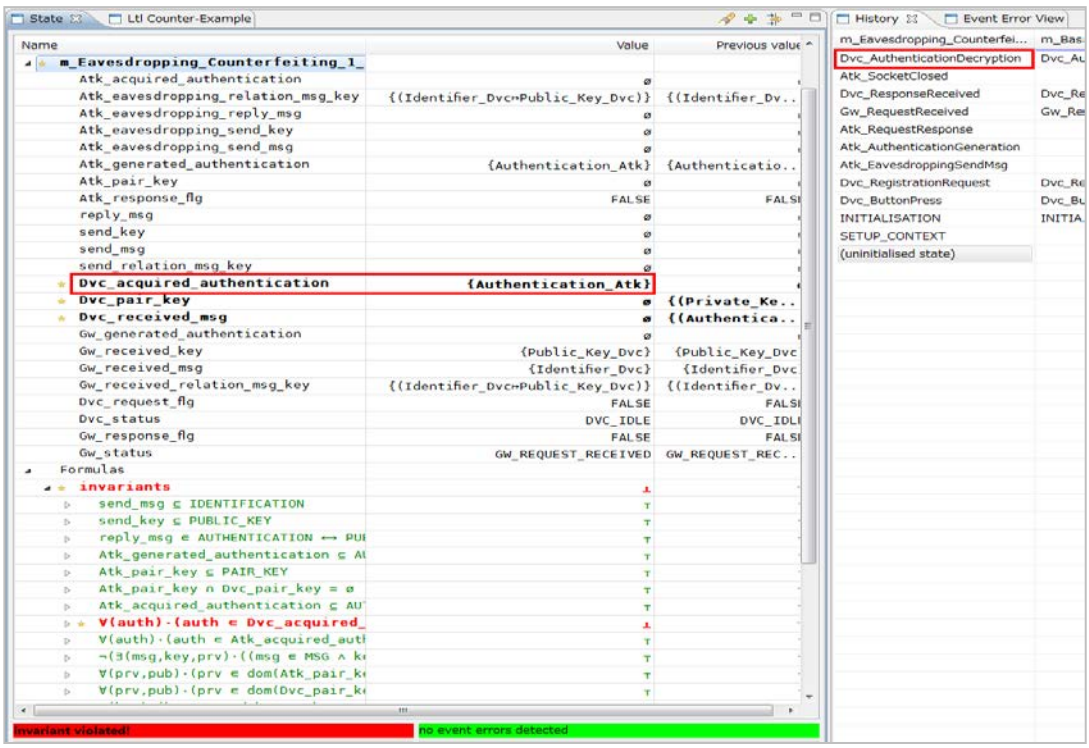
(2) ツールで証明できなかった未証明課題の検証（モデル検査）

未証明の項目について、Rodin の ProB 機能の Model Checking 機能を利用して検証を実施した。Model Checking はイベントを自動実行することにより、Invariant 違反がないかを確認する機能である[4]。実行の結果、Invariant 違反を検出した場合、イベントのヒストリーが表示され、どの手順でイベントを実行したときに Invariant 違反となるかを確認することができる。(図 B-7-6)

本検証では、Model Checking 機能によって未証明の項目が、定義したセキュリティプロパティを違反していることが判明した。また、その際のイベントのヒストリーを確認し、導入した攻撃モデルによる攻撃アクションによってセキュリティ機能が破られていることがわかった。



セキュリティが担保されている状態



セキュリティが担保されていない状態

図 B-7-6　モデル検査による Invariant 違反例

4.3. セキュリティ検証の結果

Event-B での検証は抽象モデルから詳細モデルへリファインメントした際の整合性の検証と、定義したセキュリティプロパティの証明課題の検証の 2 つを実施した。前者は、Event-B モデルの記述時にモデル自体の妥当性確認と共に実施した。後者はモデル記述後の未証明の項目の検証時に実施した。

リファインメントによる整合性の検証は生成される証明がセキュリティプロパティに関連しない項目であり、特に問題はなかった。また、Event-B モデルの妥当性確認も並行して実施し、記述したモデルは仕様及びセキュリティ機能を表現できていることを確認した。それに対して、未証明の項目を検証し、モデル検査の結果から問題箇所を特定した。問題箇所は定義したセキュリティプロパティを違反していることが判明し、セキュリティ機能の脆弱性を検出するに至った。

脆弱性は、第 3 者による攻撃によってセキュリティプロパティを破られることが判明し、それは ProB によるイベントの実行手順から攻撃者による「なりすまし」が可能であることが判明した。また、攻撃モデルを段階的に表現・モデル化したことにより、セキュリティ機能の有効性を明確化することもできた。ある程度の攻撃にはセキュリティ対策が有効に機能していることをセキュリティプロパティの証明が完了することで保証できた。さらに強力な攻撃によってセキュリティプロパティの証明が完了しなかったことでセキュリティが破られてしまうことがわかった。

Event-B により検出した脆弱性は 4 件あり、検出した脆弱性と攻撃手順を要求仕様チームへ情報展開し、要求仕様の修正を行うこととなった。また、それに伴いセキュリティ要件の改善も実施された。

5. 達成度の評価、取組みの結果

5.1. 形式手法によるセキュリティ検証の有効性

本検証は、Event-B を用いて要求仕様を形式仕様記述し、モデルの要素を集合論で表すことで数学的な証明により検証を実施した。形式モデル化によって実施できた検証は要求仕様の一部であり、工数的な制約からすべての検証を実施するには至らなかった。しかし、検証範囲を無線通信ミドルウェアが保証すべき通信プロトコルのセキュリティ機能に絞り込むことで、無視できない脆弱性を検出することができた。セキュリティ検証として攻撃モデルを導入することにより、どのような攻撃が、どのような手順で実行された場合にセキュリティ脅威と成り得るのかをモデル検査から明確化することもできた。このことから、セキュリティ検証を行う上で形式手法が有効な手法の一つであることがわかった。

5.2. 形式手法による検証結果の妥当性

本開発事業では形式手法以外の検証手段として、セキュリティ分析の専門家による要求仕

様の脅威分析を実施していた。専門家による分析においても脆弱性の指摘があり、それはEvent-B により検出した脆弱性と一致した。ただし、専門家による脅威分析と異なる点として、まず Event-B の検証範囲は専門家が実施した分析範囲より規模が小さいという点がある。Event-B の検証範囲外においても専門家による脆弱性の指摘があったことから、検証範囲をより大きくしていくことが必要であることがわかった。また、Event-B で導入した Dolev-Yao モデルでは「暗号は絶対に破られないもの」と仮定してモデル化しており、専門家による暗号系の分析には至っていない点も挙げられる。これについては、Event-B による検証が記号的モデルとしての検証であり（計算量的モデルとして検証することで暗号系の検証もできるが)、今回の検証である程度の成果が出たことを考慮すると（検証の目標にもよるため一概には言えないが)、記号的モデルでの検証を実施することがより効率的に検証できると言える。

したがって、セキュリティの専門家でなくとも、ある程度のセキュリティ知識とモデリング技術を持つことで、形式手法によってセキュリティ脅威や脆弱性を検出することが可能であると言えるだろう。

5.3. 取り組み達成度

この取り組みを通して、Event-B を用いたセキュリティ検証の手順を明確化することができた。形式手法全般的に言えることとして、検証観点の明確化を行うことは必須であり、何をモデル化することでその観点が検証できるのか、いわゆる抽象化や捨象のプロセスは重要である。そのためには自然言語から直接モデル化を行うのではなく、まず、前提条件や処理の振舞いを想定することでモデル化が必要かどうかの判断にもつながる。また、モデルの妥当性を確認する際にも前提条件や振舞いの想定とトレーサビリティを取ることが重要であり、モデルがセキュリティ要件や要求仕様の性質を表現することで形式検証による証明に妥当性があると言える。その結果、定義したセキュリティプロパティが定理証明やモデル検査によって証明が完了することで、対象システムのセキュリティの品質を保証することができる。

Event-B の証明課題の実施については多くの時間を費やした。これは、未証明のどこに問題があるのかを特定するにあたり、モデルの妥当性・要求仕様・セキュリティ要件の 3 つの観点で確認を行わなければいけないことにあった。証明の分析手法として、証明課題の分割や対話証明の実行、そしてモデル検査による Invariant 違反の特定を行うことによって、ある程度の改善は見られた。しかし、モデルの妥当性に問題がある場合に、モデルの修正により再度、証明課題を実施する必要があり、開発現場に適用する上ではモデリングの効率化を図る必要がある。これについては、継続して Event-B を実践することで改善点を見出したいと考えている。

本検証によって、開発対象システムのセキュリティの品質を保証するには至らなかったが、脆弱性を検出する上では有効な手法であった。また、本検証を実施する際に考察した検証の観点やセキュリティ特性ごとのセキュリティプロパティは、対象システムを再検証する際にも利用することができる点でも意義があったと言える。

本取り組みの結果、Event-B でのモデリング技術とセキュリティ検証として、以下の点の技術的な向上が得られた。

- 検証観点をセキュリティプロパティとして定義して検証する。
- 段階的な攻撃モデルの導入によるセキュリティ機能の有効性を検証する。
- 対話証明を容易にするために Technical Invariant を利用する。
- 証明項目の分割や、定理証明とモデル検査を使用して未証明課題を分析する。

これらは、今後 Event-B を用いたセキュリティ検証に役立つものであり、より効率的に形式検証を実践できるものであると考えている。

6. 今後の課題

実装工程への適用

Event-B による上流工程で検証したセキュリティ要件について、下流工程においても検証していくことが重要であると考えている。要求仕様やセキュリティ機能の有効性を上流工程において十分に検証したとしても、設計・製造工程において脆弱性が混入することが懸念される。脆弱性の混入を防止するための手法として、Event-B での検証モデルを、形式記述からプログラムコードが作成できる B-Method に適用する考え方がある。

B-Method は仕様を形式記述することができ、定理証明によりソフトウェア仕様を検証することが可能な形式言語である。要求仕様の検証よりは、プログラムコードの生成に重点を置いているため、抽象化レベルは低く、詳細化に焦点を当てている。Event-B は B-Method の流れを汲んで開発された言語であるため、表現方法や構成が似ており、Event-B の Invariant に記述したセキュリティ要件の制約や保証したい事柄を B-Method の Invariant に記述することで、要求仕様から設計・製造工程を一貫して検証することが可能となる。しかし、B-Method はコード生成を目的として詳細化に焦点を当てていることから、Event-B で記述した抽象レベルが必ずしも B-Method の抽象レベルと一致するわけではない。一貫した検証を実現するためには、B-Method での検証を視野に入れて、Event-B の検証モデルを作成する必要があり、上流の検証において設計レベルの検証にまでモデルの詳細度を落とし込んでいくことを想定しなければならない。それによって抽象レベルで表現したセキュリティ要件を詳細レベルに置き換えることで B-Method に設計・実装していくことが可能となるだろう。

現状は、抽象度の差を埋めるための工夫が必要ではあるものの、まだ試行錯誤を繰り返している段階である。より多くの Event-B、B-Method を用いた形式仕様記述を実践して、そこから適用方法を考察していくことが必要であると考えている。

参考文献

[1] Jean-Raymond Abrial : Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010

[2] IPA/SEC：形式手法導入課題を解決する「形式手法活用ガイドならびに参考資料」(Event-B 編), (online) http://sec.ipa.go.jp/sweipedia/catl050/catm058/cats058/56498/, 2012

[3] IPA：情報技術セキュリティ評価のためのコモンクライテリア CC バージョン 3.1, (online) http://www.ipa.go.jp/security/jisec/cc/, 2012

[4] Michael Jastram ：Rodin User's HandBook v.2.7, (online) http://handbook.event-b.org/current/pdf/rodin-doc.pdf, 2012

B－8

形式仕様記述手法を用いた
高信頼性を達成するテスト手法とその実践 [1]

1. 概要

本編は、モバイル FeliCa IC チップのファームウェア開発に形式手法を適用した事例を報告したものである。

形式手法の利用に関して 3 段階の適用レベルが示されることがあるが、そのうち、本事例は、形式仕様記述を行うレベル 0 の段階である。レベル 0 の段階とは、証明までは行わないが、数学的な記法を用いて厳密な仕様を記述する段階である。本開発では、上流工程において記述した形式仕様記述を検証した。その後、仕様に基づいたテスト工程において形式仕様記述を活用して効率的にテストを実施した。

本事例報告では、仕様の読みやすさやテスト工程での活用方法を考慮した仕様記述フレームワークの設計方法とテストにおける形式仕様記述の活用方法について述べる。テストにおける形式仕様記述の活用方法として、モバイル FeliCa IC チップ開発において定めた品質目標を効率的に実施するための形式仕様記述の活用方法について述べる。これらの取り組みの評価として、形式仕様記述を用いた場合と用いなかった場合の開発効率について簡単ではあるが考察する。また、形式仕様記述を用いることで品質目標を達成したテストの効果について、開発中に発見した不具合から考察する。

2. 取組みの目的

モバイル FeliCa IC チップはスマートフォンや携帯端末に組み込まれ、電子マネーや公共交通機関の電子乗車券やポイントカードなど広く利用されている。モバイル FeliCa IC チップのファームウェアは、例えば電子マネーの残高情報や履歴情報や鍵情報を管理するなど、セキュリティシステムにおいて中核を担っている。そのため、社会基盤システムの一部として高信頼性が求められる。

モバイル FeliCa IC チップ開発では、ソフトウェア開発工程全体を複数の工程に分けて、各工程を専門性を持ったチームが担当することで、分業をして組織的に開発を進めている。図 B-8-1 に各開発工程と各工程の成果物の関係の概略を示す。実際の開発プロセスは、図 B-8-1 のようなウォータフォールモデルに準じたものではないが、説明のために簡略化した。

[1] 事例提供:フェリカネットワークス株式会社　開発 2 部 2 課　中津川　泰正 氏、栗田　太郎 氏

以下では、開発工程全体を要求分析工程と仕様策定工程と設計・実装工程とテスト工程に分けて説明をする。また、図中では各工程で作成する文書やソースコードの関係を示した。図 B-8-1 において、上流工程において作成した仕様書は、設計書と実装コードとテスト項目表などの入力となることを示した。そのため、仕様書が誤っていたり、曖昧な記述があったり、理解容易性に問題があったり、開発中の仕様変更が仕様書に正しく反映されていなかったりすると、開発スケジュールの遅延や開発費用の増加など、開発効率の問題につながる可能性がある。また、開発効率の問題にとどまらずに、仕様の曖昧さや不完全さが市場において重大なトラブルの原因となることが考えられる。

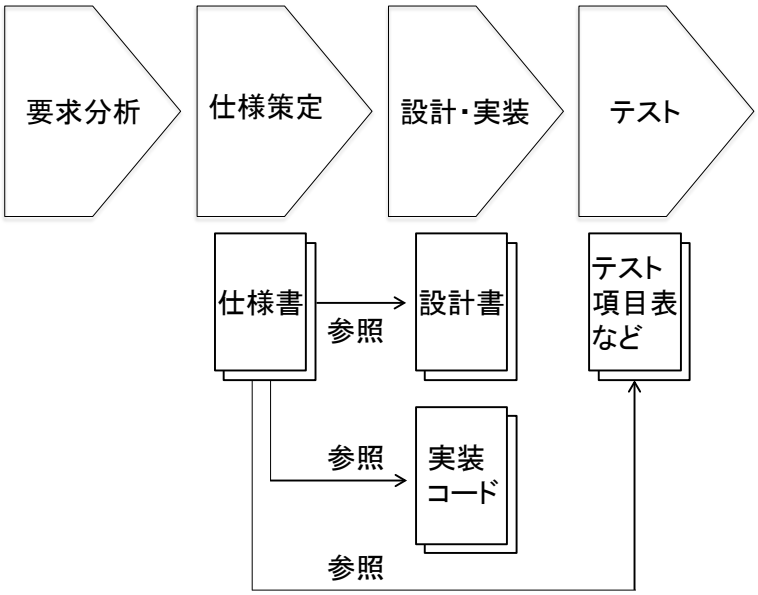


図 B-8-1　開発工程と成果物の関係

これらの問題を解決するために、上流工程において仕様を厳密に記述し検証するために数多くの開発手法が提案されている。その中の 1 つに形式仕様記述手法がある。形式仕様記述手法は、数学的な記法を用いて厳密な仕様を記述するための仕様記述の手法である。また、記述した仕様を形式仕様記述手法によって上流工程で検証することにより、早期に仕様の不具合を取り除き、後工程における手戻りを減らすことができる。ここでは形式仕様記述手法を用いて記述した仕様を形式仕様記述と呼ぶこととする。

モバイル FeliCa IC チップ開発では、2 つの製品開発において形式仕様記述手法を適用した。初めて形式仕様記述手法を適用した 1 つ目の製品開発において、前述の上流工程において作成する文書の課題に加えて、以下の課題認識から形式仕様記述手法に着目した。

- モバイル FeliCa IC チップは広く社会基盤の一部として利用され、製品開発における品質確保が重要な課題である。
- モバイル FeliCa IC チップは同一の仕様を複数のハードウェアプラットフォームに

実装しており、製品の相互接続性を確保するためには、仕様の厳密性が重要な課題である。

・　モバイル FeliCa IC チップはセキュリティ製品として全ての入力に対して正常系、準正常系の振る舞いを厳密に定義する必要がある。

1 つ目の製品開発において、形式仕様記述手法を用いて仕様を厳密に記述し、検証を行うことで、これらの課題に対して形式仕様記述手法の効果を実感することができた。一方で、実際に形式仕様記述手法を用いて、仕様策定工程から設計・実装工程やテスト工程までの開発を経験することで、以降において述べる新しい課題・目標を得ることができた。形式仕様記述手法を用いて仕様を記述した仕様策定者は、形式仕様記述手法を使いこなすことで仕様の検証を行い、多くの仕様の不具合を上流工程において見つけることができた。設計・実装工程やテスト工程においても、形式仕様記述を参照して設計・実装・テストを行っていたが、仕様策定工程以外の工程において、さらに形式仕様記述を使いこなすための課題・目標が見えてきた。1 点目の課題・目標は、仕様策定者以外の開発者が形式仕様記述を読むときの仕様の読みやすさに関する課題と目標であった。この課題と目標について 2.1 において述べる。2 点目の課題・目標は、テスト工程において、形式仕様記述を参照するだけではなく、具体的にテストの実施効率を向上させるための形式仕様記述の活用方法に関する課題と目標である。この課題と目標について 2.2 において述べる。

2.1. 仕様の読みやすさに関する課題と目標

仕様策定者以外の工程を担当する開発者から、形式仕様記述は、厳密に書かれているために自然言語の仕様書と比べて厳密性や完全性は優れているという意見をもらったが、仕様の読みやすさについて多くのコメントをもらった。仕様策定者以外の工程を担当する開発者からの仕様の読みやすさに関する問い合わせや意見について考察をすると次のようになった。

(1)　形式仕様記述を記述する目的として、2 つの目的があった。1 つ目は、仕様アニメーションと呼ばれる方法によって形式仕様記述を言語処理系の上で動かすための入力データを与えて仕様の検証を行う目的である。仕様アニメーションとは、記述した仕様を仕様記述言語処理系の上で実行することである。プログラムの実行と区別して、仕様アニメーションと呼ぶことが多い。2 つ目の目的は、開発者に仕様を伝える目的である。課題は、1 つ目の目的のための仕様を動かすための記述が、仕様として伝えるための記述なのか、単に仕様アニメーションによって仕様を動かすための記述なのかが読み手には分からないというものであった。「仕様としてどの箇所を読めばいいのか。どの箇所が動かすための仕組みの部分で読まなくていいのか。」という問い合わせがあった。

(2)　設計者からは、形式仕様記述について、仕様として規定する範囲と設計者が自由に設計することができる範囲が仕様書からは読み取れないといった指摘を受けた。例え

ば、「仕様書で定義したデータ構造について、そのまま設計において用いるべきなのか。設計者が違うデータ構造として再定義をすることができるのか。」といった問い合わせがあった。

(3) 1 つ目の製品開発においては、形式仕様記述の各行の上に同様の内容の仕様を自然言語を用いて併記した。この形式仕様記述と自然言語を用いた 2 つの記述について、記述誤りや仕様変更対応漏れによって、記述間に不整合があるなどの問題があった。また、仕様策定者は 2 つの記述をメンテナンスする必要があり、作業効率も問題であった。

(1), (2)の課題については、1 つ目の製品開発においては、これらの課題を解決するために設計者と議論をしながら、形式仕様記述の読み方に関するガイドラインを作成して、読み手に (1), (2) の方針を伝えることによって課題を解決した。2 つ目の製品開発において、1 つ目の製品開発において得られた (1), (2), (3) の課題から次のような目標を設定した。

- (1)と(2)の課題を解決する仕様の読みやすさに優れた仕様記述フレームワークと仕様を実現する。
- (3)の課題を解決するために、自然言語の仕様を記述しなくても、形式仕様記述のみを用いて、読み手に仕様を伝えることができるようにする。

(1) (3) の課題を解決する仕様の読みやすさに優れた仕様記述フレームワークについて 4.1 において述べる。(2) について本稿では割愛する。

2.2. テストにおける課題と形式仕様記述手法の活用方法の目標

1 つ目の製品開発のテスト工程において、形式仕様記述を参照してテスト項目を作成していたが、形式仕様記述を参照するだけではなく、テスト効率を具体的に向上させるために形式仕様記述を活用していくことを目標として設定した。具体的にテスト効率を向上させるとは、形式仕様記述を用いることによって、これまで行っていた作業を形式仕様記述の活用によって無くしたり、形式仕様記述に置き換えることができるといったテスト効率の具体的な向上のことをいう。本章では、まず、テストにおける課題について述べる。その後、この課題を解決するための形式仕様記述の活用方法に関する目標について述べる。

図 B-8-2 を用いてテストにおける課題ついて説明する。図 B-8-2 は「①テスト対象」に対して「②入力」を与えてテストを実施する流れを模式的に表したものである。「①テスト対象」に「②入力」を与えることによりテスト対象は「③入力時の内部状態」から「④出力時の内部状態」へとテスト対象の内部状態が遷移して、テスト対象から「⑤出力」を得る。テストを実施するために、これらの一連の流れを実行するためのテストスクリプトを作成した。テストスクリプトには「⑤出力」が仕様どおりの出力であることを確認するために、「⑥」に示した、入力と入力時の内部状態から仕様書を参照して、期待する出力を手作業によりテスト

スクリプトに記載した。ここでは、この期待する出力を期待値と呼ぶ。「⑤出力」の期待値と同様に「④出力時の内部状態」についても、全内部状態のうち「⑪仕様上、変化した範囲の内部状態」について期待値をテストスクリプトに記載した。

テストにおける課題は、2 点あった。 1 点目は入力として用いるテストのデータバリエーションの範囲に関する課題であり、2 点目は出力時の内部状態について期待値として確認するデータの範囲に関する課題であった。

まず、1 点目について、「①入力」として用いるテストデータのバリエーションの範囲について、「⑧入力の全バリエーション」を手作業で網羅することが現実的ではないという課題があった。例えば、64 bit の入力について、全ての入力をテストする場合を考えると 2 の 64 乗通り(1.8 × 10 の 19 乗通り)の膨大な件数のテストが必要になる。そのため、手作業によって入力にバリエーションを持たせたテストを行う範囲を、「⑨仕様上の判定条件から抽出した入力のバリエーションの範囲」とした。この範囲であれば、仕様書から判定条件を抽出して、各判定条件について同値分割や境界値分析といったテスト手法を用いながらテスト件数を減らすことができる。この「⑨仕様上の判定条件から抽出した入力のバリエーションの範囲」を、ここでは Level 1 の段階のデータバリエーションの品質目標と呼ぶこととした。次に「⑨」の範囲外である仕様上判定条件として表れない入力パラメータにバリエーションを持たせたテストを行うことを考えた場合、前述のとおり、網羅するために必要なテスト件数から、手作業によって入力全体を網羅することはできないという課題があった。この課題に対して、社会インフラの一部としての役割を担う開発対象が求められる品質や過去の製品開発において開発中に見つけた不具合を考えると、「⑧ 入力の全バリエーション」を全て網羅することは難しいとしても、機械的にランダムに入力とするテストデータを自動的に生成して、「⑨」の範囲外の入力についてもバリエーションを持たせたテストを行うことを目標とした。この「⑨」の範囲外を対象にして機械的にランダムに入力とするテストデータを生成してテストを行うことを、ここでは Level 2 のデータバリエーションに関する品質目標と呼ぶ。また、Level2 のデータバリエーションのテストでは、テスト対象から得られた出力が正しいことを確認するために、手作業では期待値を作成することができないので、期待値を自動生成するような機械支援の仕組みを構築する必要がある。このデータバリエーションに関する Level 2 の品質目標を達成するための機械支援の仕組みとして、以下の仕組みが必要となる。

(1)　「⑨仕様上の判定条件から抽出した入力のバリエーションの範囲」の範囲外である「②入力」を機械的にランダムに生成する仕組み
(2)　「⑤出力」が正しいことを確認するための期待値となる出力結果生成器
(3)　「④出力時の内部状態」が正しいことを確認するための期待値となる内部状態生成器

(2) の出力結果生成器と (3) の内部状態生成器は、これまでの開発では仕様書を参照して 2 つの生成器をテスト工程において開発をしていたが、2 つ目の製品開発では形式仕様記述

を活用することを目標とした。ここでは、説明を簡単にするために、外部から与える「②入力」に関するデータバリエーションについて説明を行ったが、「③入力時の内部状態」についても同様にデータのバリエーションがある。つまり、仕様上の判定条件から抽出した Level1 の範囲と、仕様上の判定条件には表れない Level2 の範囲がある。この「③入力時の内部状態」に関する入力時のテストデータのバリエーションについては 3.2 において述べる。

次に、2 点目の出力時の内部状態の期待値としての確認範囲の課題について述べる。図 B-8-2 の「⑦」に示したように、テストスクリプトには、出力時の内部状態に関する期待値として、仕様上変化した範囲のみを記載した。この内部状態について、変化した範囲のみを確認することを、ここでは Level1 の期待値の確認範囲と呼ぶ。期待値の確認範囲について、開発対象に求められる品質特性から、仕様上の内部状態全体を出力結果として確認をする品質目標とした。この内部状態全体を確認範囲とした品質目標を、ここでは Level 2 の期待値の確認範囲と呼ぶ。Level2 の期待値の確認範囲について、期待値を手作業により作成するには範囲が広すぎるという課題があった。このため、内部状態全体を機械的に生成して、全範囲を機械的に確認することを目標とした。この Level2 の機械支援の仕組みを構築するためには、前述の以下の仕組みが必要となる。

(3)「④出力時の内部状態」が正しいことを確認するための期待値となる内部状態生成器

この (3) の期待値の生成器として、形式仕様記述手法で記述した仕様書を活用することを目標とした。

これらの課題の具体例と解決方法について 4.2 と 4.3 において述べる。テストの実施効率や Level2 の品質目標を達成することの効果については、5 において分析をする。

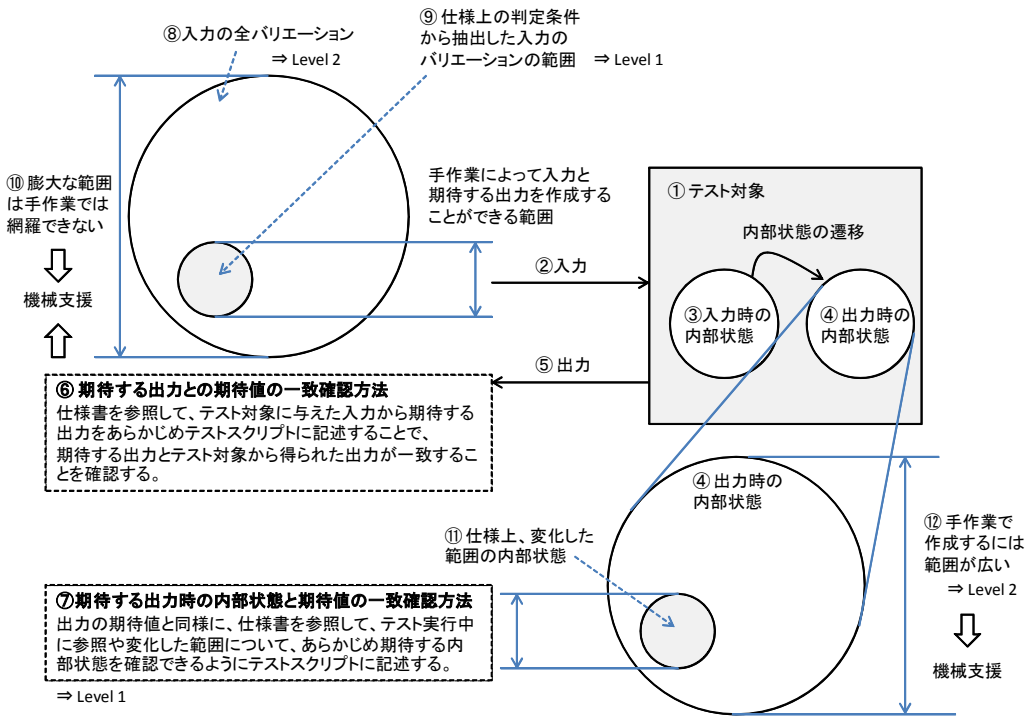


図 B-8-2　本書において述べる品質目標に関する課題

3. 取り組み対象、適用技術・手法、評価・計測

3.1. 取り組み対象

取り組み対象は 2014 年 2 月現在、開発中のモバイル FeliCa IC チップ開発である。このモバイル FeliCa IC チップは、DES (Data Encryption Standard) 暗号方式に加え、AES (Advanced Encryption Standard) 暗号方式を備えており、セキュリティをさらに強化している。このモバイル FeliCa IC チップはハードウェアとファームウェアから構成されており、本書ではこのファームウェア開発について述べる。

3.2. 適用技術・手法

ソフトウェアの品質を効率的に確保するための手法として形式手法がある。形式手法の利用に関して 3 段階の適用レベルが示されることがあるが、そのうち、本事例は、形式仕様記述を行うレベル 0 の段階である。レベル 0 の段階とは、証明までは行わないが、数学的な記法を用いて厳密な仕様を記述する段階である。形式仕様記述を行うことによる効果の考察は、IPA／SEC の上流品質技術部会 人材育成 WG が取りまとめた「実務家のための形式手法 厳密な仕様記述を志すための形式手法入門 第二版 事例 : 成功事例」など IPA のホームページ上で公開されている参考文献がある。この参考文献にあるように仕様策定工程において、レベル 0 の形式仕様記述を行うだけであっても、これらの課題に対して効果を得ることができる。

本開発では、形式仕様記述を行うために形式仕様記述言語 VDM++ を用いた。VDM++ はモデル指向型の形式仕様記述言語である。モデル指向型とは、状態機械の仕様をその状態を構成するデータ構造と、それに対する操作の入出力仕様で定める方法である。VDM++ には仕様を記述するために、事前条件や事後条件や不変条件の記述、集合や写像などのデータ構造を扱うための言語仕様を提供している。

VDM++ の統合開発環境として VDMTools [1] がある。本開発では、この VDMTools を用いた。VDMTools を用いることで、記述した仕様の構文チェックや型チェックを行うことができる。また、VDM++ の言語処理系の上で仕様アニメーションにより記述した仕様を検証することができる。仕様アニメーションとは、記述した仕様を仕様記述言語処理系の上で実行することである。プログラムの実行と区別して、仕様アニメーションと呼ぶことが多い。

3.3. 取り組みの達成度評価方法

取り組みの達成度の評価方法として、形式仕様記述を用いた場合と形式仕様記述を用いなかった場合について簡単ではあるが開発効率を比較する。また、形式仕様記述を活用したテストにおいて発見した不具合から形式仕様記述を活用したテストの効果を評価する。

4. 取り組みの実施、及び実施上の問題、対策・工夫

本開発では、仕様策定工程において、形式仕様記述言語を用いて仕様を記述した。また、仕様アニメーションによる検証を仕様策定工程において行った。設計・実装工程では、形式仕様記述を参照して設計・実装を行った。テスト工程では、単体テストから統合テストやシステムテストなど、様々なテストを実施した。テスト工程においては、形式仕様記述を活用して効率的にテストを実施した。4.1 では、仕様策定工程において形式仕様記述言語 VDM++ を用いて仕様を記述したときの工夫について説明する。4.2 では仕様に基づくテストにおいて本開発で定めた品質目標について述べる。4.3 では、4.2 において述べた品質目標を達成するためのテスト実施内容について説明する。

4.1. 形式仕様記述手法を用いた仕様の記述

4.1.1. 形式仕様記述手法を用いた仕様の記述対象

まず、形式仕様記述言語を用いて記述を行う仕様の記述対象を決めた。仕様には、複数のタスクが競合した場合のタイミング仕様や、ある規定時間以内に処理を終了させるといった性能に関する仕様や、ハードウェアが故障した場合の仕様など、様々な仕様がある。これらの時間の要素を含む仕様や、ハードウェアプラットフォームに依存した仕様は、形式仕様記述言語ではなく、自然言語を用いて記述した。形式仕様記述言語を用いて記述を行う対象とした仕様は、複数の製品間で共通となるべき振る舞いを規定した読み手の多い仕様や、仕様のボリュームや複雑さから仕様の曖昧さに起因する市場トラブルの原因となる可能性が高い仕様を選択した。これらの観点から、入力と入力時の内部状態と出力と出力時の内部状態の関係について、形式仕様記述言語を用いて仕様を記述することとした。

4.1.2. 仕様記述フレームワーク・仕様記述ライブラリの設計

形式仕様記述言語を用いて記述する仕様の対象を決めた後に、開発ドメインの特徴に応じて仕様記述フレームワークやドメイン固有の仕様記述ライブラリを設計した。仕様記述フレームワークの設計において、以下の 3 つの仕組みを構築するという工夫を行った。

(1) 上流工程において、仕様アニメーションによる検証を行うことを目標の 1 つとしていた。このため、仕様記述内には、仕様アニメーションにより仕様を動かすことを目的とした記述と、仕様書として読み手に仕様を伝えることを目的とした記述が混在した。これらを明確に分離する仕組みを構築した。

(2) 仕様記述の読み手である設計者に対して、仕様が規定する範囲と設計者が決める範囲を明確に伝えるための仕組みを構築した。例えば、形式仕様記述においてデータ構造を定義した場合、そのデータ構造を設計においても同じデータ構造とすべきか、それとも、設計者が性能など考慮して他のデータ構造に置き換えることができるのかといったことを仕様記述から読み取ることができるようにした。

(3) 仕様記述からテスト項目の抽出やレビューを効率的に行うための仕組みを構築した。

上記の (1)～ (3)については、論文 [2] において詳細に述べられている。本書では(1)について 4.1.3 において簡単に述べる。(2) については本書では割愛する。(3) については、4.1.3 において簡単に述べる。4.2 と 4.3 において、論文 [2] では述べられていない、品質目標とその目標を達成するためのテスト実施内容について述べる。

4.1.3. 具体的な仕様記述例

以下では、具体的な記述例を用いて開発における取り組みについて説明する。記述例では詳細な VDM++ の言語仕様を説明することを目的としていないため、詳細な説明は行わない。VDM++ の言語仕様については、VDM++ 言語マニュアル [3] に詳細な説明がある。また、この記述例は実際のモバイル FeliCa IC チップ開発で使用した記述とは異なるが、開発における取り組みを簡潔に説明するには十分であると考える。

まず、仕様記述フレームワークや開発ドメイン固有の仕様記述ライブラリなどの検討・記述を行った。その後、これらのフレームワークやライブラリを用いて入力と出力の関係を仕様として記述した。具体的な記述例として、実際のモバイル FeliCa IC チップの仕様とは異なるが PIN (Personal Identification Number) と呼ばれる本人確認のための暗証番号を照合する仕様を用いる。仕様記述において、まず、型を定義した。その後、型を用いて入力と出力の関係を記述した。

型の定義について例を用いて説明をする。PIN 照合の場合、PIN 値として 0 から 9 までの値からなる 4 桁の整数を用いる。この PIN 値の型定義を図 B-8-3 に示す。

```
1  public PIN = seq of nat
2  inv pin == len pin = 4 and
3             forall i in set inds pin & 0 <= pin(i) and pin(i) <=9;
```

図 B-8-3　型定義の記述例

1 行目において PIN 値は seq of nat という型であることを定義している。nat は 0 から始まる自然数であり、seq of A は A からなる列 (sequence of A) を意味する。2 行目の inv は不変条件 (invariant) の頭文字である。不変条件とは型が常に満たすべき条件である。2 行目と 3 行目において PIN 型の不変条件として、列の長さは 4 であり、全ての列の要素は 0 から 9 の範囲内であるということを定義している。このように、型の不変条件を定義できる点が VDM++ の特徴の 1 つである。仕様アニメーションにより仕様を動かしながら、不変条件に違反するようなケースがないことを調べることができる。この仕様アニメーションにより、仕様記述内の矛盾を見つけるといった一貫性の検証ができる。

次に、仕様アニメーションにより動かすことを目的とした記述と読み手に仕様を伝えることを目的とした記述を分離する仕組みについて、PIN 照合仕様の記述を用いて説明する。仕様アニメーションにより動かすことを目的とした記述は、図 B-8-4 の「PIN 照合仕様_実行」関数と、この関数内から呼び出している 3 行目の「impl_PIN 照合仕様_実行」関数である。

```
1   public PIN照合仕様_実行 : INPUT * STATE -> OUTPUT * STATE
2   PIN照合仕様_実行(input, state_in) ==
3     impl_PIN照合仕様_実行(input, state_in)
4   post PIN照合仕様(input, state_in, RESULT.#1, RESULT.#2);
```

図 B-8-4 「PIN 照合仕様_実行」関数の記述例

3 行目の「impl_PIN 照合仕様_実行」関数は、頭文字に implementation の略称である「imple_ 」を付けることによって、この関数は動かすための仕組みであることを読み手に伝える命名規則とした。読み手に仕様を伝えることを目的とした記述は、4 行目の post というキーワードの後に記述した「PIN 照合仕様」関数である。「post」は事後条件(post-condition) を表したキーワードであり、この「PIN 照合仕様_実行」関数を実行した後に成立している条件を「post」以降に書くことができる。この関数の処理の流れは次のようになる。入力 (INPUT 型) と入力時の内部状態 (STATE 型) の値を引数として「PIN 照合仕様_実行」関数を呼び出すと「impl_PIN 照合仕様_実行」関数を実行して、この関数の戻り値として結果 (OUTPUT 型) と 出力時の内部状態 (STATE 型) が返る。この 2 つの戻り値が 4 行目の「RESULT.#1」と「RESULT.#2」の変数に入り、「PIN 照合仕様」関数を実行する。このようにして、入力を「PIN 照合仕様_実行」関数に与えることにより、「post」に記述した「PIN 照合仕様」関数を評価することができる。

このような仕組みにした理由は、「PIN 照合仕様」関数において、仕様を入力と出力の関係式として記述することにより、簡潔な仕様記述としたかったためである。平方根を求める例を用いて記述の簡潔性について説明をする。平方根を求める機能を in と out の関係式で表した場合、「abs (out**2 - in) <= 許容誤差」というように、「out を二乗した値から in を減算した値の絶対値が許容誤差以内であること」と平方根の仕様を簡潔に、in と out の関係式として記述することができる。一方、「impl_PIN 照合仕様_実行」関数の場合、in からニュートン法などの手法を用いて out を求めなければならない。このように、入力と出力の関係を記述することで、仕様を簡潔に表現することができる。この仕組みによって「PIN 照合仕様」関数には読むことを目的とした仕様を記述して、「impl_」などの動かすことを目的とした記述と分離した。これにより、「How」の要素を分離した「What」の要素からなる仕様を読み手に伝えるようにした。

次に仕様記述からテストケースの抽出やレビューを効率的に行うための仕組みについて述べる。仕様策定工程や設計工程やテスト工程において、ディシジョンテーブルが広く用いられている。ディシジョンテーブルとは、条件の組み合わせとアクションの関係を表したものである。例えば、PIN 照合の仕様について、ディシジョンテーブルを参考にした表を使用すると表 B-8-1 のようになる。表 B-8-1 の No.3 列を見ていくと、条件の組み合わせとして、PIN 照合の対象とした 「Application ID が存在するか？」と「PIN 照合試行回数が上限以内か？」という条件が YES で「PIN 照合番号が正しいか？」という条件が NO の場合

に、結果として「PIN 番号不正」となることが分かる。また、内部状態の変化として「実行後の PIN 照合試行回数」が「1 増加」するということを表している。このようにディシジョンテーブルを用いることで条件と結果の組み合わせとして、仕様を簡潔に表現することができる。

表 B-8-1　PIN 照合仕様をディシジョンテーブルを用いて表した例

分類	項目	No.1	No.2	No.3	No.4
条件	ApplicationID が存在するか？	NO	YES	YES	YES
	PIN 照合試行回数が上限以内か？	—	NO	YES	YES
	PIN 照合番号が正しいか？	—	—	NO	YES
結果	ApplicationID 不正	YES			
	PIN 照合上限回数オーバ		YES		
	PIN 番号不正			YES	
	PIN 照合成功				YES
内部状態の変化	実行後の PIN 照合試行回数	変化なし	変化なし	1 増加	0 にクリア

テストケースの抽出やレビューを効率的に行うための仕様記述フレームワークの設計において、表 B-8-1 に示したディシジョンテーブルの構成を用いた仕様記述フレームワークを検討した。その理由を以下に示す。

- ディシジョンテーブルは、仕様を簡潔な構造で表現することができる。また、一覧性にも優れている。そのため、この構造を仕様記述フレームワークに適用することで、フレームワークもまた簡潔になると考えたためである。
- ディシジョンテーブルは、テストとの親和性が良い。4.3 において述べる仕様に基づくテストケースの抽出においてディシジョンテーブルを用いている。仕様記述をディシジョンテーブルの構成と合わせておくことで、仕様とテスト項目の対応関係が明確になると考えた。これにより、仕様から体系的にテスト項目が抽出できるようになるため、仕様とテスト項目のレビューも容易になると考えたためである。

図 B-8-5 と図 B-8-6 にディシジョンテーブルの構造を用いた仕様記述の例を示す。図 B-8-5 の「PIN 照合仕様」は図 B-8-4 の 4 行目の事後条件から呼び出している。図 B-8-5 の 3 行目から呼び出している「PIN 照合結果判定仕様」関数は、表 B-8-1 の「条件」の組み合わせを表したものである。「PIN 照合結果判定仕様」関数の中身を図 B-8-6 に示す。この関数内において、デジョンテーブルの「条件」行にある各条件について、各条件名を関数名とした関数を用いることにより、ディシジョンテーブルと仕様記述の対応関係を明確にした。図 B-8-5 の「PIN 照合仕様」関数の 4 行目から 11 行目は表 B-8-1 の「結果」行に対応している。例えば、図 B-8-5 の 3 行目の「PIN 照合結果判定仕様」関数において、「<PIN 番号不正>」

の戻り値が返ってきたとき、9 行目の「PIN 番号不正の仕様」関数が呼び出される。この「PIN 番号不正の仕様」関数内で、例示はしなかったが表 B-8-1 において示した「PIN 照合試行回数が 1 増加していること」という仕様を記述している。

```
1   public PIN照合仕様 : INPUT * STATE * OUTPUT * STATE -> bool
2   PIN照合仕様(input, state_in, output, state_out) ==
3     cases PIN照合結果判定仕様(input, state_in):
4       〈ApplicationID不正〉
5         -> ApplicationID不正時の仕様(input, state_in, output, state_out),
6       〈PIN照合上限試行回数オーバ〉
7         -> PIN照合上限回数オーバ時の仕様(input, state_in, output, state_out),
8       〈PIN番号不正〉
9         -> PIN番号不正時の仕様(input, state_in, output, state_out),
10      〈PIN照合成功〉
11        -> PIN照合成功時の仕様(input, state_in, output, state_out)
12    end;
```

図 B-8-5 「PIN 照合仕様」関数の記述例

```
1   public PIN照合結果判定仕様 : INPUT * STATE -> OUTPUT
2   PIN照合結果判定仕様(input, state_in) ==
3     if not ApplicationIDが存在するか？(input, state_in) then
4       〈ApplicationID不正〉
5     elseif not PIN照合試行回数が上限以内か？(input, state_in) then
6       〈PIN照合上限試行回数オーバ〉
7     elseif not PIN照合番号が正しいか？(input, state_in) then
8       〈PIN番号不正〉
9     els
10      〈PIN照合成功〉;
```

図 B-8-6 「PIN 照合結果判定仕様」関数の記述例

このように、ディシジョンテーブルと対応付けた形式仕様記述とすることで、形式仕様記述言語を熟知していない読み手でも仕様記述の構造を理解して形式仕様記述を読むことができるように工夫した。これらの工夫に加え、関数名に日本語を用いることにより、1 つ目の製品開発では、形式仕様記述と自然言語の両方の記述を併記していたが、今回の開発では自然言語では仕様の記述を行わずに、形式仕様記述だけによって仕様を伝えることができるように工夫をした。また、これらのディシジョンテーブルと対応付けるといった工夫や関数名に日本語を用いるといった工夫により、仕様のレビュー効率を高めた。また、テスト項目はディシジョンテーブルを用いて作成したため、ディシジョンテーブルの構造を用いている形

式仕様記述から体系的にテスト項目を作成することができた。これにより、仕様とテスト項目の対応関係が明確となり、テストの作業効率とテスト項目のレビュー効率を高めた。

4.2. 仕様に基づくテストの品質目標

モバイル FeliCa IC チップのファームウェア開発のテスト工程では、単体テストや統合テストやシステムテストなど様々なカテゴリのテストがあるが、本章では特に、製品実装が仕様を満たしていることを確認することを目的とした仕様に基づくテストについて述べる。品質目標として、2.2 において述べたテストのデータバリエーションに関する品質目標と期待値の確認範囲に関する品質目標を定めた。

テストのデータバリエーションおよび期待値の確認範囲について、図 B-8-7 に示した段階を用いた。図 B-8-7 は、入力と入力時の内部状態に着目して、入力は仕様上の判定条件から抽出した入力の範囲とそれ以外の範囲、内部状態は仕様上参照または変更した範囲とそれ以外の範囲という 2 つの段階を図示している。データバリエーションにおける Level1 の段階とは、その仕様において、判定条件として表れる入力のパラメータと、仕様上参照または変更を行っている内部状態に着目してテスト項目を抽出する段階である。データバリエーションにおける Level 2 の段階とは、その仕様においては参照も変更も行っていない入力と内部状態についてデータのバリエーションを持たせたテストを行う段階である。例えば、PIN 照合の仕様では、入力で指定した PIN 値と入力で指定したアプリケーション ID が保持する PIN 値を比較する。PIN 照合仕様において、Level1 の段階では、入力として指定したアプリケーション ID が保持している PIN 値のデータにバリエーションを持たせたテストを行う。Level2 の段階では、指定したアプリケーション ID 以外のアプリケーション ID の PIN 値にデータのバリエーションを持たせたテストを行う。例えば、電子マネー A の PIN 値の照合を行うテストにおいて、電子マネー A の PIN 値に加えて他の電子マネー B の PIN 値にデータのバリエーションを持たせたテストを行う。

期待値の確認範囲の段階もデータバリエーションの段階と同様に Level1 の段階とは、仕様が参照や変更した範囲の値が正しいことを確認する段階である。Level2 の段階では、仕様から参照も変更も行っていない範囲の値について、変化していないことを確認する。

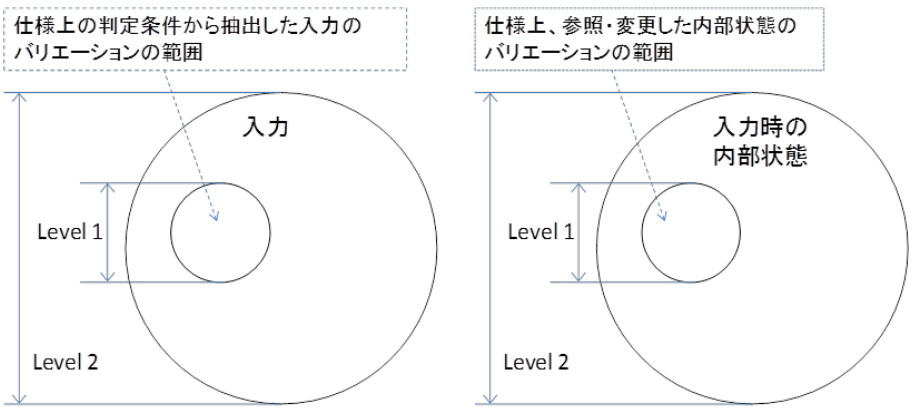


図 B-8-7　品質目標のレベル

テストデータのバリエーションおよび期待値の確認範囲について、Level1 の段階は一般に広く行われていることが多い。しかし、Level2 の段階は、人手で実施した場合、テスト項目数および確認範囲が膨大になってしまう傾向があるため、Level2 の段階のテストを実施することは容易ではない。例えば、PIN 照合仕様において、指定したアプリケーション ID 以外のアプリケーション ID の PIN 値にデータバリエーションを持たせようとした場合、IC チップが保持することができるアプリケーション ID の数が仮に 100 個として、アプリケーション ID の ID として取り得る値が 1 から 100 であった場合、1 つのアプリケーションが登録されている場合から 100 個のアプリケーションが登録されている場合までを考えると、膨大なテストパターン（100C1 + 100C2 + 100C3 + ... 100C100）となる。期待値の確認範囲においても、Level2 の段階では、例えば、100 個のアプリケーションが IC チップ内に存在した場合、PIN 照合対象のアプリケーションに関する期待値だけではなく、残りの 99 個のアプリケーションの期待値をあらかじめ準備する必要がある。このように、Level2 の段階になると、データバリエーションのパターンと期待値の確認範囲が膨大になる点が課題となる。

4.3. 品質目標を達成するためのテストの実施内容

本章では仕様に基づいたテストのうち、4.2 において述べたデータバリエーションおよび期待値の確認範囲について Level1 と Level2 の段階の品質目標を達成したテスト実施内容について述べる。

まず、期待値の確認範囲について、Level1 の品質目標を達成するためのテストの流れを図 B-8-8 を用いて説明する。「①形式仕様記述」を参照しながら手作業により「②テスト項目」をディシジョンテーブルを用いて作成した。「②テスト項目」から半自動生成により「③テストスクリプト」を作成した。半自動生成とは、テストスクリプトのテンプレートを手作業で作成し、このテンプレートと「②テスト項目」を用いて機械処理を行うことによって「③テストスクリプト」を自動生成することを半自動生成と呼ぶこととした。Level 1 を満たす期待値の範囲については「③テストスクリプト」に直接記載した。

Level 2 の品質目標を満たす期待値として、「③テストスクリプト」を用いて仕様アニメーションにより「①形式仕様記述」を動かすことにより得た出力時の内部状態を用いた。製品実装に対して「③テストスクリプト」を実行した後に得られた内部状態と「①形式仕様記述」から得た内部状態を比較することにより、期待値に関する Level 2 の品質目標を達成した。

次に、テストのデータバリエーションについて、Level2 の品質目標を満たすテストの流れを図 B-8-8 に示した。「①形式仕様記述」を参照しながら手作業により「⑤ランダムテストデータ生成ツール」を作成した。「⑤ランダムテストデータ生成ツール」とは、ランダムに入力となるテストデータを生成するツールである。このツールは、まず、あらかじめ指定した選択確率に基づいて「結果」を選択する。例えば、PIN 照合の場合は、「ApplicationID 不正」・「PIN 照合上限回数オーバ」・「PIN 番号不正」・「PIN 照合成功」の中から選択確率に基づいて「結果」を選択する。その後、その「結果」を実現するためのパラメータをランダムに選

択する。例えば、結果の中から「PIN 番号不正」の結果を選択した場合は、「①形式仕様記述」から得た内部状態を参照して、IC チップ内部に保持している Application ID をランダムに選択する。このとき、「PIN 番号不正」のエラーを発生させるために「PIN 照合上限回数オーバ」とはならないように Application ID を選択する必要がある。テスト対象の Application ID を選択したら、IC チップが保持している PIN 値とは異なる PIN 値をランダムに選択することによって「PIN 番号不正」のテストを行う。実行後の期待値の確認は、Level1 のデータバリエーションのテストと同様に「①形式仕様記述」から得た内部状態と製品実装に対して実行した後に得た内部状態を比較することによって行う。また、内部状態だけではなく、製品実装の出力が正しいことは、形式仕様記述の出力と製品実装の出力を比較することにより判定した。このテストでは、例として PIN 照合について説明をしたが、新規 Application の登録や PIN 情報の変更などといった、IC チップの内部状態を変更する処理を連続して長時間実行した。これらの処理によって、IC チップが内部状態として保持する情報をランダムに変更することができる。これにより、PIN 照合の仕様からは参照および変更を行わない Level2 の範囲についてもバリエーションを持たせたテストを実施することができたと考える。

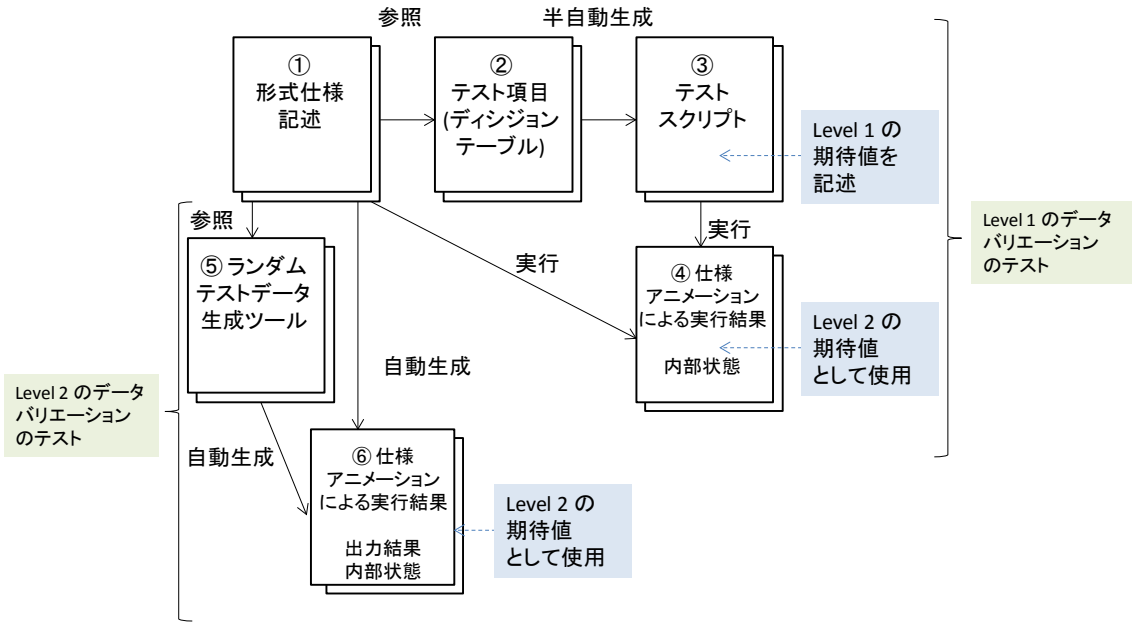


図 B-8-8　品質目標を達成するためのテストの実施内容

5. 達成度の評価、取り組みの結果

本章では開発効率の観点および開発中に発見した不具合から、本書で報告した取り組みについて評価を行う。

5.1. 開発効率に関する評価

表 B-8- 2 に 4.3 において述べた品質目標と課題について示し、その課題を解決するためのテスト実施内容についてまとめた。データバリエーションおよび期待値の確認範囲について、Level1 の品質目標は、人手による作業によって達成をすることができた。しかし、Level2 の段階では、テスト対象となるデータバリエーションの範囲および確認する期待値の範囲が膨大になる。そのため、人手による作業によって Level2 の品質目標を達成することが現実的ではなかった。そこで、機械的にテストデータを生成する方法と機械的に期待値を生成する方法を用いた。

表 B-8- 2　4.3 において述べた品質目標・課題とその課題を解決するためのテスト実施内容

品質目標		課題	テスト実施内容
データバリエーション	Level 1	―	・ディシジョンテーブルを用いてテスト項目を作成 ・半自動でテストスクリプトを生成
	Level 2	・テストパターンが膨大 ・機械的に生成する手法が必要	・ランダムにテストデータを生成 ・ランダムにテストデータを生成するために形式仕様記述から内部状態を取得 ・出力の期待値を形式仕様記述から取得
期待値の確認範囲	Level 1	―	・半自動でテストスクリプトに期待値を記載
	Level 2	・あらかじめ期待値を作成する範囲が膨大 ・機械的に生成する手法が必要	・形式仕様記述から内部状態全体の期待値を自動生成

機械的にランダムにテストデータを生成するには、内部状態を参照しなければならない。例えば、PIN 照合を成功させるためには、IC チップの内部に保持している PIN 値を参照できなければ、成功させるための PIN 値が分からない。そのため、IC チップの内部状態を生成する内部状態生成器が必要である。また、出力結果が正しいことを確認するためにも、出力結果生成器が必要である。期待値の確認範囲について、Level2 の品質目標を達成するためにも、内部状態生成器が必要である。本適用事例では、この内部状態生成器と出力結果生成器として、上流工程において作成した形式仕様記述を流用した。もし、形式仕様記述を流用できなかった場合、新たに仕様書を参照して、内部状態生成器と出力結果生成器を作成す

る必要がある。上流工程において作成した形式仕様記述を流用することで、新たにこれらの生成器を作る必要がないという点において開発コストと開発期間を削減することができた。また、新たに仕様書からこれらの生成器を作成した場合、その生成器の品質を高める必要がある。形式仕様記述は、仕様策定者や設計者や評価者など多くの開発者がレビューを実施しており、上流工程において仕様アニメーションによる検証も行っているため、内部状態生成器および出力結果生成器として品質は確保できていた。これらの生成器の品質という点においても、形式仕様記述を流用することにより効率的に品質を確保することができた。このように、形式仕様記述を用いることによって、Level2 の品質目標を達成するためには開発を行わなければならなかった 2 つの生成器を新たに開発する必要がなくなり、具体的に開発効率を向上させることができた。

5.2. 開発中に発見した不具合から Level2 の品質目標を評価

本開発において、4.2 において述べた Level2 の品質目標を定めて、この品質目標を達成するために 4.3 において述べたテストを実施した。その結果、いくつかの不具合を見つけることができた。大部分の不具合は Level1 の品質目標のテストにおいて、見つけることができる不具合であった。しかし、様々な条件が組み合わさった場合に発生する検出困難な不具合や設計レベルの単体テストを実施したとしても見つけることが難しい不具合など、Level2 の品質目標のテストでなければ検出することが難しい不具合を見つけることができた。

テストのデータバリエーションの Level2 の品質目標を達成するために、ランダムにテストデータを生成するテストを実施した。このテストにおいて発見した不具合は、設定する操作と取得する操作の 2 つの操作において、一貫性がないために正しい動きとはならない不具合であった。設定する操作と取得する操作について、それぞれ単体でテストをしたときには、正しく動いているが、2 つの操作を組み合わせたときに正しい動きとはならないケースであった。また、この設定する操作のバリエーションも複数存在し、ある特殊な条件における設定操作において不整合が発生した。運用観点のテストにおいて、この特殊な条件でのテストを実施していなかったため、データバリエーションについて Level2 の品質目標のテストにおいてこの不具合を見つけた。

この確認範囲の品質目標として Level2 を達成するために、形式仕様記述から内部状態を取得して製品実装の内部状態と比較した。このテストにおいて見つけた不具合は、ある値を設定をしたときに、設定対象の値は正しく設定しているが、設定範囲の誤りによって変更してはならない範囲まで値を設定してしまうような不具合であった。設計レベルの単体テストは実施していたが、この単体テストにおいては Level1 の品質目標である変更した範囲の期待値のみを確認していたため、Level2 の品質目標を達成した仕様に基づくテストでしかこの不具合を見つけることができなかった。この不具合は、本来、変更してはならないパラメータを変更してしまうという市場において影響のある不具合であった。

このように、Level2 の品質目標を定めたテストにおいて、他のテスト工程で見つけること

が難しく、さらに、市場において影響のある不具合を見つけることができた点から、効果的な活動であったと考える。

6. 今後の取り組みと考察

この取り組みを行った事例提供者の見解は以下のとおりである。

今後の取り組みとして、本書において報告した内容からさらに機械処理による自動化の範囲を広げることを検討していきたい。つまり、図 B-8-8 において述べたテストの流れにおいて、人手による作業を減らしていきたいと考える。最終的な目標は図 B-8-8 の「①形式仕様記述」の作成を行えば、Level2 の品質目標を達成するテストを機械的に行うことができるようにすることである。このように、形式仕様記述言語を用いて仕様を記述することによって、自然言語のように記述して終わりであった仕様ではなく、テスト工程において活用することができる仕様とすることができる。さらに、機械処理による自動化によって、今までよりも効率的に品質を高めていくための様々な可能性が開けると考える。

本書では、形式仕様記述フレームワークの設計について具体的な PIN 照合の仕様記述を用いて、具体性に重点をおいて説明を行った。これは、品質を効率的に確保するための出発点は、正しさの基準となる仕様が厳密に書かれていることであると考えたからである。如何に様々な観点において工数をかけてテストを行ったとしても、テストにおいて参照をしている、基準となる正しさの記述が曖昧であったり、誤りを含んでいる場合は、製品の品質を確保したり開発効率を高めたりすることは難しい。高信頼性を達成するためのソフトウェア開発の出発点の 1 つは、レビューやテストにおいて活用することができる厳密な仕様の記述方法と仕様の活用方法について、具体的に考えることであると考える。これらの形式仕様記述の記述方法と活用方法には、一般的な手法はないと考える。それは、開発ドメインの特徴や開発における課題や開発予算や開発期間がそれぞれ異なるため、個別に考えていく必要があるためである。本書において述べた事例報告は、モバイル FeliCa IC チップのファームウェア開発において形式仕様記述手法を適用した 1 つの適用事例ではあるが、高信頼性を達成するための個別のソフトウェア開発について、様々な視点で広く考える機会になるものと考える。

参考文献

[1] SCSK：VDMTools、http://www.vdmtools.jp/

[2] 中津川 泰正 ： FeliCa IC チップ開発への実行可能な形式仕様記述の実践に基づく設計・構成法の提案、学位論文、九州大学、2012

[3] SCSK：VDM++ 言語マニュアル、http://www.vdmtools.jp/

(このページは空白です)

B－9

Orthogonal Defect Classification 分析による欠陥除去と品質の成熟度可視化[1]

1. 概要

ソフトウェア開発に於いて、その成熟度を評価し、品質を向上させるためには、一般的に 2 つの手法が用いられている。ソフトウェア成熟度の評価には、信頼度成長曲線のようにマクロ的に品質を分析する「統計的手法」が用いられ、品質の向上のためには、個別の欠陥を分析し除去する「要因分析的手法」が用いられている。「統計的手法」ではソフトウェア全体の品質を捉えることができるが、開発工程における問題領域を特定し根本的解決を図ることが難しく、「要因分析的手法」では欠陥の発生原因や工程はわかるが、障害件数に比例して、実施が困難となる。この両者の課題を解決すべく、ソフトウェアの障害分析手法である、Orthogonal Defect Classification（以下、ODC）分析（直交欠陥分類）を品質検証に活用した。この手法は、個々の欠陥を直交する複数の属性で分類することにより、効率的に原因・傾向を分析するもので、開発工程の問題箇所を特定し集中して品質改善をすると共に、ソフトウェアの成熟度をより多角的評価できる利点がある。本稿では、ODC 分析の方法論とその有効性、および導入にあたっての取り組みを述べる。

2. 取組みの目的

ODC 分析導入以前に問題となったプロジェクトでは、設計行為と検証行為は互いに関連づけされておらず、検出された障害に対し個別の修正作業が施されたが、欠陥が作りこまれた工程に対する抜本的な品質改善活動を行うまでには至らなかった。

試験終盤になっても障害が出続ける状況下で、品質が担保できず、テストを繰り返すことにより、開発日程に遅延が生じたプロジェクトも散見された。

また、実施した試験によってソフトウェア最終品質を十分に保証したとは言いがたく、試験の総件数のみで感覚的に出荷判断を行う場合もあった。

この問題に対応するために、障害分析として、以下の２つの手法が取られていた。

（1）　「統計的手法」すなわち、障害の出現傾向を全体的に統計分析する手法。総障害件数を信頼度成長度曲線モデルで近似し、残存障害数を推定するもの。

（2）　「要因分析的手法」すなわち、個別障害の原因を一件ずつ調査し対策を打つ手法。

---

[1] 事例提供:オリンパスソフトウェアテクノロジー株式会社　サポートセンター　山崎　隆 氏

「統計的手法」ではマクロ的に現状を理解することはできるが、原因や発生工程の特定ができず、「要因分析的手法」では、個別障害の発生原因や工程はわかるが、障害件数が増大すると分析が困難となり、全体の品質状況を鳥瞰することも難しくなるという欠点もあった。

これら 2 つの手法のジレンマを解く上で前提となる仮説を以下のように考えた。

- 欠陥は一様に分布するのではなく、特定の領域に偏在する。
- 早期に問題領域を特定し対策を打つことにより、その影響範囲を拡散させずに済ませることができる。
- 表層にまず現れる単純な障害の裏には複雑系の障害も潜んでいる。
- 個々の障害を修正するだけではなく、ソフトウェア開発プロセス上の問題工程を特定しプロセス改善することが抜本的な品質向上に有効である。

本事例では、これら仮説をよりどころに、以下の 3 つの課題を解決することを目的としている。

(1) 開発プロセスにおけるどの局面に障害の源泉があるのかを見出したい。
(2) ソフトウェア品質の可視化を行い、残存する障害の傾向を分析したい。
(3) 最終ソフトウェアの品質を担保し、説明責任を果たしたい。

3. 取組みの対象、適用技術・手法、評価・計測

3.1. 対象製品・プロジェクト、適用工程

対象製品・プロジェクト、および適用工程について示す。

(1) 比較的に大規模な組み込み系ソフトウェアおよび IT システムの検証において有効である。具体的に、同社での対象プロジェクトでは、数十万から数百万ステップ規模の医療機器システムに対しその優位性が確認できた。
(2) 開発規模が小さいもの（障害数が限られ、労力をかけずに障害の要因分析が行える規模）に関しては有効性を発揮することはできない。具体的には、数名の体制で短期間に開発を行うプロジェクトにおいては、実装コードやその障害を全体的に見通すことができるので本手法は有効ではない。
(3) 検証業務を対象としているが、設計段階でのデザインレビュー等でも同じ手法が展開可能と考える。

3.2. 方法論（どのような技術・手法を用いて実施したか）

方法論として、Orthogonal Defect Classification（直交欠陥分類）による障害分析を導入した。ODC 分析は、1992 年 IBM Watson 研究所により提唱された障害の定量的分析手法である。排他的（Orthogonal）な複数の属性を用いて分析することにより、障害を多次元のソフトウェア空間領域上に位置づけることできる。

具体的に ODC 分析では検出された障害に直交した複数の属性を付与し、多角的な視点から分析を行う。独自で重複・冗長のない属性を定義し、その属性のみに注目しながら分類することにより効率的に分析を行うことができることを特徴とする。

ODC 分析により得られる結果は、偏在する障害領域を特定するだけではなく、それが作りこまれた開発工程をも特定することが可能となる。

また、ODC 分析は、前述の「統計的手法」と「要因分析的手法」の中間に位置する分析手法とも言える。たとえて言うなら、「統計的手法」は「森全体を眺める」やり方で、「要因分析手法」は「森に生えている一本一本の木を調べる」やり方で、ODC 分析はその中間に位置するものと考えると理解しやすい。（図 B-9-1）

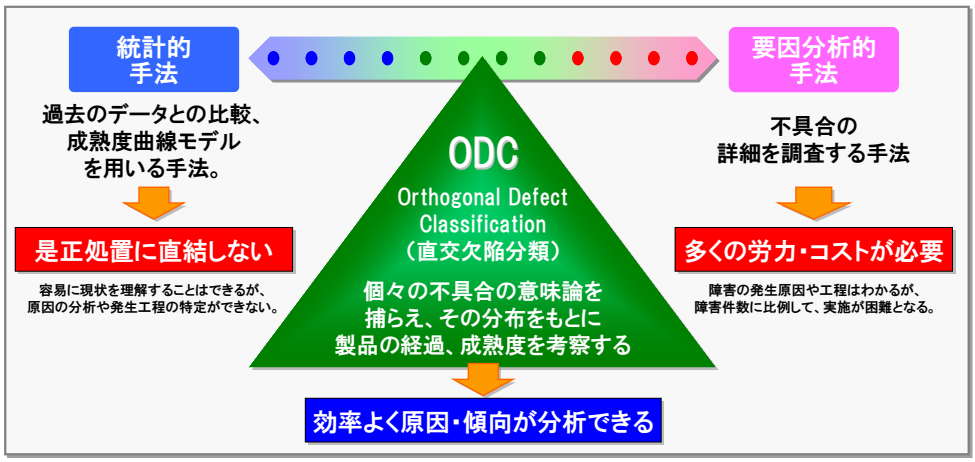


図 B-9-1　ODC 分析の位置づけ

ODC 分析は各障害に複数のタグを付け、そのタグの整理をすることに似ている。このタグが「ODC 属性」に相当する。（図 B-9-2）

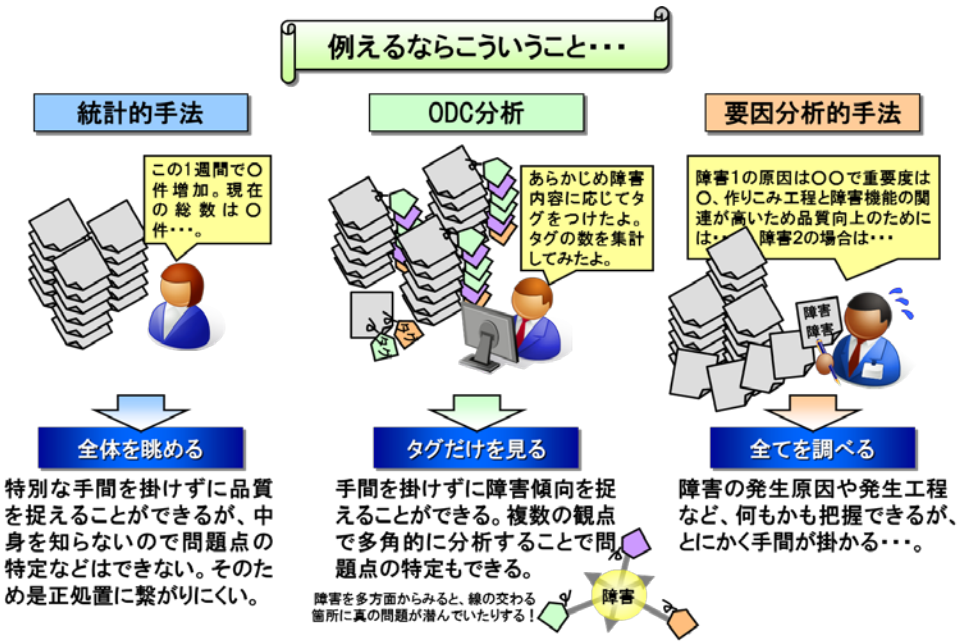


図 B-9-2　ODC 分析の概念

ODC 分析では、障害は直交する属性のマトリクスの中に偏在して存在すると考える。その障害領域により問題の重大性やそれに対する打ち手が変わってくる。(図 B-9-3)

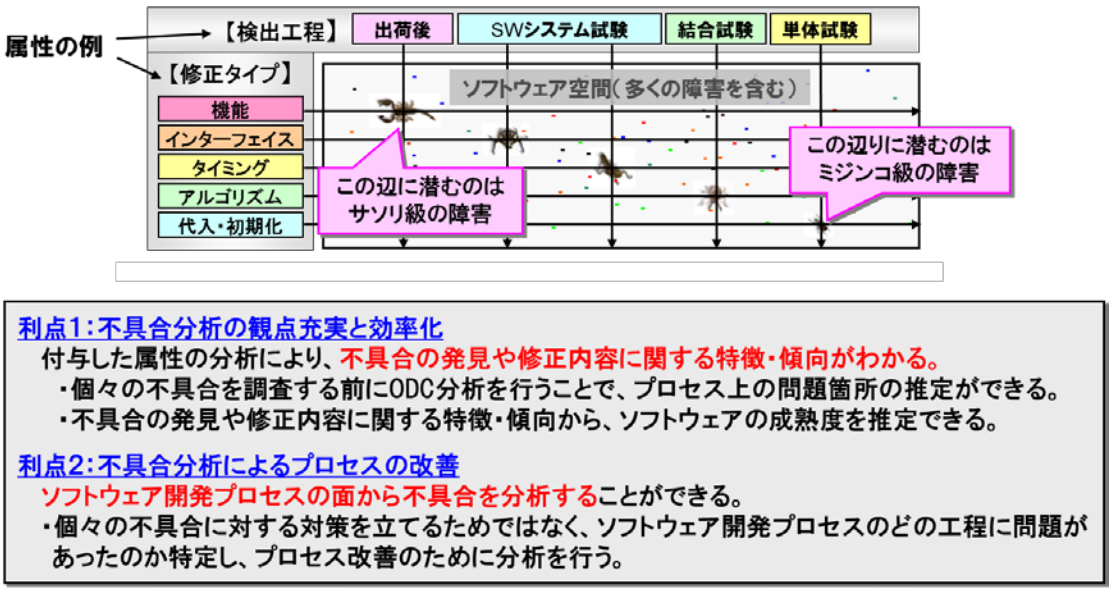


図 B-9-3　ODC 分析の利点

3.2.1. ODC 属性

ODC 分析では、8 つの属性が定義されているが、現在は開発の実態に照らし合わせ、次の 5 属性を採用している。

(1) 検出工程（Defect Removal Activities）
(2) 発生トリガー（Triggers）
(3) 障害タイプ（Defect Type）
(4) 障害実装タイプ（Qualifier）
(5) 修正ソース種別（Age）

各々の ODC 属性は障害の発見者により決定するものと、修正者により決定するものに大別される。(図 B-9-4)

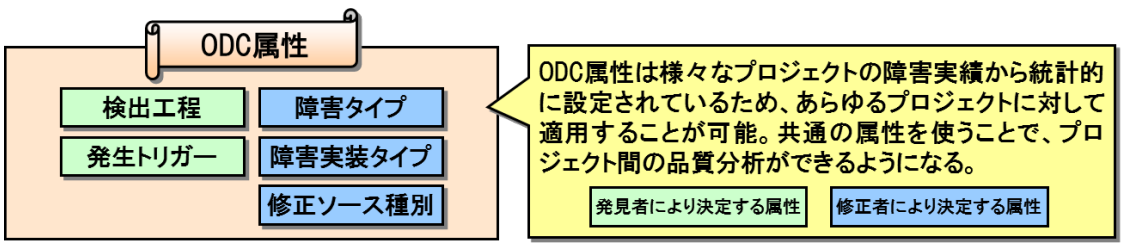


図 B-9-4　ODC 属性

以下、この 5 つの ODC 属性について記述する。

3.2.1.1.「検出工程」

障害を検出した工程を識別する。

どの試験フェーズで障害が発見されたかを分類するものである。

この属性の決定は発見者が行う。（表 B-9-1）

表 B-9-1　「検出工程」

選択肢	意味	備考
結合試験	結合試験中に検出された障害。	小 深刻度 大 検出工程は、いつ障害が検出されたかを示す。 同ランクの障害なら、より下流で検出される方がプロダクトとしては深刻といえる。
SWシステム試験	SWシステム試験中に検出された障害。	
出荷試験	出荷試験中に検出された障害。	
出荷後	ユーザーに納入後に検出された障害。	

3.2.1.2.「発生トリガー」

障害を検出するに至ったきっかけを識別する。

どのような試験をしていた際にでた障害かを分類するものである。

この属性の決定は発見者が行う。（表 B-9-2）

表 B-9-2　「発生トリガー」

選択肢	意味	備考
基本操作(標準設定)	表B-9-3「発生トリガー」の選択肢参照。	小 手順の複雑度 大 発生トリガーは、試験手順の複雑度を示す指標となる。 より複雑な試験が行われているほど、試験は成熟している(しっかり試験されている)といえる。
基本操作(オプション等の変更あり)		
操作順序に依存		
複数機能の相互作用		
負荷・ストレス		
回復・例外		
起動・再起動		
構成		

表 B-9-3　「発生トリガー」の選択肢

選択肢	意味
基本操作 （標準設定）	パラメーター、オプションなどは全く変更せず、単一機能のみを実行したとき。
基本操作 （オプション等の変更あり）	パラメーター、オプションなどを変更し、単一機能のみを実行したとき。
操作順序に依存	複数の機能を特定の順番で実行したとき（各機能を別々に行えば正常動作する場合）、 もしくは単一機能を何回も実行したとき。
複数機能の相互作用	複数の機能が同時に動作しているとき。もしくはその後。
負荷・ストレス	各リソース（メモリーなど）の限界値付近でテストしたとき。
回復・例外	リカバリー/例外処理のテストケースを実行した後。
起動・再起動	シャットダウン、スリープ、ネットワーク遮断、電源遮断などから通常状態に戻した後。
構成	あるソフトウェアをダウンロード/インストールした後。もしくは、あるハードウェアを接続した後。

「発生トリガー」の選択肢の選び方を示す。論理的ステップを踏んで段階的に決定していくと良い。（図 B-9-5）

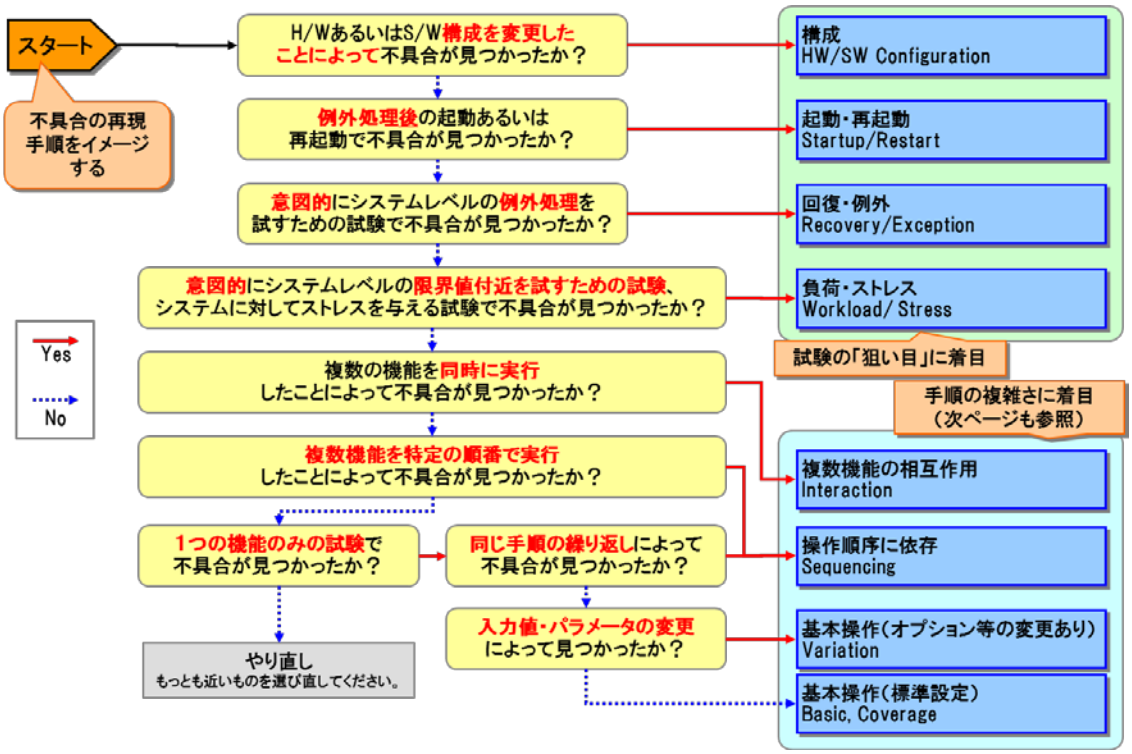


図 B-9-5　「発生トリガー」選択肢決定のための論理ステップ

「発生トリガー」では、基本操作から複数機能の相互作用までの複雑度を規定することで実装の成熟度や試験の網羅度を推し量ることが可能となる。(図 B-9-6)

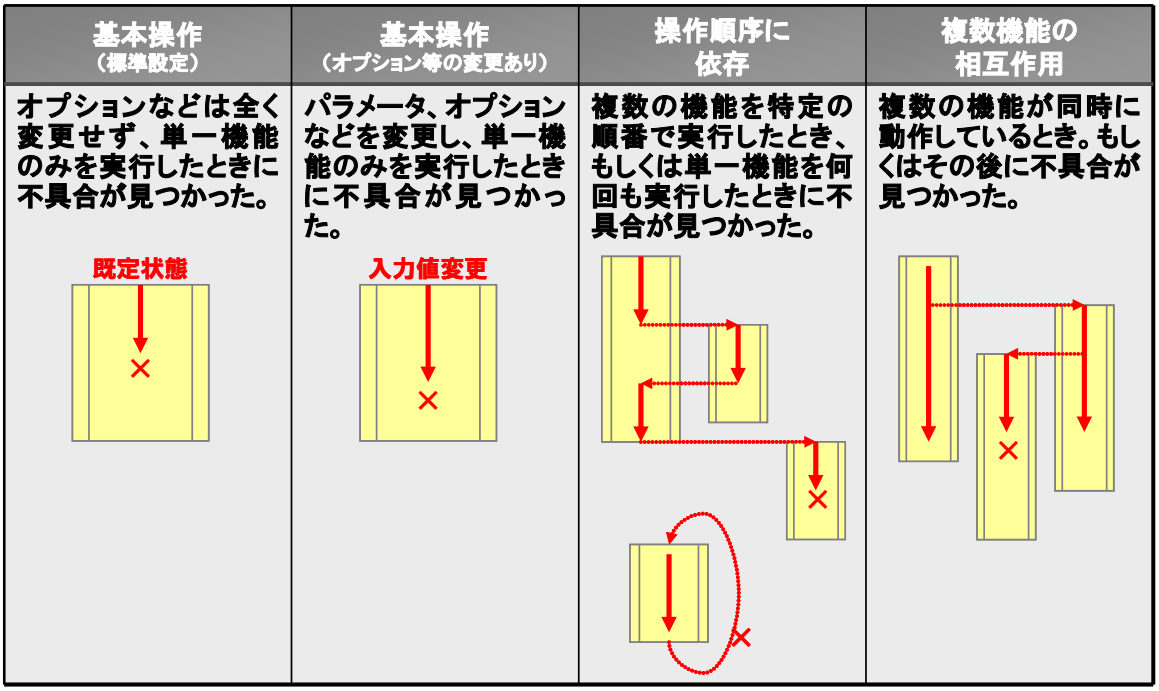


図 B-9-6　「発生トリガー」における手順の複雑さの分類

3.2.1.3.「障害タイプ」

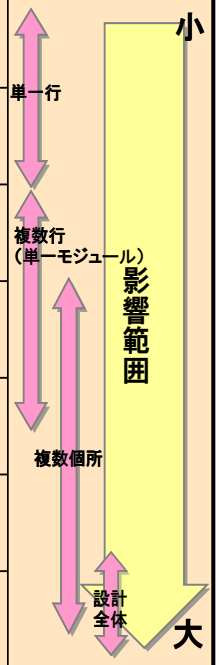
修正内容に基づき、障害を分類する。

この属性の決定は修正者が行う。(表 B-9-4)

表 B-9-4　「障害タイプ」

選択肢	意味	備考
値の代入/初期化	表B-9-5「障害タイプ」の選択肢参照。	小 他への影響度 大 障害タイプは、その障害が他に与える影響を考える上での指標となる。 開発終盤で影響度の大きい障害が多発する場合は注意が必要である。
値のチェック		
アルゴリズム		
共有リソースのアクセス制御		
インタフェース/メッセージ		
関連付け		
機能/クラス		
GUI	表B-9-5「障害タイプ」の選択肢参照。	

表 B-9-5　「障害タイプ」の選択肢

選択肢	意味	具体例	修正・影響範囲
値の代入/初期化	値の代入、初期化の修正が必要な傷害。(ただし、複数のステートメントの修正が必要な場合は、「アルゴリズム」を選択する。)	値の初期化ミス／クラス生成時にコンストラクター呼び出し忘れ、呼び出し結果が正しくない・・・	
値のチェック	条件式におけるデータチェックの修正が必要な障害。(ただし、複数行の修正が必要な場合は、「アルゴリズム」を選択する。)	IF文、switch文の値チェックミス・処理忘れ／メソッド引数のチェック部のミス・処理忘れ・・・	
アルゴリズム	メソッド内、関数内のロジックの修正が必要な障害。	IF文内の処理のロジックミス／リスト構造のサーチロジックのミス／ローカルデータ構造のミス・・・	
共有リソースのアクセス制御	共有リソースのアクセス制御部分の修正が必要な障害。	排他処理ミス／共有メモリ上のデータ更新処理の順番ミス・・・	
インタフェース/メッセージ	モジュール、デバイスドライバー、関数、コンポーネント、オブジェクト間の、パラメータやデータ受け渡しなどにかかわる修正が必要な障害。	インターフェースは数値へのポインターを指定しているが、コードそのものは文字へのポインターと扱っていた場合など。	
関連付け	クラス内のメソッド、データ間の問題、インスタンス化および継承に関する修正が必要な障害。	クラスメソッドがオーバーライドされていて動作が予想と違っていた、インスタンス可数が間違って制限されていた場合など。	
機能/クラス	設計変更を必要とし、機能実現性や(ユーザー、ハードウェアなどに対する)外部インターフェース、グローバルデータの構造に影響する障害。	クラス不足、データサイズが要求に満たない、などの機能実現性に影響する設計変更が必要な場合。	
GUI	リソースデータのみ修正が必要な障害。ソースコードの修正は不要。	GUIリソースの修正など。	

3.2.1.4.「障害実装タイプ」

障害の実装パターンを 3 種類の選択肢で分類する。

この属性の決定は修正者が行う。(表 B-9-6)

表 B-9-6　「障害実装タイプ」

選択肢	意味	備考
誤実装	誤って実装。	多くの障害は「誤実装」を原因とする。
未実装	実装し忘れ。	もし開発終盤になって「実装し忘れ」が検出された場合、プロセスに問題があると推測できる。
余分	余計な実装が障害の原因となった場合。	「余分」が障害の原因となることは少ないが、発生した場合は原因を調べる必要がある。

3.2.1.5.「修正ソース種別」

コードを修正した場合、そのコードの種類を識別する。

この属性の決定は修正者が行う。(表 B-9-7)

表 B-9-7　「修正ソース種別」

選択肢	意味	備考
新規	今回の開発で新規に作成したコードで問題が発生。	一般的な問題。
既存	以前の開発から流用したコードで障害が発生。	すでに出荷済みのソフトウェアにも、潜在的障害があることを示す。
改造/変更	以前の開発から流用したコードを改良した箇所で障害が発生。	大量に検出された場合、影響範囲の確認不足などが考えられる。
二次障害	障害を修正した箇所で別の障害が発生した。	

3.2.2. その他の一般属性

厳密には直交する ODC 属性ではないが、いくつかの障害情報は ODC 属性と同列に扱った。

- 障害の重要度を表す「障害ランク」
- 障害が発生した時期を識別する「発生日」
- インシデントの最終的な対処を識別する「解決状況」
- 障害が発生した箇所（機能、モジュールなど）を識別する「発生箇所」

などがそれにあたる。

3.2.3. ODC 分析による開発プロセス領域評価

障害がどの開発工程に起因するものなのか、ODC 分析により特定することができる。

以下に、障害タイプと障害実装タイプにより、開発工程の問題箇所を特定する考え方を示す。たとえば「機能・クラス」が「未実装」であった場合は、要件・仕様策定の段階で、機能の記述漏れや曖昧な表現があったことが推察される。この問題は重大で、要件定義書、仕様書の見直しを図る対策が迅速に打たれるべきである。また「値の代入・初期化」や「値のチェック」の「誤実装」があった場合は、実装時にコーディングミスが発生したものと考えられ、比較的軽微な障害として分類される。対策としては、実装者のスキルチェック、コードレビューの強化などの対策が取られるべきである。但し、これらの障害は試験フェーズの初期段階で出し尽くすべきもので、「検出工程」の終盤（システム試験など）においてもこのような障害が頻発しているとしたら、プログラムが成熟しているとは言いがたい。(図 B-9-7)

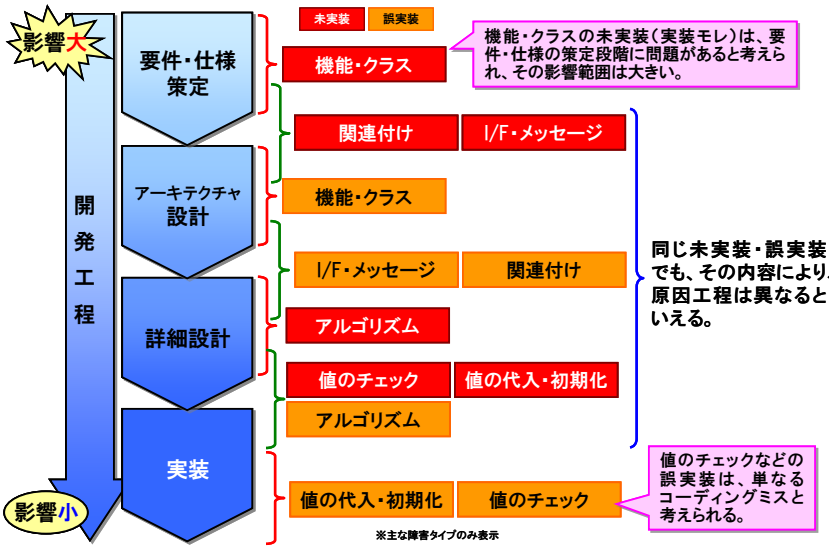


図 B-9-7　ODC 分析による開発プロセス領域の評価

3.3. ODC 分析のステップ論

ODC 分析では多角的なデータが得られるが、一方、様々な情報（ODC グラフ）が散在し最終的な判断を下すことが難しくなる（どのように分析してよいか分からなくなる）ことも懸念された。そのため同社では、ODC 分析のステップを規定したプロセスフローを確立し、それに則った分析及びレポートを行っている。（社内では考案者の名前をとって「タケダメソッド」と呼んでいる。）これにより、ODC 分析の流れが統一され、かつ最終的な評価を一義的に導出することが可能となった。（図 B-9-8）

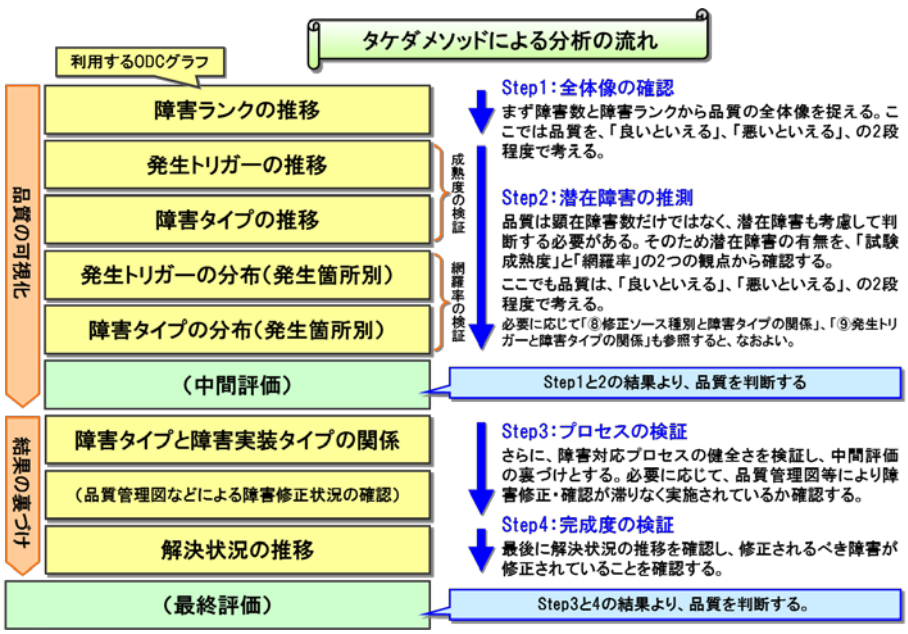


図 B-9-8　ODC 分析のステップ論

4. 取組みの実施、および実施上の問題、対策・工夫

4.1. ODC 分析による開発・検証の成熟度評価

以上、ODC 分析の方法論について記述してきたが、次に「発生トリガー」と「障害タイプ」を例に具体的な ODC 分析の事例を示す。

4.1.1. 「発生トリガー」の事例

検証が充分になされているかを「発生トリガー」にて判断することができる。図 B-9-9 では障害が収束傾向にあってもその障害の大半が基本操作によるものであったケースで、より複雑な障害がまだ内在していることが推測された。試験の内容が充分に深耕されておらず単純な操作が主体的である可能性が高いと判断した。

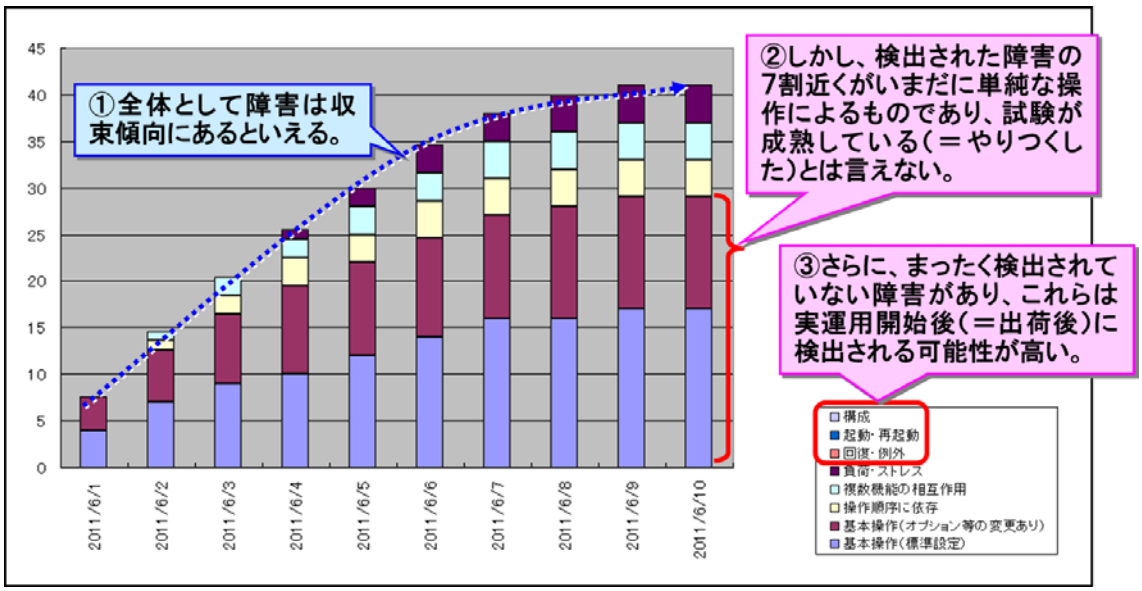


図 B-9-9　「発生トリガー」による試験成熟度の評価事例 1

一方、図 B-9-10 では、基本操作で起こる障害が早い段階で出尽くし、終盤では満遍なく「発生トリガー」が検出できている。また、終盤では障害の発生頻度が減り、収束傾向にある。この事例では、試験が単純な操作から複雑な操作まで実行できており、かつ、発生した障害も取りきれており、プロジェクトが成熟していると判断した。

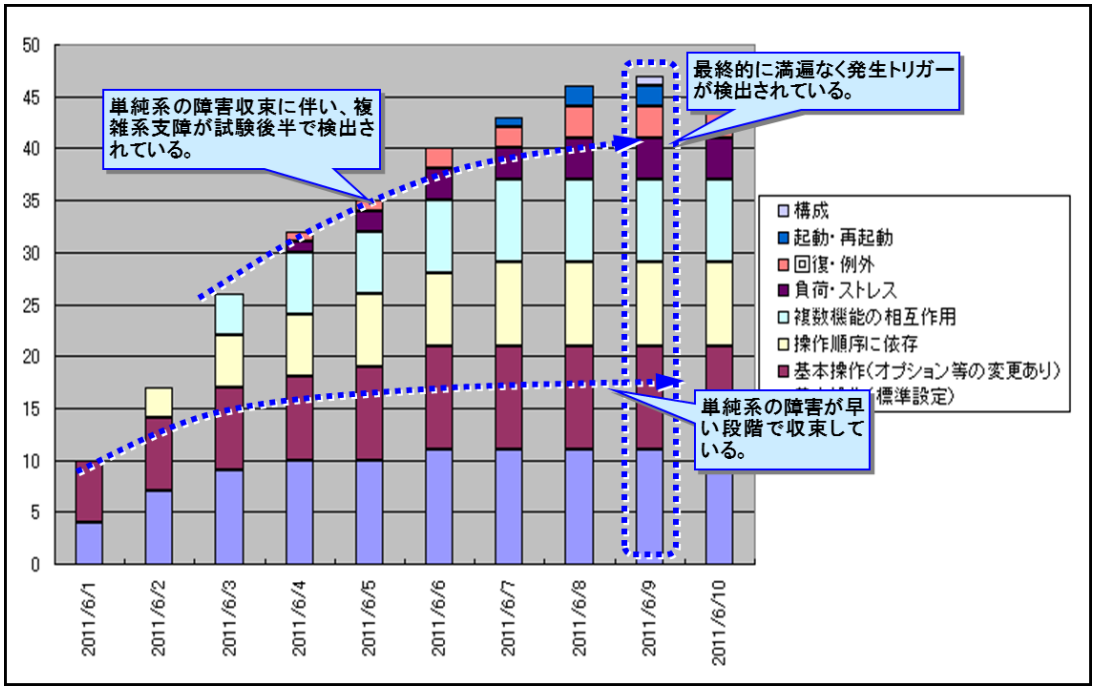


図 B-9-10　「発生トリガー」による試験成熟度の評価事例 2

4.1.2. 「障害タイプ」の事例

実装の成熟度を「障害タイプ」にて判断することができる。図 B-9-11 では、結合試験から SW システム試験へ移行しても障害の出現傾向に違いが見られず、「機能・クラス」などの重大な障害が収束傾向になく、かつ単純なケアレスミスが続いているようなケースで、実装が成熟しているとは言いがたく、今後二次障害が懸念されると判断された。

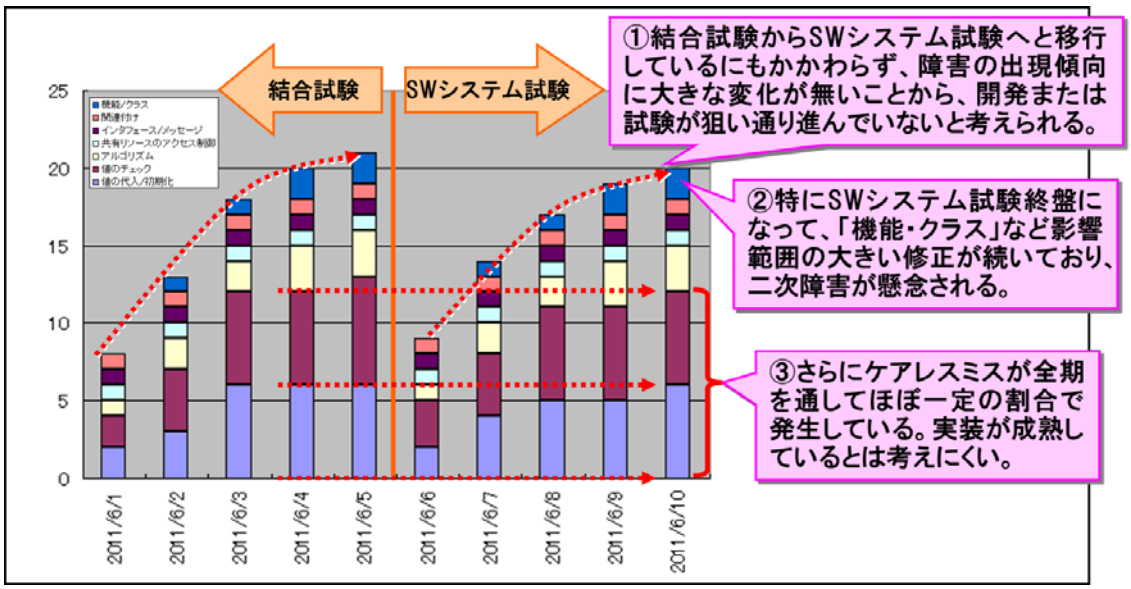


図 B-9-11　「障害タイプ」による実装成熟度の評価事例 1

一方、図 B-9-12 では、結合試験において「値の代入/初期化」、「値のチェック」といったコーディングエラーが取りきれており、システム試験では、より複雑なトリガーも満遍なく検出し、かつ、その数も収束傾向にあるため、実装の成熟度は高いものと判断された。

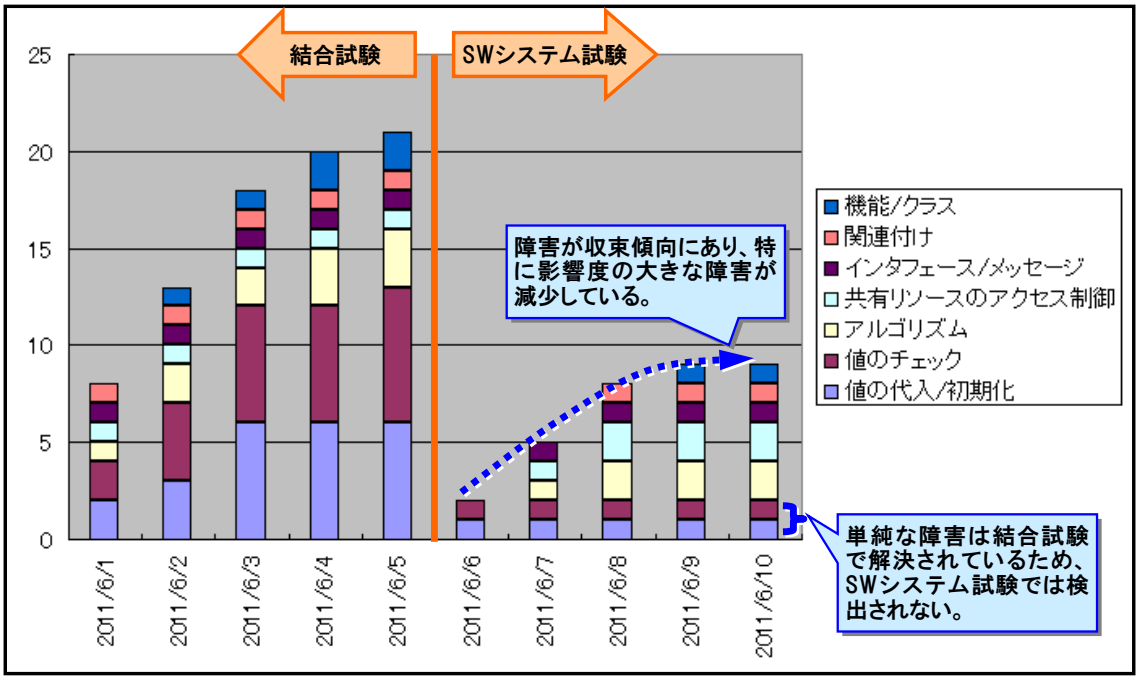


図 B-9-12　「障害タイプ」による実装成熟度の評価事例 2

4.2. ODC 分析を実施するためのプロセス構築

ODC 分析を正しく実行するためには、障害の発見者、修正者が共に、ODC 属性を良く理解し正しく割り付けることが鍵となる。また、その割り付けはプロジェクト毎にぶれることなく一義的に行わなければならない。　そのため、SQA (Software Quality Assurance) が各プロジェクトの障害登録のチェックを行い、指導を強化すると共に、障害管理ツールそのものに手を加え、必ず ODC 属性を正しく付与するような仕組みの構築が求められた。

また SEPG (Software Engineering Process Group) を中心とした「標準開発プロセス・ワーキンググループ」を結成し標準開発プロセスの品質管理サブプロセス定義にて、導入マニュアルやテンプレート、ガイドラインを整備し全社員にイントラ上で開示されるようにした。

4.3. 具体的な取組み内容、活動

2007 年 ODC 分析の提唱者である IBM より直接スキルトランスファーを受け、2008 年よりパイロットプロジェクトでの展開を開始した。ODC 分析導入にあたっては、以下の取組みを行った

(1) 障害管理ツール（Mantis, Trac 等）を改変し、ODC 属性を記述する欄を設ける。
(2) 障害管理ツールに登録されたデータから、エクセルの ODC グラフを出力するマクロを作成。
(3) 勉強会を実施し、全設計者、テスターに属性を理解してもらい、障害管理時に正しく ODC 属性を付与してもらうように指導。
(4) ODC 分析結果をフェーズ移行判定の判定材料とすることを明示。

導入初期段階では上記の活動を行っても、しっかりと属性を付与してもらうことは難しかった。特に修正者が付与する「障害タイプ」の属性は難易度が高く SQA によるガイダンスを強化する必要があった。自発的に属性を付与してもらい、有効な ODC 分析が定常的に行えるまでには、最低 3 サイクルのプロジェクトを回す必要があると感じられた。

4.4. 実施過程でのでき事

実装による基本障害が、システムテストや出荷テスト時に検出されるとそのソフトウェアの品質にまだ問題が内在されていると判断し、プロセスの問題箇所を特定し、集中して改善し、リグレッションテストにより障害が根治したかを判断する、といった緊急対策が施された。

また、システム試験での障害の出現傾向が結合試験と同じ場合は、結合試験自体が完了していないと判断し、一旦システム試験を中止し、結合試験のやり直しを行うこともあった。結果として、その方が正しかったと判断している。

4.5. 人材育成、意識改革等

開発設計者への理解と協力が重要なため勉強会やガイドラインの整備、PM（Project Manager）による障害管理の徹底依頼などを実施し、継続的に導入展開を進めてきた。

初期の段階では、属性を付与することになじめないエンジニアもいたが、ODC 分析では、グラフにより結果を可視化できるため、開発者・検証者の ODC 分析導入に対するモチベーションは比較的高いと考える。

5. 達成度の評価、取組みの結果

ソフトウェアの成熟度を可視化し、改善につなげる一連の流れは、「障害件数を把握」し、その「障害原因を分析」し、「残存障害の予測」を行い、その「情報を蓄積し活用」することにより達成できる。「障害原因の分析」の部分に関し ODC 分析を採用することにより、以下の 4 種の手法で品質評価を体系的に実施できる枠組みが可能となった。（図 B-9-13）

(1) 「障害件数の把握」として『品質管理図』
(2) 「障害原因の分析」として『ODC 分析』
(3) 「残存障害の予測」として『信頼度成長曲線』

(4)　「情報の蓄積と活用」として『品質メトリクス』

特に ODC 分析による障害原因の特定は、開発設計プロセスにおける改善領域を絞り込み、主導的に手を打つことが可能となり、かつ信頼度成長曲線との組み合わせによって、最終ソフトウェアの品質を担保する説明責任を果たすことに寄与した。

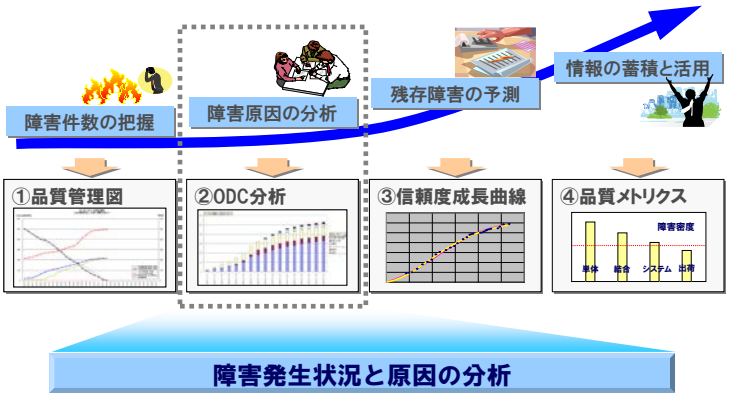


図 B-9-13　品質可視化の手法群

また、ODC 分析では、試験を実施した障害発見者により決定する属性と、開発サイドの障害修正者により決定する属性の両面から、ソフトウェアの品質を多角的に可視化できるため、開発者とテスターの協業が生まれ、ソフトウェアの品質を高めようとするベクトルを一致できる副次的な効果が大きかった。具体的には、とかく開発者は発見された障害の修正を余儀なくされるためテスターを快く思わないケースがあったり、テスターもただ障害を検出すればよいと品質担保の観点で主体性に欠ける一面もあったりするが、ODC 分析では両者が障害の原因に向き合い改善していこうという機運が生まれ結果として組織全体に寄与する波及効果があった。

6. 今後の取組みと考察

ODC 分析手法は、検証領域に留まらず、開発設計段階の設計レビュー、コードレビューに関しても活用することが可能である。レビューで上がる有効指摘に関して、同様な ODC 属性を付与し、分析することにより、開発上流プロセスの問題箇所を特定することができると共に、そのレビュー自体が有効に機能していることを実証することも可能となる。

参考まで、レビュー工程での ODC 属性「発生トリガー」を提示する。

表 B-9-8　レビュー工程における「発生トリガー」事例

発生工程	ODC属性「発生トリガー」	説明
設計レビュー コードレビュー	デザイン適合性	検査者は仕様・要求との整合性を調べていたとき
	ロジック/データフロー	検査者は基本的なプログラム検査、ウォークスルーなどでロジックやデータの流れを調べていたとき
	後方互換性	検査者は旧バージョンとの整合性を調べていたとき
	横互換性	検査者は関連製品・サービスとの整合性を調べていたとき
	同時性	検査者は共有リソースのアクセス制御などを調べていたとき
	内部文書一貫性	検査者は仕様書間、コード内コメント、ヘッダー記述などの整合性を調べていたとき
	言語依存	検査者は実装する言語特有の設計やコードの間違いを調べていたとき
	副作用	検査者はレビュー中のコードの使用によって予見される他のシステム、ソフトウェア、機能、コンポーネントの好ましくない動作を発見したとき
	まれな状況	検査者は設計文書では想定されていない特殊な状況を発見したとき

ODC 分析は、単にグラフ化し可視化すること自体は難易度が高くなく、むしろそれを読み解き、将来を予測し、事前に最善の打ち手を取るといった洞察力の観点で分析者の能力に差が生じる。その部分に対しては、前述のタケダメソッドのような分析の思考プロセスを確立し、ステップを踏んだ検討ができるような体系を構築することが有益であると考える。今後、開発者、試験担当者や SQA の暗黙知的な思考過程を精査し、より良い形式知体系の構築が期待される。

参考文献

[1] Ram Chillarege, Indepal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnn-Process Measurements, IEEE Transaction on Software Engineering, Vol. 18, November 1992

[2] M.Butcher, H.Munro, T.Kratschmer, Improving software testing via ODC: Three case studies, IBM System Journal Vol.41, No.1, 2002

[3] ソフトウェア品質保証知識体系ガイド（SQuBOK Guide）P322-324

B－10

モデル検査の適用による上流工程での設計誤りの発見 [1]

1. 概要

ソフトウェアのバグの中には、実行時の稀なタイミングでのみ発生するバグや条件の特殊な組み合せによってのみ発生するバグ等、テスト工程での発見が困難なものが存在する。こうしたバグは、設計工程における設計誤りが原因である場合も少なくない為、テスト工程の対策だけでは多大なコストが発生してしまう。

こうした問題の対策として、モデル検査の適用が考えられる。モデル検査によって上流工程で設計内容を網羅的に検査し、下流工程への設計誤りの流出を防止することで品質を確保できる。特に、組込みシステムでは、機能的な品質に加え、信頼性や安全性といった品質が強く求められることから、網羅的な検査が可能なモデル検査に対するニーズは高い。

株式会社 東芝 ソフトウェア技術センターでは、組込みシステム開発にモデル検査を適用し、品質の確保に取り組んでいる。本事例では、モデル検査を適用する為に行った同社の取り組みにおける工夫について紹介する。

2. 組込みシステム開発における問題意識とモデル検査への期待

2.1. 組込みシステム開発における問題意識

組込みシステム開発ではソフトウェアの大規模化、複雑化が進んでおり、特に、制御が複雑化するケースが増えている。その結果、設計工程における誤りに起因したバグが問題となっている。具体的な設計誤りとしては、例えば、仕様の変更やシステムの統合によって追加された機能が他の機能と意図しない相互作用を起こし、不正な結果を導く場合があることが挙げられる。

設計誤りに起因するバグを防ぐ為には、機能追加の例であれば、それぞれの機能が動作する条件を精査し、競合状態に陥らないように制御する必要がある。しかし、大規模になればなるほど複雑な制御が要求される為、競合状態に陥らないようにする為の制御の間違いや必要な異常処理の抜け漏れなどの設計誤りをレビューだけで網羅的に確認し、品質を確保することは困難になっている。その一方で、これらの設計誤りを見逃してしまうと重大な故障を招く可能性がある為、開発期間内に確実にこれを発見する必要がある。

こうした背景から、テスト工程で多くのテストを実施することで設計誤りも含めたバグを

[1] 事例提供: 株式会社 東芝　ソフトウェア技術センター　鷲見　毅 氏

発見し、品質の確保に努めているのが実情である。しかし、テスト工程で設計誤りを発見すると、後戻りが大きくなって修正に必要なコストが増大してしまう。その為、設計工程において、レビュー以外の方法で確実に設計誤りを発見、修正することが必要である。

2.2. モデル検査への期待

テスト工程でバグを発見した場合、まずバグの再現確認を行う。その上で原因の究明を行い、設計誤りが原因であると判明した場合は設計の見直しを行って、リグレッションテストで修正の完了を確認する。しかし、意図しない相互作用によるバグは、実行のタイミングによっては発生しない場合がある等、容易に再現できない場合が多い。その為、設計レビューやテストでの発見は容易でない。これに対し、モデル検査は検査対象となるシステム（以降、検査対象システム）の動作を網羅的に検査し、どの動作においてバグが発生したのか容易に特定できる。そこで、特にタイミングに起因するバグについて、テスト実施コストや後戻りコストを低減する目的でモデル検査を用いることとした。

モデル検査は形式手法のひとつで、検査対象システムの振舞いをモデル化し、その性質を検査する技術である。図 B-10-1 に示すとおり、モデル検査器は、検査モデルと検査式を入力として、検査モデルが検査式を満たさない動作を反例として発見し、実行ログとして出力する。反例が発見されない場合は、検査モデルが検査式を満たすことが保証される。

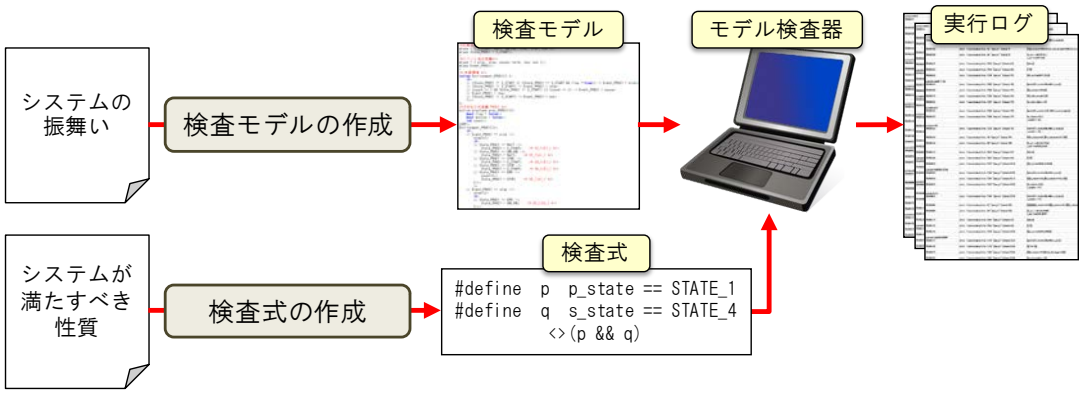


図 B-10-1　モデル検査の手順

ここでの検査モデルとは、システムの振舞いを表す有限オートマトンであり、状態遷移図等で表される情報をモデル検査器で解釈可能な専用言語を用いて厳密に記述したものである。もうひとつの入力である検査式は論理式であり、例えば、“機能 A と機能 B は同時に実行されない”のようなシステムが満たすべき性質を、時相論理を用いて記述したものである。

モデル検査器の 2 つの入力は、モデル検査を実施する検証者が作成する。検査モデルは設計書にシーケンス図や状態遷移図として記述されているシステムの振舞いを入力として、検査式は要求仕様書に記述されるシステムが満たすべき性質を入力として作成する場合が多い。従って、モデル検査は設計工程で実施することが可能である。またモデル検査器は、入力さ

れた検査モデルにデッドロックやライブロック等のあってはならない動作が含まれないか、検査モデルが検査式を満たしているかを、網羅的かつ自動的に検査することができる。こうした特徴から、設計工程における設計誤りの発見にモデル検査を適用することが効果的である。

モデル検査器には幾つかの種類がある。本事例では、組込みシステムの振舞いを記述するのに十分な専用言語を持つモデル検査器の中で、比較的検査モデルを作成しやすいSPIN[1][2]を用いることにした。

3. モデル検査適用における課題と工夫

3.1. モデル検査適用の課題

モデル検査の適用にあたり、実際の開発プロジェクトのドキュメント（要求仕様書、設計書）を用いてモデル検査を行うことが可能であるか、モデル検査を行う上での難しさは何かを分析し、技術的な課題と開発プロジェクトの体制に関する課題に分類した[3]。本稿では技術的な課題とその解決について説明する。

分析によって明らかになった技術的な課題は、図 B-10-2 に示す 3 つである 。以下で、技術的な課題の詳細について述べる。

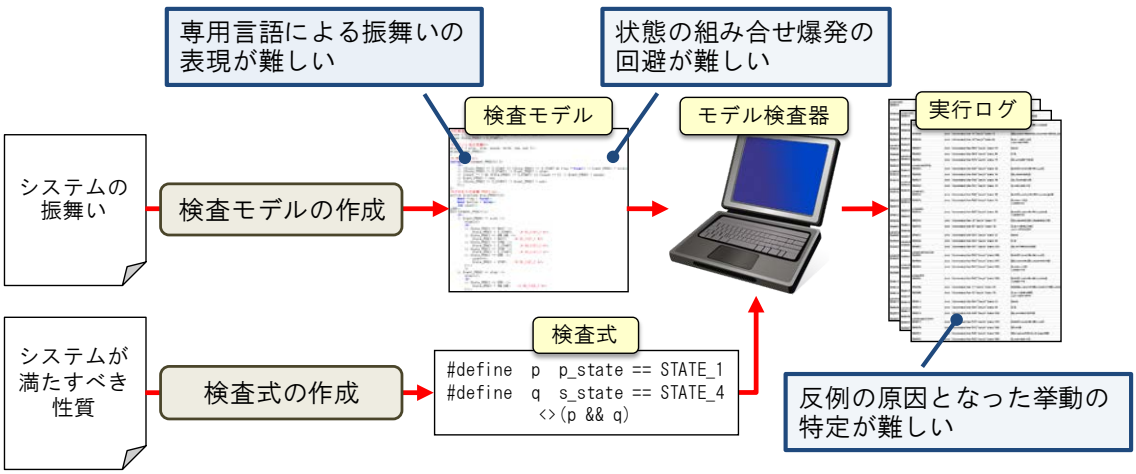


図 B-10-2　モデル検査適用の課題

1 つ目の課題は、検査モデルの作成そのものの難しさであり、これには 2 つの難しさがある。まず、検査モデルを作成する為の専用言語を習得しなければならないことである。SPINが入力とする検査モデルは、Promela（Process Meta Language）という専用言語で記述する必要がある。その為、検査モデルを記述する為には Promela を習得しなければならない。もう 1 つの難しさは、検査対象システムの振舞いと Promela とを対応付けることの難しさである。一般的に、ドキュメントは自然言語や UML 等で記述されているが、それらの記述をPromela にどう対応付け、どう記述すれば良いかは定まっていない。その為、検証者によっ

て作成される検査モデルが異なったり、最悪の場合、ドキュメントに記述された検査対象システムの振舞いを Promela で正しく表現できなかったりする問題が起きる。こうしたことから、検査モデルを作成する上での何らかの方針を定めなければ、正しい検査モデルを作成することが難しい。

2 つ目の課題は、状態の組み合せ爆発に対する対策である。モデル検査器は、検査モデルに記述された処理の中から、その時点で実行可能なものをひとつ選択して実行し、検査式に違反しないか検査する。これを、検査モデルに記述された全ての処理に対して行うことで、網羅的な検査を実現している。しかし、検査モデルが複雑になればなるほど実行可能な処理の数は増加し、どの処理をどの順序で実行するかという組み合せの数も爆発的に増加する。その結果、検査しなければならない状態の組み合せ数が増加して、モデル検査器が検査を完了できない事態に陥ってしまう。その為、状態の組み合せ爆発を回避する手段を用意しなければ、モデル検査による検査結果を得られなくなってしまう。

3 つ目は、モデル検査器からの反例の出力についての課題である。SPIN では、反例が発見されると、その時の実行ログが出力される。設計誤りを発見、除去する為には、実行ログから反例の原因となった処理を特定し、対応する箇所を修正しなければならない。しかし、実行ログはテキストベースの処理の羅列になっている為、原因の特定には向かない形式になっている。実行ログから反例の原因を特定する為に、実行ログの内容を原因の特定に適した形式に変換する必要がある。

3.2. 適用における工夫

3.1 で挙げた適用の課題に対して、それぞれ以下のように対策を行い、モデル検査の適用を容易にした（表 B-10-1）。以降で、それぞれの課題への対策の詳細を説明する。

表 B-10-1　適用の課題とその対策

課題	対策
専用言語による検査モデル作成が難しい	モデル作成方針を定義
状態の組み合せ爆発の回避が難しい	状態縮約手法の開発
反例の原因特定が難しい	反例チャートの作成

3.2.1. モデル作成方針を定義

モデル作成方針では、検査モデルがシステムの振舞いを正しく表現するように、検査モデルを作成する手順と Promela での記述方法の 2 つを定めた。

作成手順は、最初にシステムの振舞いを状態遷移表で記述し、状態遷移表から Promela を作成することとした。組込みシステムはイベント駆動型のシステムが多く、その振舞いを表

す為に設計書では状態遷移図が用いられることが多い。しかし、状態遷移図は可読性等を考慮して暗黙知となっている遷移や自己遷移を省略することがある。設計誤りを発見する為には抜け漏れの無い記述が必要になるが、状態遷移図では図が煩雑になる為、抜け漏れの発見が難しい。そこで、図 B-10-3 のような状態遷移表のテンプレートを定め、これらの情報を全て埋めてから Promela を作成することで、省略されている部分を明示的に記述できるように工夫した。こうすることで、設計書に記述されたシステムの振舞いに抜け漏れがある場合は、この段階で容易に発見することができる。

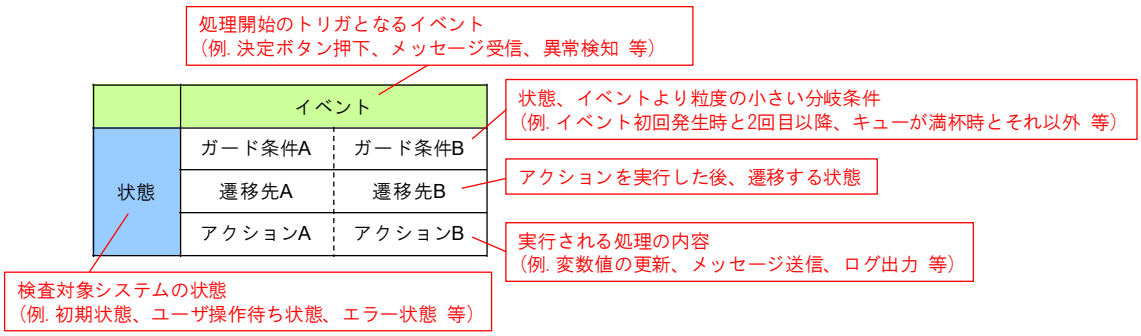


図 B-10-3　状態遷移表によるシステムの振舞いの記述

Promela での記述方法では、上記のように記述された状態遷移表の内容を Promela でどう記述するかの対応付けを図 B-10-4 のように定義した。図 B-10-4 に示す対応付けは、我々がモデル検査を適用しようとしている組込みシステムの特徴を考慮している。考慮すべきと判断した特徴は、以下の 3 つである。

(1)　複数の機能が並行動作する
(2)　各機能は周期実行される
(3)　イベントは各機能とは独立して発生する

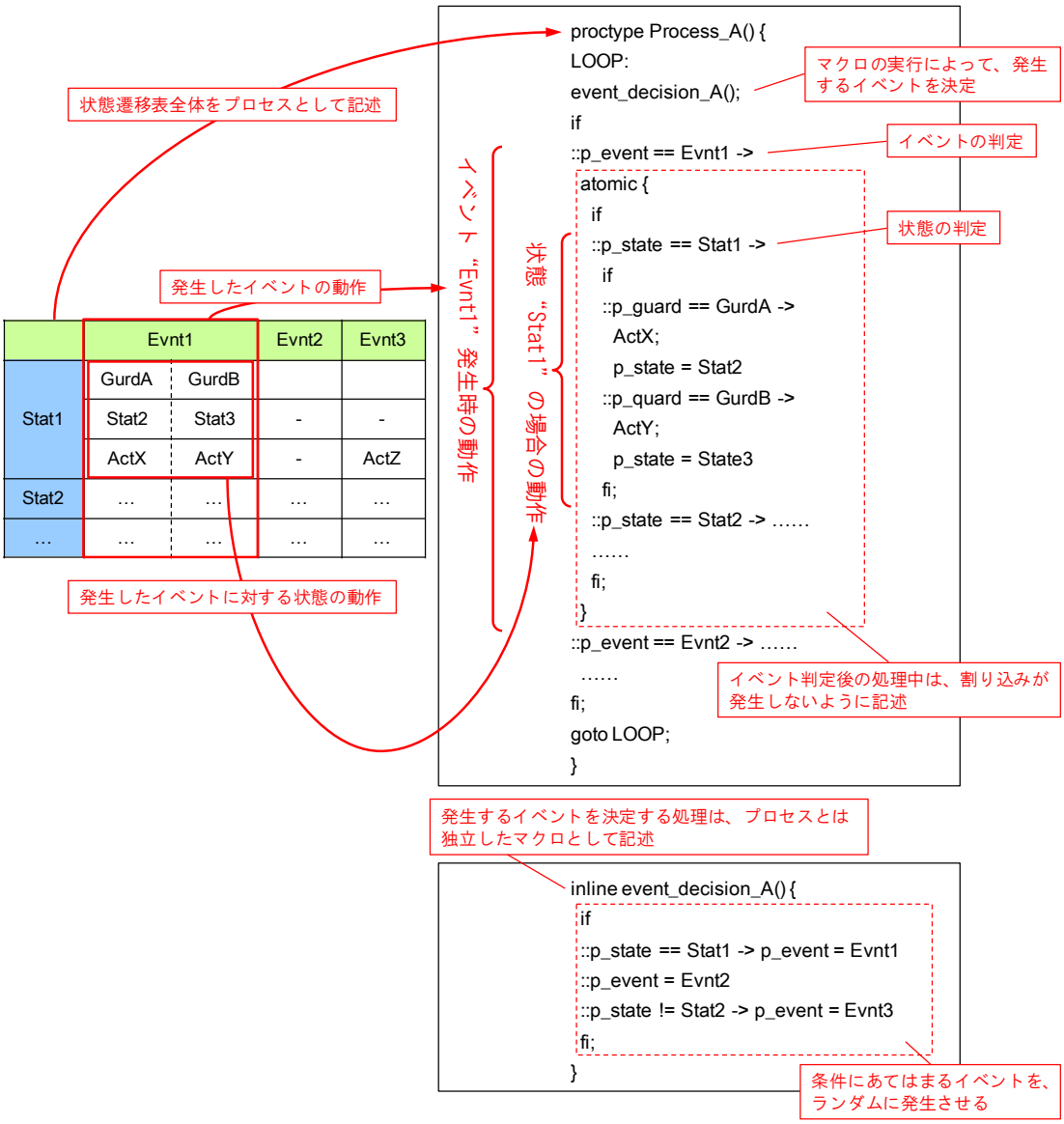


図 B-10-4　状態遷移表から Promela への対応付け

組込みシステムでは、複数の機能が並行動作する場合がほとんどである。SPIN では、並行動作する可能性を持った一連の処理をプロセスで表現する。そこで、各機能の振舞いをそれぞれ状態遷移表で表し、それらを各々プロセスとして記述することで、機能の並行動作を表現している。

また、組込みシステムの機能は、周期実行される場合が多く、周期ごとに機能外部からの入力を受け取り、それに応じた処理を周期内で実行する。こうした動作を表現する為に、プロセスの先頭でイベントを決定し、イベントに応じた処理を実行するように Promela を記述することとした。また、イベントの判定を行った後は、処理が完了するまで他のプロセスからの割り込みが発生しない記述とした。周期実行の場合、同一周期内で何度もイベントを確認することはほとんど無く、一旦、イベントを判定した後、そのイベントに応じた処理を周

期内で完了させるという動作をする。その為、こうした特徴を Promela の記述方法にも反映させた。

最後に、イベントの発生条件は各機能に依存しない場合がほとんどなので、プロセスとは別に動作させる必要がある。そこで、条件に応じたイベントの発生をプロセスとは独立したマクロが行うように Promela の記述方法を定めた。

こうしたモデル作成方針を定めることで検査モデル作成の難しさを緩和し、同時に検査モデルそのものの品質を一定に保っている。

3.2.2. 状態縮約手法の開発

検査対象システムの振舞いに影響を与える外部環境（ユーザや外部システム等）が存在する場合に、状態の組み合せ爆発が起きるが、これを回避できる状態縮約手法を開発した。外部環境が存在する場合、その影響を検査モデルに反映させる為に外部環境を状態遷移表で記述すると、検査モデルが複雑化して組み合せ爆発を発生させてしまう。一方で、外部環境の影響を反映させなければ、検査モデルは検査対象システムの振舞いを正しく表現できなくなってしまう。

この時、検査対象システムと外部環境をそれぞれ個別の状態遷移表として記述すると、2つの状態遷移表が並列して動作する検査モデルとなる。この為、検査対象システムが外部環境の変数を参照せず、外部環境の状態遷移のみが単独で起きるような場合も検査すべき状態としてカウントされてしまう（図 B-10-5）。

検査対象システムの状態遷移表

	P_Evnt1	P_Evnt2	P_Event3	P_Event4
P_State1				
	P_State2	-	-	P_State3
	-	-	-	-
P_State2				
	-	P_State3	P_State1	-
	-	-	-	-
P_State3				
	P_State2	-	-	P_State1
	X = Ex_Val	-	-	X = Ex_Val

外部環境の状態遷移表

	Ex_Evnt1	Ex_Evnt2
Ex_State1		
	Ex_State2	-
	Ex_Val = 1	-
Ex_State2		
	-	Ex_State1
	-	Ex_Val = 5

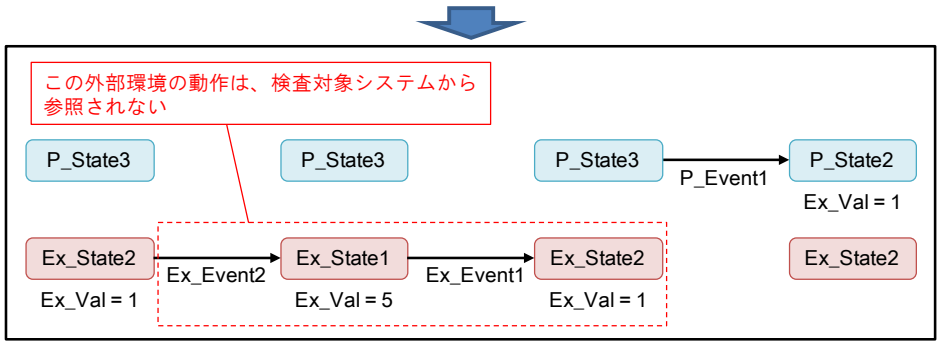


図 B-10-5　検査結果に影響しない実行列

そこで、適切に状態の組み合せ数を減らす為に、状態縮約のアルゴリズムを開発した。その考え方は、図 B-10-5 に示したような検査結果に影響しない実行系列を除外するというものである。参照されない外部環境の動作を含んだ場合のモデル検査の検査結果は、それを含まない場合と同じになる。この点に着目し、外部環境を状態遷移表で作成するのではなく、検査対象システムから参照される変数として宣言し、それを図 B-10-6 で示すように Promela の記述に追加することとした。こうすることによって、参照される外部環境の全ての値が検査対象システムの状態遷移表に条件分岐として網羅される。その一方で、検査対象システムから参照されない外部環境の動作は、組み合せから除外される為、組み合せの数を減らすことができる。

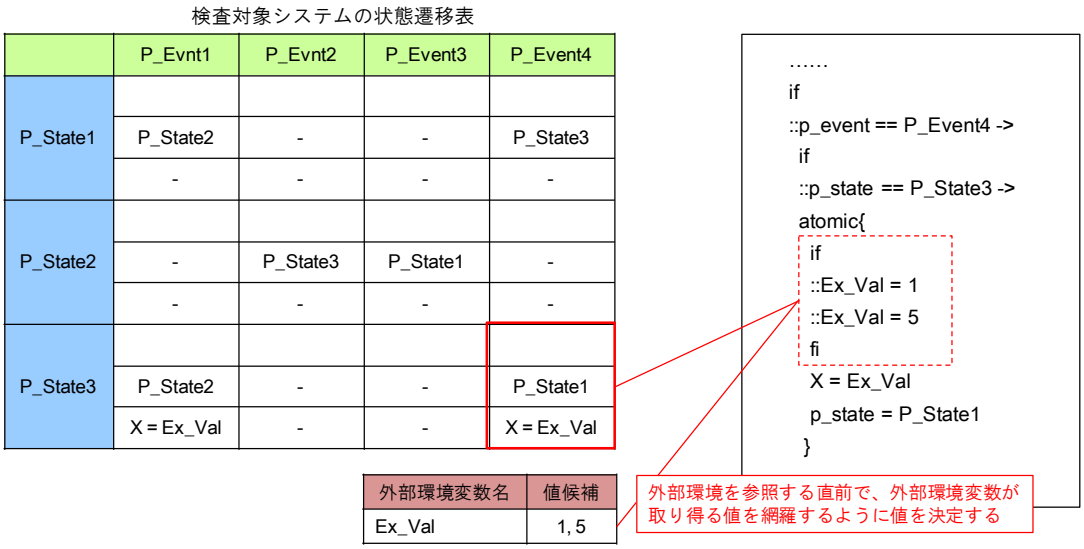


図 B-10-6　状態縮約手法

この状態縮約手法を、表 B-10-2 に示す規模の検査対象システムと外部環境に適用した結果を、表 B-10-3 に示す。表 B-10-3 にあるように、状態縮約手法を実施しない場合と比較すると約 10 分の 1 まで状態の組み合せを減らすことができており、検査対象システムがこの程度の規模であれば、状態の組み合せ爆発を起こすことなくモデル検査を行える。また、DB やネットワーク、別システム等の外部環境の影響を全く受けずに独立して動作するシステムは、ほぼ皆無と言って良い。その為、参照されない外部環境の動作を適切に除外するこの状態縮約手法は、多くの場合で有効と考えられる。

表 B-10-2　状態縮約手法を適用した対象の規模

	状態数（個）	イベント数（個）	遷移数（個）
検査対象システム	10	14	467
外部環境	2	3	3

表 B-10-3　状態縮約手法の効果

	状態の組み合せ数（個）	モデル検査時のメモリ使用量（Mbyte）
状態縮約を実施せず	86684846	1163.475
状態縮約を実施	7656871	501.659

3.2.3. 反例チャートの作成

反例チャートの作成は、SPIN から出力される実行ログを、プロセスの状態、発生したイベント、変数の値を一覧できる形式で整理し、1 ステップの処理毎にタイミングで揃えて表示することで可読性を向上させた（図 B-10-7）。

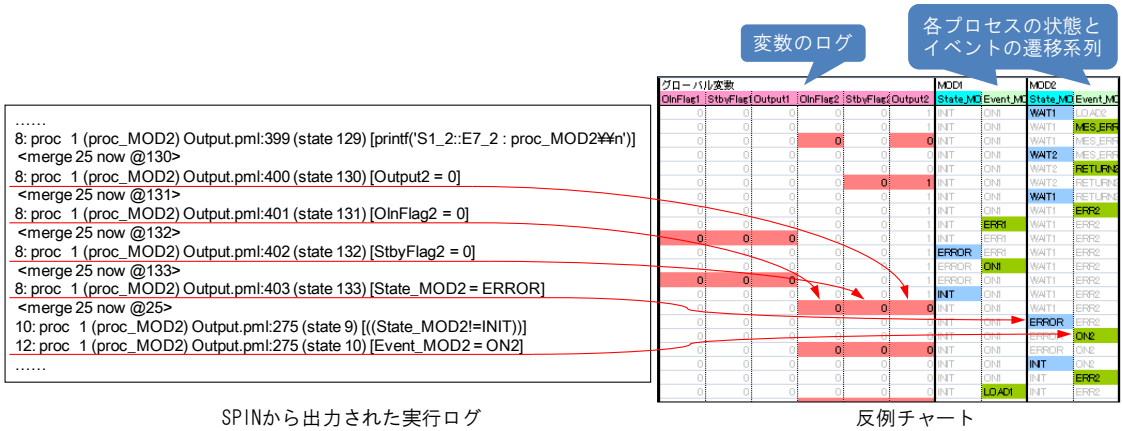


図 B-10-7　反例チャート

反例チャートでは、各ステップで実行された処理の結果をハイライトすることで、各ステップにおける処理内容の把握を容易にしている。また、1 ステップごとのプロセスの状態、発生したイベント、変数の値を全て表示することで、例えば、イベントが発生したステップにおけるプロセスの状態や変数の値を把握することも容易にしている。こうした工夫によって、反例チャートを辿ることでプロセスの状態や変数の値がどんなタイミングで更新されて反例に至ったかが把握しやすくなり、反例の原因特定が容易になっている。

3.3. 適用の効果

3.2 で述べた工夫によって、組込みシステム開発にモデル検査を適用し、実際に設計誤りを発見することができている。発見した設計誤りには、「複数の機能からそれぞれ出力される制御信号が、特殊な条件において競合する」、「マスタ - スレイブの関係を維持して動作すべき複数のスレッドが、あるタイミングで同時にマスタになる場合がある」のようにシステムの振舞いに対する影響が大きく、また発生する条件が稀なものも含まれている。発見した設計誤りに対しては、検討から漏れてしまっていた動作を追加したり、条件の見直しを行ったりすることで、設計工程で修正を加えている。このような稀な条件によって発生するバグをテスト工程で発見した場合、バグの再現確認に多くのコストが必要になる。これに対してモデル検査では、バグが発生する場合の動作を反例チャートによって容易に確認することができる。こうした、テスト工程におけるバグの再現確認のコスト、設計工程まで戻って修正を行うコストを考えると、モデル検査の適用によって 1 プロジェクトあたり平均して、テスト工程の約 30%のコストを低減できた。

またこれらの工夫は、ツールによる自動化を行っている。ツールには、状態遷移表から検査モデルを自動生成する機能、状態縮約の機能、反例チャートを作成する機能を実装している[4]。ツールによる自動化を行ったことで、状態遷移表が記述できれば、モデル検査についての専門的な知識が無くともモデル検査を行うことができるようになった。

4. まとめと今後の課題

本稿では、組込みシステムの開発にモデル検査を適用し、設計誤りを早期発見する為に行った工夫について紹介した。

自身の状態と外部のイベントによって処理を決定する組込みシステムの振舞いを状態遷移表で記述し、状態遷移表から検査モデルを作成する手順を定めた。状態遷移表を用いることで、検査対象システムの振舞いを抜け漏れ無く、厳密に記述することが可能になる。その上で、状態遷移表から検査モデルを作成するモデル作成方針を定め、これに従って検査モデルを作成することで検査モデルの品質を一定に保てるようにした。さらに、状態縮約手法と反例チャートによりモデル検査の実施を容易にした。こうした技術的な課題を解決する為の工夫によりモデル検査の適用が可能となり、設計工程で誤りを発見、修正できるようになった。

一方で、開発プロジェクトの体制に関する課題は十分に解決できていない。例えば、モデル検査の適用は主に設計工程で行う為、従来型の開発プロセスと比較して上流工程のコストが増大する。その為のリソースをどのように確保するか、またモデル検査を誰が行うべきかといった体制面での課題がある。こうした体制面に対する対策が今後の課題である。

参考文献

[1] SPIN Model Checker, The: Primer and Reference Manual, Gerard J.Holzmann, Addison-Wesley Professional(2003)

[2] Principles of the Spin Model Checker, Mordechai Ben-Ari, Springer London(2008)

[3] 実用化に向けたモデル検査適用手法の開発、池田信之、今村紀子、高田沙都子、東芝レビューVol.62 No.9(2007)

[4] モデル検査技術による上流工程での効率的な設計検証、高田沙都子、森奈実子、村田由香里、東芝レビューVol67 No.11(2012)

B－11

上流工程の要求を効率的に閉ループシミュレーションする次世代 SILS の開発 [1]

1. 概要

本編では、上流工程でのソフトウェア検証に関わる取組みとして、トヨタ自動車株式会社（以下、同社とする）の事例を紹介する。

大規模化と複雑化が急速に進む自動車制御系開発は、コンカレント開発へ移行が進んでいる。開発と検証を効率的に行うためにプラントモデルの入出力に関して柔軟な設計ができ、実用的なシミュレーション速度を実現できる必要があった。

現在広く普及している閉ループシミュレーション技術の HILS （Hardware-In-the-Loop Simulation）は、Electronic Control Unit (ECU) 制御系の下流の工程の開発プロセスで用いられている。より高度な要求に対してより効率的に制御系の開発を進めるためには、更に上流の工程から閉ループシミュレーションを活用したい。そこで、MILS（Model-In-the-Loop Simulation）機能を内包し、かつ SILS（Software-In-the-Loop Simulation）としての主要な機能をほぼ網羅した次世代 SILS を開発した。

それによって、正確な割り込み処理実行タイミングと高速なシミュレーション速度の両立を実現した。本事例は、2012 年 10 月 4 日に自動車技術会秋季学術講演にて発表したものを参考にしている。

2. はじめに

近年の自動車開発では、燃費向上・排ガス規制対応・車両運動性能向上の要求を全て満たすために、制御系の大規模化と複雑化が急速に進んでいる。そのような状況の中でも品質を確保しつつ開発の期間やコストが増大することを避けるために、開発プロセスとしては、エンジンやトランスミッションなどの実機の完成を待たずして制御系の開発を進めるコンカレント開発への移行が進んでいる。コンカレント開発では、実機が存在しない状況であっても制御系の開発を進めるために、実機の代わりとなるモデル (プラントモデル) が必要であり、更にそのプラントモデルを制御系と結合して閉ループシミュレーションを行うための技術も必要である。

現在広く普及している閉ループシミュレーション技術としては Hardware-In-the-Loop Simulation (HILS) が挙げられるが、HILS は Electronic Control Unit (ECU) を用いるこ

[1] 事例提供: トヨタ自動車株式会社 /富士通テン株式会社　藤原　靖尚 氏、伊藤　久弘 氏 / 魚住　晴長 氏、福岡　亘二 氏

とを前提としており、制御系の開発プロセスとしては下流の工程である。冒頭で述べたようなより高度な要求に対してより効率的に制御系の開発を進めるためには、更に上流の工程から閉ループシミュレーションを活用したい。その技術として、制御系に制御ソフトだけを用い、HILS の前工程で利用可能な Software-In-the-Loop Simulation (SILS) や、更に上流で制御モデルを用いる Model-In-the-Loop Simulation (MILS) が存在する。

SILS の構成は、制御アプリケーション専用型[1]もしくはマイコンエミュレーション型[2]が一般的である。

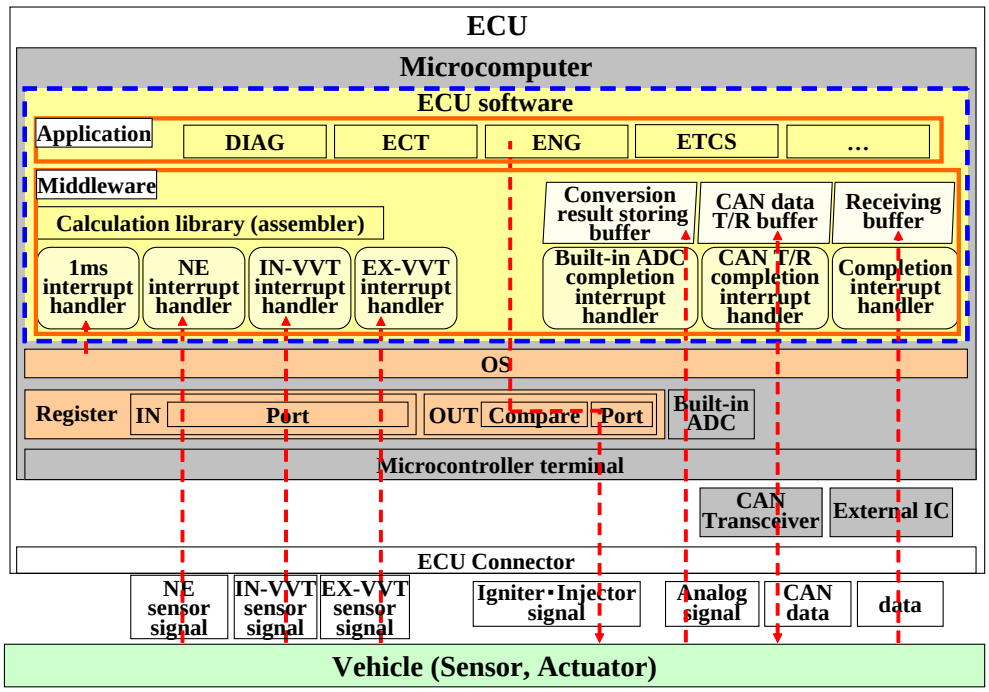


図 B-11-1　General Engine-ECU configuration diagram

制御アプリ用 SILS は、主に図 B-11-1 のアプリケーション部分に該当する C 言語ソースコードのデバッグに使われる。ECU のターゲットマイコンではなく汎用の PC で動作させるため、アクチュエータやセンサーの信号を処理するミドルウェアより下層のコードは SILS 専用の代替ソフトを使用し、コンパイラもターゲットマイコン用とは異なる。この構成の利点として、HILS よりもプラントモデルに対する制約が緩く、プラントモデルの設計や制御ソフトとの入出力に関する自由度が高いことや、汎用の統合開発環境が使えることが挙げられる。背反として、On Board Diagnosis (OBD) の一部やミドルウェアなどのコードを除外しているためそれらの開発には使えず、また代替ソフトが大規模になってしまう傾向にある。

マイコンエミュレーション型 SILS では、ターゲットマイコン用の命令セットをソフトウェア的に処理することで、ターゲットマイコン用オブジェクトコードを汎用 PC 上で一切の変更無く動作させる。制御アプリ用 SILS と異なり代替ソフトなどは不要であるが、プラントモデルとの入出力はマイクロプロセッサの入出力で規定され、プロセッサ以外は外部回路

を含め全てプラントモデルとして作成する必要がある。従って、制御アプリ用 SILS と比べてプラントモデルとの入出力に関する設計の自由度が極端に低いため、制御機能の開発をフロントローディングする狙いに対してマイコンエミュレーション型 SILS を使うことは難しい。

自動車制御開発をより効率的に行うために、SILS としては制御ソフト全体を対象とし、プラントモデルとの入出力に関して柔軟な設計が可能でなければならない。また、実用的なシミュレーション速度が求められることは言うまでもない。自動車は複数のコントローラによって制御されているため、コントローラ間の通信も模擬する必要がある。加えて、エンジン制御系では時間周期処理だけでなくクランク角度同期割り込みも存在するため、SILS の有効活用には各処理の優先順位や実行順序も考慮した正確な割り込み実行タイミングの実現も重要である。

SILS よりも更に上流で使う MILS に対しては、既存の制御系をベースに一部新規開発を行うプロセスに合うように、上述の機能を満たす SILS との深い連携が求められる。

そこで今回同社は、MILS 機能を内包し、かつ SILS としての主要な機能をほぼ網羅した次世代 SILS を開発したので報告する。

3. 次世代 SILS について

エンジン制御ソフト開発への適用を例に、次世代 SILS の詳細を述べる。次世代 SILS は図 B-11-1 に示す ECU 構成を前提とし、MathWorks 社の MATLAB/Simulink®上で動作する。図 B-11-2 にシステム構成の概略図を示す。

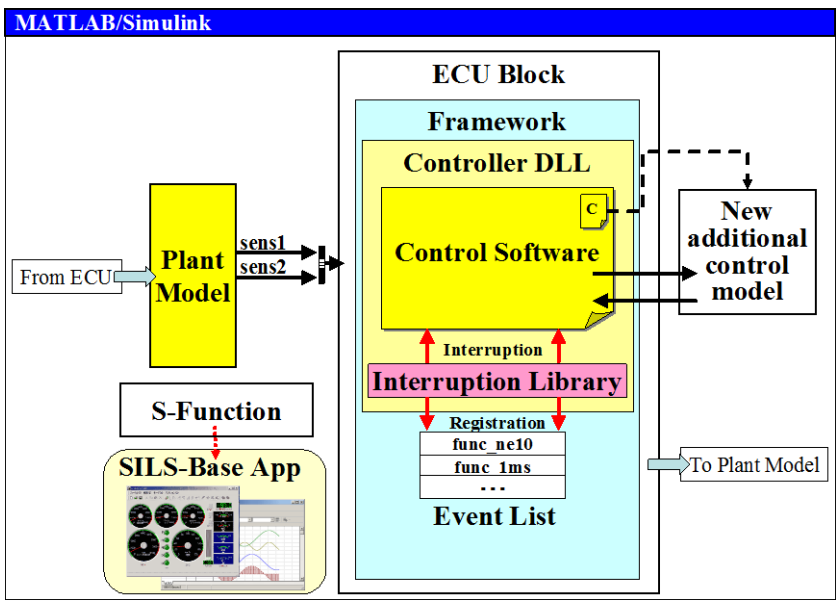


図 B-11-2　System configuration of the new SILS

次世代 SILS の中枢は図 B-11-2 中のフレームワークで、制御ソフトの実行を司っている。制御ソフトは、フレームワークと連動させるためのライブラリと結合し、使用する(図 B-11-2 中の DLL)。図 B-11-2 では簡単のために一つの制御ソフトだけを例示しているが、フレームワークは複数の制御ソフトを実行することができるようになっており、制御ソフト間のデータ受け渡し機能によって CAN 通信の簡易模擬も行う。

各制御ソフトの処理は図 B-11-1 に示す OS が実行しているタスク処理の模擬であるが、優先度なし/処理時間 0 秒で実行する簡易模擬である。処理のタイミングについては、割り込みを含め正確に模擬している（詳細後述)。また、ミドルウェアも含めて動作させるために、ルネサス エレクトロニクス社製 V850®マイコンのレジスタ処理を SILS 用に C 言語ライブラリとして実装した。制御ソフトの SILS 用コンパイルには Microsoft 社の Visual Studio®を使用する。

正確なタイミングで制御ソフトの処理を実行するための一般的な方法としては、マイコンエミュレーションのように割り込みを発生させるためのセンサー信号を模擬し、シミュレーションをマイコンクロック相当（ナノ秒オーダー）で実行する方法が考えられる。しかし、このような手法だけではシミュレーション全体の実行速度が低下してしまう。

割り込み実行タイミングの正確さを保ちつつもシミュレーション速度の低下を抑えるために、次世代 SILS ではプラントモデル用のソルバーと制御ソフト用のクロック処理を分離し、最適な時間間隔を独立に指定できるようにした（実行間隔分割機能)。加えて、全ての制御ソフトの割り込み発生タイミングを SILS フレームワーク内部で時系列リスト化し一元管理することで、センサー信号の模擬を大幅に効率化した（リスト管理機能)。

図 B-11-3 を例に、割り込み処理の流れを具体的に説明する。前提条件として、プラントモデルの計算間隔を入力 AD 変換処理周期とし（ここでは 4 ms とする)、制御ソフトの処理間隔を割り込み実行最小時間のサンプリング時間として（ここでは 125 ns とする)、シミュレーションした例を示す。

まず、現在時刻（0 ms とする）におけるエンジン回転数とクランク角度から、次回 1 ms 時間周期処理時刻(1.024 ms)における 10 degCA クランク角度同期割り込み発生時刻(1.280 ms）とプラントモデル積分時刻（4 ms）を算出し、フレームワークのリストへ登録する。シミュレーション現在時刻が、リストの先頭に登録された時刻（例の場合は 1.024 ms）に到達すれば、フレームワークによって 1 ms 時間周期処理が実行される。実行後は次回 1 ms 時間周期処理の発生時刻（2.048 ms）を再度算出し、フレームワークの時系列リストに登録する。同様にして 10 degCA クランク角度同期割り込みも実行される。プラントモデルの計算は、積分時刻に到達した段階で MATLAB に処理を移し、Simulink のソルバーによって積分が実行される。また、制御ソフトの割り込みが発生しない期間については次の割り込み発生時刻まで制御ソフトの現在時刻を進めることで効率的に割り込みを実行する。

以上の機能により、正確な割り込み処理実行タイミングと高速なシミュレーション速度の両立を実現している。

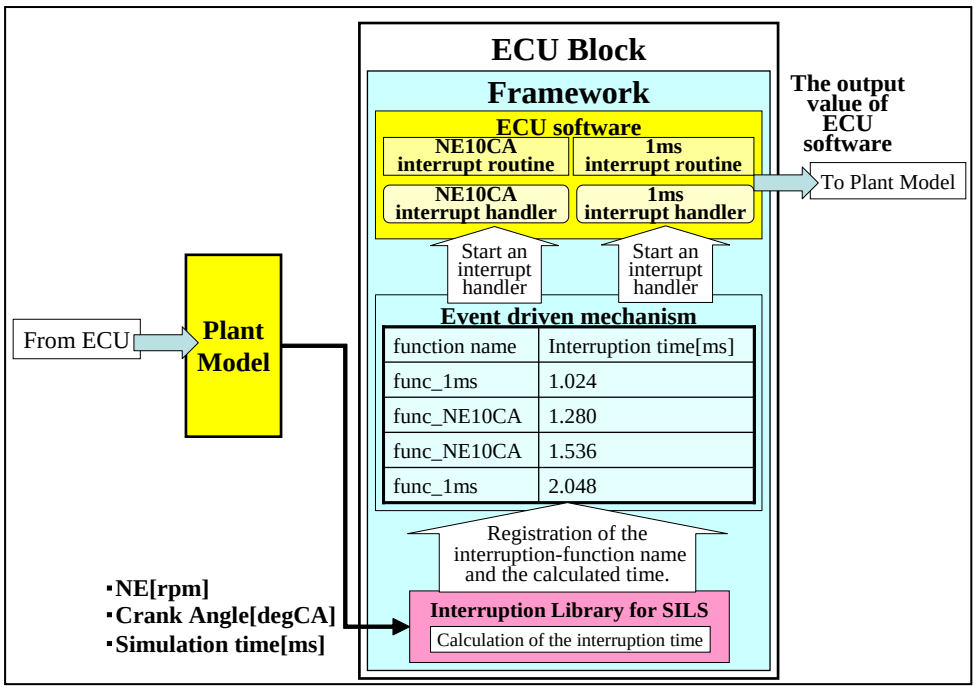


図 B-11-3　Event list function

ここまでは制御ソフト（C 言語ソースコード）を対象にした SILS の構成や機能について述べた。しかし、前章で述べた通り、制御系の開発プロセスにおけるフロントローディングを進めるためには制御ソフトの開発用に留まらず、更に上流の制御モデルの開発にも閉ループシミュレーションを活用したい。既存の制御系をベースに一部ロジックの新規開発を行うプロセスにおいて、このことは SILS を主体とする環境に新規ロジックをモデルとして組み込み、開発することを意味する。

図 B-11-2 内右上に示すように、一部ロジックがモデルになっている SILS を構築するために必要な機能は、主体である制御ソフト（C 言語ソースコード）側から望みのタイミングで制御モデルへ実行を移し、制御モデルの処理が終わり次第制御ソフトへ実行を戻す機能と、制御モデル側から制御ソフト内の変数を読み書きする機能である（+M 機能）。

次世代 SILS では、+M 機能の具体的な実現手段として、制御ソフト内の適切な箇所に制御モデルをコールする関数を書き加えることで、所望のタイミングで制御モデルを実行可能になっている。そして、制御モデル内に配置した専用ブロック経由で制御ソフト内部の変数に読み書きのアクセスをする。制御モデルの実行は、先に述べた実行間隔分割機能によってコントローラ側の一部として制御ソフトと一元管理され、処理されていく。この機能を用いることで、新規制御ロジックのモデルを開発している段階から、閉ループシミュレーションを利用することが出来る。

続いて、開発した各機能の評価結果について報告する。

4. 評価

まず、制御ソフト全体（制御アプリ＋ミドルウェア）を SILS として構築することができるか確認した。評価方法として、文献 0 の制御アプリ用 SILS に相当する SILS と次世代 SILS それぞれで、同じエンジン制御ソフトを SILS 化した時に実際にコンパイルしたファイル数をそれぞれカウントし、制御ソフトの評価対象範囲を定量化した（図 B-11-4 左）。次世代 SILS は ECU に実装される制御ソフトのほとんどを SILS 化対象として採用出来ていることが分かる。

次に、シミュレーション速度について評価した。先に述べた 2 つの SILS それぞれで同一のプラントモデルを用いて 10 秒のシミュレーションに要する実時間を計測した（図 B-11-4 右）。その結果、既存の SILS と比べて次世代 SILS は約 15%の速度低下に留まっていることが分かった。

これらの結果から、次世代 SILS は処理速度の面で実用性を損なうことなく、SILS 化対象となる制御ソフトの範囲を拡大することに成功していることが確認できた。

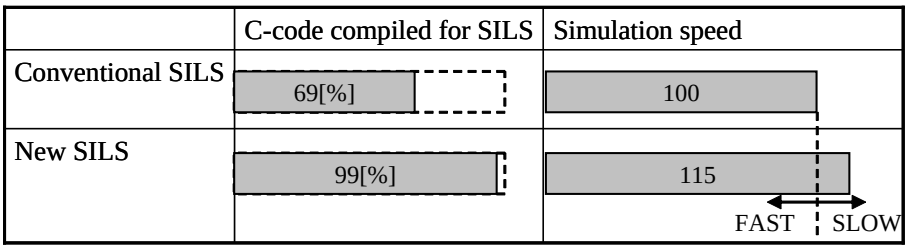


図 B-11-4　Comparison between conventional SILS and new SILS

次に、クランク角度同期割り込み機能と+M 機能について、合わせて評価を実施した。評価方法として図 B-11-5 に示すテスト環境を使用し、5 degCA 毎に実行されるクランク角度同期割り込みから制御モデルをコールする。制御モデル側では制御ソフト内部のカウンタ変数を読み込み、カウントアップ後、制御ソフト内部へ書き戻す。

また、そのカウンタ変数が時間軸ではなく、クランク角度軸でカウントされていることをエンジン回転数を変化させることで確認する。

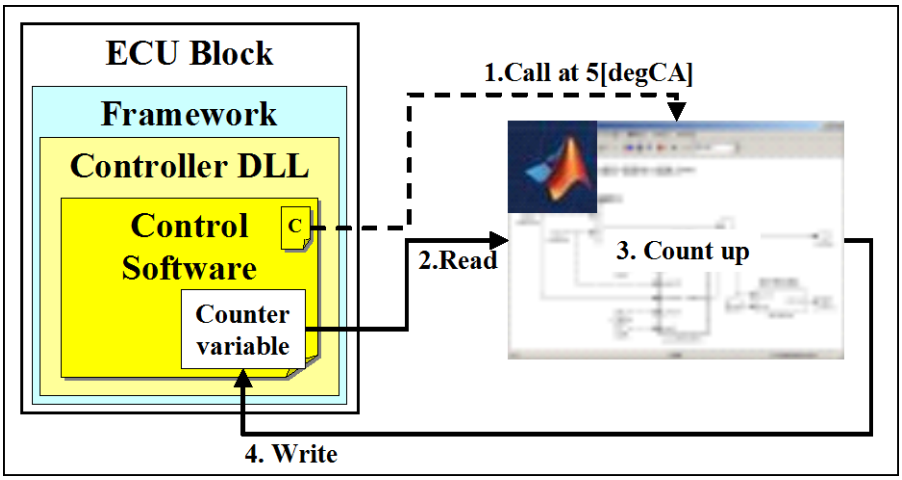


図 B-11-5　Test environment

図 B-11-6 にカウンタ変数とエンジン回転数のシミュレーション結果を示す。図 B-11-6 上段のモデルは図 B-11-5 中の制御モデルに対応する。エンジン回転数を 1000 rpm から 3000 rpm まで線形に変化させた際に、カウンタ変数のカウントアップの時間間隔が短くなっていることが確認出来る。これにより、制御モデルが時間軸ではなくクランク角度軸で正常に実行されており、エンジン回転数の変化によりクランク角度同期割り込みの実行タイミングが変化していると判断出来る。

以上より、クランク角度同期割り込みと+M 機能が正常に動作していることを確認出来た。

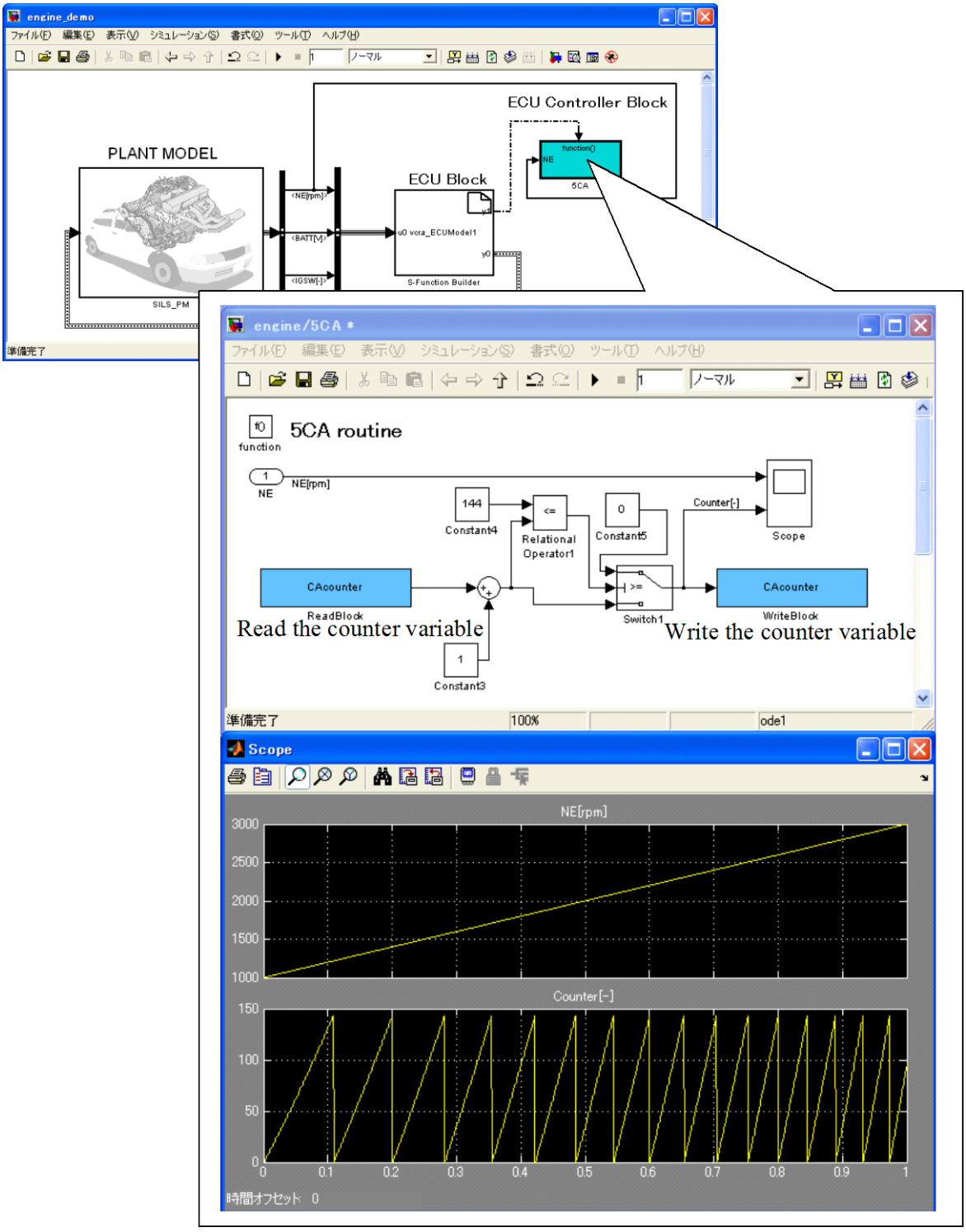


図 B-11-6　Simulation result

5. まとめ

制御系開発のフロントローディングに有用な次世代 SILS を開発した。本 SILS では、プラントモデルと制御ソフトの実行間隔分割機能と、制御ソフト実行のためのリスト管理機能を組み合わせることで、クランク角度同期割り込みタイミングを正確に模擬しつつも、処理速度の低下を抑制することに成功した。また、+M 機能により、制御ソフト開発だけでなく制御モデル開発にも有用な閉ループシミュレーション環境とすることができた。

本論文では適用対象としてエンジン制御系に特化したが、自動車のパワートレーン制御としては駆動やハイブリッドなど他の制御系と一体で動作しており、次世代 SILS もパワートレーン制御全体に適用していきたい。ただし、タスク優先度を考慮していないことや CAN 通信の模擬レベルが簡易なものであることが問題とならないかの検証は今後の課題である。

また、SILS は ECU のターゲットマイコンと異なるプロセッサ上で利用するため、実際の演算負荷や通信負荷などを直接的に確認することはできない。目的に応じてマイコンエミュレーションや HILS、実車など使い分けることが必要であり、開発プロセス全体の中での SILS の役割を明確化しておくことが重要である。SILS を利用するのに必要な設備は 1 台の PC だけであり、展開することは容易であるので、例えば制御ソフト開発に携わる OEM とサプライヤで共通環境として SILS を利用すれば、より効率的に制御ソフト開発を進めることができると期待される。また、プラントモデルの対応も必要となるが、制御機能検証やエンジン制御ソフトの適合に活用することも可能なはずであり、これらの用途拡大に向け、今後も開発を継続していく。

参考文献

[1]H. Brückmann, J. Strenkert, Dr. U. Keller, B. Wiesner, Dr. A. Junghanns. “Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL”, Getriebe in Fahrzeugen 2009, Friedrichshafen, 2009-06-30/07-01, http://www.qtronic.de/doc/DCT_2009.pdf (accessed on July 12, 2012)

[2]GAIA System Solutions INC, “CoMET System Engineering Environment”, Electronic Design and Solution Fair 2001, 横浜, 2001-02-01/02, http://www.cqpub.co.jp/dwm/eventreport/edsf2001/gaia/comet.pdf (accessed on July 12, 2012)

５．　おわりに

多くの検証技術の適用事例を紹介いただいた。色々な工夫によって、一定の効果を得られたことが示されている。世の中に認知された手法をそのまま適用するだけでは、開発現場での効果にどれほど寄与できるかは疑問であり、今回紹介いただいたような適用時の実践的で細かな一工夫を行うことによって、初めて役立つものになるということを知り得た。開発現場での実践的な工夫は、まだまだ存在すると予想し、それらを引き続き収集していきたいと考えている。

本報告書の内容は、提供いただいた事例に基づいて編集しています。