厳密な仕様記述における形式手法成功事例
調査報告書

2013 年 1 月

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

はじめに

独立行政法人情報処理推進機構　技術本部ソフトウェア・エンジニアリング・センター(以降、IPA/SEC と略す)では、仕様書作成に形式手法を用いた成功事例についてヒアリング調査を実施し、その結果から仕様書作成に係る諸問題と原因を分析し、それに対する形式手法の活用による解決策として、以下の 2 点をとりまとめました。

①「形式手法を上流工程における記述法として導入する際の技術的解決策」

②「日本語による仕様の記述において誤解の挿入を少なくする技術的解決策」

本報告書は、「2011 年度 システムエンジニアリング実践拠点事業」として、株式会社 SRA に委託し実施しました「形式手法を用いた日本語による仕様書作成に関する調査」をベースにしています。

厳密な仕様記述における形式手法成功事例

【調査報告書】

独立行政法人情報処理推進機構

エグゼクティブサマリー

本書は、上流工程における仕様書作成に形式手法を用いて成功した国内外の事例に対しヒアリングを行い、仕様書作成に係る諸問題の根本原因が何か、それを形式手法の活用によりどう解決したかを調査した結果を報告するものである。ヒアリング結果の分析を通して、(1)形式手法を上流工程における記述法として導入する際の技術的解決策の抽出、および(2)日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出を実施した。

調査においては、国内 3 プロジェクト（うち 2 事例は同一組織）、国外 8 事例を対象とした対面のヒアリングを実施した。プロジェクト実施者の観点からヒアリング項目および回答肢を準備した上で、構造化インタビューおよびメールによるフォローアップ調査を必要に応じて実施するなど、効果的な調査とするための工夫を行った。

「(1)形式手法を上流工程における記述法として導入する際の技術的解決策の抽出」を実施するにあたっては、形式手法の活用により仕様書に関連したどのような問題が、どのように解決あるいは軽減されたかを、ヒアリング対象毎に調査し、形式手法の導入に際して各ステークホルダにどのような手段を用いて手法への理解と活用を促したのかを分析した。その結果、形式手法を上流工程における記述法として導入する際には、(i) 問題領域の用語をそのまま記述したり、自然言語記述と形式言語記述の併記を支援したりできること(ii) ある観点におけるプロジェクト全体に渡る仕様を記述することができること(iii) ある程度の検証（準備、実施、確認）を自動的あるいは半自動的に行うことができること(iv) 仕様そのものや仕様に付随する情報を様々な形（図、表、自然言語）で抽出・加工したり、逆に取り込んだりするための仕掛けをもつこと(v) 複数の形式手法間の役割分担を支援できること(vi) 過去の開発資産を簡単に参照できること、という 6 個の要件を技術的に解決することが必要であることを同定した。

「(2)日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出」にあたっては、上流工程において日本語による仕様記述の曖昧さを回避し誤解の挿入リスクを軽減することに成功した実プロジェクト関係者を対象としたヒアリングを通して、日本語という自然言語を仕様書作成時に誤解挿入リスクを最小限にしながら取り入れ、多様なステークホルダの合意形成と効果的な関与を可能とするための、背景に形式手法を有するような構造化された日本語表現という技術的方策の効果および課題の分析を行った。その結果、日本語による仕様の記述において誤解の挿入を少なくする技術的解決としては、記述、追跡、比較、導出という 4 つの項目において、いくつかの基本動作を実現する必要があるとの結論に至った。そして、それらの基本動作の支援に必要となる、(i)語彙管理機能、(ii)テキスト分析機能、(iii)記述からの参照機能、(iv)記述間の突合せ機能、(v)構成管理機能という 5 つの機能を同定した。

目次

1. 本調査について

本書は、上流工程における仕様書作成に形式手法を用いて成功した国内外の事例に対しヒアリングを行い、仕様書作成に係る諸問題の根本原因が何か、それを形式手法の活用によりどう解決したかを調査した結果を報告するものである。ヒアリング結果の分析を通して、「(1)形式手法を上流工程における記述法として導入する際の技術的解決策の抽出」、および「(2)日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出」を実施した。なお、本調査報告書は「形式手法入門教材」[1]の副読本を作成するために活用するものである。

1.1 背景と目的

ソフトウェア開発において上流工程における文書化の不十分さに起因する品質の低下や開発コストの上昇を低減するための手法として形式手法が挙げられる。実際のソフトウェア開発プロジェクトに形式手法を適用するためには、ステークホルダの理解、導入のためのガイドライン、教育などが必要である。

本調査では、１）形式手法が仕様書作成に関連する諸問題をどのように改善したか、２）形式手法を導入することでどのような効果があったのか、３）ステークホルダ―や協力組織とのコミュニケーションをどのようにしたのか、４）どのような教育をしたのかといった観点から、上流工程における仕様書作成に形式手法を用いて成功した事例を対象としたヒアリング調査を実施し、

- 形式手法を上流工程における記述法として導入する際の技術的解決策
- 日本語による仕様の記述において誤解の挿入を少なくする技術的解決策

を抽出することを目指した。

1.2 本書の構成

以下に続く 2 章では、技術的解決策の抽出に向けて、実施したヒアリング調査の概要を説明する。まずヒアリング調査の手順を説明し、続いてヒアリング対象とした選定事例を示す。本調査では、国内 3 事例、海外 8 事例、計 11 事例を、ヒアリング対象として選定した。ヒアリングした具体的な内容を列挙し、最後にヒアリング対象とした事例プロジェクトの特徴を述べる。

3 章では、実施したヒアリング結果の概要を説明する。各ヒアリング項目に対して、各々の事例プロジェクトの担当者により回答された結果を列挙する。

4 章では、ヒアリング対象とした 11 件の事例プロジェクトについて、(1)プロジェクトの基本情報と特性、(2)プロジェクトの概要、(3)開発プロセスの概要、(4)記述と支援ツール、(5)厳密な仕様記述の効果、(6)厳密な仕様記述を適用するための工夫と効果、(7)顧客とのコミュニケーション、(8)開発者内のコミュニケーション、(9)教育、という 9 個の観点か

[1] 実務家のための形式手法 教材「厳密な仕様記述を志すための形式手法入門」の公開 http://sec.ipa.go.jp/reports/20121113.html

ら、個別に分析した結果を示す。

5 章では、事例を横断的に分析し、考察した結果を論じる。5.1 では、ヒアリングの結果から明らかとなった形式手法の適用箇所について、成功している事例に共通する点、あるいは類似した点という観点から説明を行う。5.2 では、同じくヒアリング結果から明らかとなった形式記述と自然言語（日本語）をはじめとするそれ以外の記述との併存について、共通する点あるいは類似した点という観点から説明を行う。形式手法の導入に関しては、1990 年にアンソニーホール(Anthony Hall)[2]が「形式手法の 7 つの神話」として、典型的な形式手法に関わる誤解を 7 つ列挙している。5.3 では、ヒアリング結果を踏まえた上で、これらの誤解に対する実証的な反証を列挙する。

6 章では、ヒアリング調査結果とその分析および考察から導出した、技術的解決策を説明する。6.1 では、形式手法を上流工程における記述法として導入する際の技術的解決策を説明する。6.2 では、日本語による仕様の記述において誤解の挿入を少なくするための技術的解決策を説明する。

7 章では、本調査全体を振り返り、形式手法がこれから適用され仕様記述の質へとつながるための、より広い視野での方向性について考察し、本報告書を結ぶ。

巻末に、Appendix として本調査のヒアリングに用いた質問と回答肢のリスト、ヒアリング分析時の着目点、および参考資料のリストを示す。

[2] Anthony Hall　http://www.anthonyhall.org/

2. ヒアリング調査の流れ

本章では、実施したヒアリング調査の流れについて述べる。

2.1 ヒアリング調査手法

本ヒアリング調査を実施するにあたり、あらかじめ選定した調査対象事例に関わりの深い研究者 2 名とコンタクトを取った。それに基づきヒアリング対象者を選定した。選定したヒアリング対象者に対して、実際のヒアリングで使用する質問項目を踏まえた上で作成した質問票を電子メールであらかじめ送付し、事前調査を行った。

まずヒアリングに先立って、「(1)形式手法を上流工程における記述法として導入する際の技術的解決策の抽出」を実施するための計 13 項目を網羅するような計 47 個の質問と回答肢（Appendix 1.1 参照）、および「(2)日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出」を実施するための計 8 項目を網羅するような計 15 個の質問と回答肢（Appendix 1.2 参照）を、プロジェクト実施者の観点を取り入れ、準備した。各ヒアリング対象プロジェクトには事前にヒアリング項目をまとめた事前質問をメールで送付し、また関連文書等の提供をあらかじめ依頼した。質問票に対しては多忙のため返信がなかった一部を除いて回答を得た。一部、質問票への回答に対して使用語彙の出現頻度を測定するなどの認知科学的アプローチも取り入れ、対面ヒアリングの準備をした。

ヒアリング対象者は、この質問票に基づいて、プレゼンテーション資料や関連論文といった資料を実際のヒアリングに際して準備していることが多く、事前調査として質問票を送付することで、より効果的なヒアリングを実施することができたと考えられる。

また、次節に説明するように、事前に送付した質問票に基づいてヒアリング対象者から別の事例の紹介を受けた。結果として、本調査の主旨に沿う事例として計 6 事例が追加できた。

実際の対面ヒアリングでは、質問項目とある程度の回答をあらかじめ想定するという構造化インタビューの形式をとった。ヒアリング対象者の希望に応じて、その多くがプレゼンテーション形式で行われ、プレゼンテーションの中でヒアリング項目に関する言及を抽出した。また、直接言及のなかった項目については適宜口頭質問によって聞き取りを行った。

対面ヒアリングは、分析用途のみで外部に提供しないという条件の下で、許可された場合にのみビデオ撮影を行った。この撮影したビデオ記録は、ヒアリング結果の分析と考察に活用した。

2.2 ヒアリング対象選定事例

表 2-1 に、本調査で実施したヒアリングの対象プロジェクト、およびヒアリング実施日を示す。国内 3 事例、国外 8 事例、計 11 事例である。

表 2-1: ヒアリング事例一覧

ドメイン	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 実施日	ヒアリング 場所	ヒアリング セッション 番号
組込	FeliCa ファームウェア(1)	フェリカネットワークス 株式会社	2012/03/30	東京都	**HS01**
	FeliCa ファームウェア(2)	フェリカネットワークス 株式会社	2012/04/19	東京都	**HS02**
	BPS1000 紙幣検査機	Aarhus University	2012/04/10	オランダ	**HS03**
航空	SHOLIS	Altran Praxis Limited	2012/04/03	英国	**HS04**
	iFACTS	Altran Praxis Limited	2012/04/03	英国	**HS05**
証券	TradeOne	SCSK 株式会社 （旧 日本フィッツ株式会社）	2012/04/27	東京都	**HS06**
テキスト処理	オランダ軍 メッセージ分析	C.H.E.S.S. Group	2012/04/10	オランダ	**HS07**
	CAVA	Aarhus University	2012/04/10	オランダ	**HS08**
セキュリティ	MULTOS CA	Altran Praxis Limited	2012/04/03	英国	**HS09**
	Tokeneer	Altran Praxis Limited	2012/04/03	英国	**HS10**
E-コマース	オランダ花市場	C.H.E.S.S. Group	2012/04/10	オランダ	**HS11**

本ヒアリング調査では対象としてあらかじめ選定した調査対象は、[HS01] FeliCa ファームウェア(1)：フェリカネットワークスの IC カードシステム(phase1)、[HS02] FeliC ファームウェア(2)：フェリカネットワークスの IC カードシステム(phase2)、[HS06] TradeOne：SCSK の証券パッケージ、[HS11]オランダ花市場のオークションシステム、[HS05] iFACTS：ヒースロー空港の航空管制システムであった。

本調査では、これら 5 件の必須対象に加えて、6 件の対象事例を加え調査を実施した。[HS03] BPS1000 紙幣検査機、[HS07] オランダ軍メッセージ分析、[HS08] CAVA の 3 事例は、[HS011] オランダ花市場オークション事例に関するヒアリング事前調査のやり取りの中で本調査の主旨に沿ったものとして提示され、分析対象に加えた。[HS04] SHOLIS、[HS09] MULTOS CA、[HS10] Tokeneer の 3 事例は、[HS05] iFACTS：ヒースロー空港の航空管制システムのヒアリング事前調査のやり取りの中で、本調査の主旨に沿ったものとして提示され、分析対象に加えた。

2.3 ヒアリング対象者の選定

ヒアリング対象者の選定にあたっては、各調査対象事例について内容を良く知り、当該プロジェクトを代表して説明できる立場の人物であることを条件とした。

具体的には、[HS01][HS02][HS06]は国内事例であり、それぞれの開発企業の開発（技術）責任者に直接コンタクトを取り、ヒアリングを依頼した。

[HS03][HS07][HS08][HS11]は欧州での形式手法実践研究の第一人者であるデンマーク Aarhus University の Peter Larsen 教授と連絡を取り、当該プロジェクトでの技術的責任者または技術コンサルタントをヒアリング対象として選定し、ヒアリングを依頼した。[HS05]は英国における形式手法研究の第一人者である South Bank University の Jonathan Bowen 教授より開発した企業の主任技術者の紹介を受け、ヒアリングを依頼した。[HS04][HS09][HS10]は[HS05]のヒアリング対象から本調査の主旨に沿ったものとして提示された事例であり、同一人物にヒアリングを依頼した。

表 2-2 に、対面したヒアリング対象者の、プロジェクトにおける役割を示す。

表 2-2: ヒアリング対象者の役割

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング対象 者の役割
HS01	FeliCa ファームウェア(1)	フェリカネットワークス 株式会社	開発責任者
HS02	FeliCa ファームウェア(2)	フェリカネットワークス 株式会社	開発責任者
HS03	BPS1000 紙幣検査機	Aarhus University	技術コンサルタント
HS04	SHOLIS	Altran Praxis Limited	Principal Engineer
HS05	iFACTS	Altran Praxis Limited	Principal Engineer
HS06	TradeOne	SCSK 株式会社 （旧　日本フィッツ株式会社）	開発責任者
HS07	オランダ軍 メッセージ分析	C.H.E.S.S. Group	開発責任者
HS08	CAVA	Aarhus University	技術コンサルタント
HS09	MULTOS CA	Altran Praxis Limited	Principal Engineer
HS10	Tokeneer	Altran Praxis Limited	Principal Engineer
HS11	オランダ花市場	C.H.E.S.S. Group	開発責任者

2.4 ヒアリング内容

「作業項目１：形式手法を上流工程における記述法として導入する際の技術的解決策の抽出」を目的として実施した、ヒアリング項目を以下に列挙する。

① 上流工程でのバグ挿入に係る諸問題の発見を可能にした工夫、方法

② 曖昧さや不明確さのようなバグ挿入を許すような表現の除去に関する工夫、方法

③ 上記②と、従来のレビュー・インスペクションとの相違点、効率性

④ 形式手法導入の効果と、従来手法との相違点

⑤ 形式手法適用の可能性、または、適用支援ツールの活用方法

⑥ 上記①から⑤で導入された方法論、ツール導入に要したワークロードおよびコスト

⑦ 上記①から⑤で形式手法を有効に活用するためのスキル育成方法、組織展開方法

⑧ バグ挿入自体を少なくすること、および、挿入時点から近い時点でのバグの除去をシステム・ソフトウェア開発ライフサイクルプロセスの上流工程で可能にした管理方法論

⑨ 上流工程以降の工程における、形式手法のもたらした効果

⑩ 上流工程で記述した仕様の、以降の工程での修正とその理由、従来手法の導入との違い

⑪ 修正量の減少が、プロジェクトの QCD 向上効果に与える影響のメトリックス評価

⑫ 形式手法導入を決定した管理者あるいはプロジェクトリーダーの思い

⑬ 自然言語処理の観点での誤解の最小化

これらのヒアリング項目をまとめるにあたって着目した分析項目を、Appendix 2.1 に示す。

「作業項目２：日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出」を目的として実施した、ヒアリング項目を以下に列挙する。

① 日本語をプロジェクトの上流工程でステークホルダ間の共通語として使用するにあたり、誤解を生じさせないコミュニケーションの訓練方法

② 上流工程でステークホルダ間での誤解を最小化して仕様を記述したにもかかわらず、後工程において誤解に基づくと推定される問題点が発見された場合、それが看過された理由、また、その対応策

③ 上記①と②の技術的解決策として、日本語での仕様記述の影響を最小化する手法。形式手法をベースにしている場合、形式手法をあまり意識しないで仕様記述できるような方法論

④　上記③の方法論において、通常の日本語を使用して仕様を記述する場合との比較、有効性

⑤　上記③の方法論において、その導入における問題点とその対応方法

⑥　背景に形式手法を持つ構造化された日本語での仕様書作成のための方法論としての、今後の見通し

⑦　特定のドメインに対する DSL 的アプローチ、及び一般的な方法論としてのアプローチ方法

⑧　技術的解決策としての今後の見通し

これらのヒアリング項目をまとめるにあたって着目した分析項目を、Appendix 2.2 に示す。

2.5 ヒアリング対象事例の特徴

ヒアリング対象とした計 11 件の事例の特徴を概観する。

表 2-3 に、ヒアリング対象とした事例プロジェクトを、形式手法を導入した主な動機という観点から整理したものを示す。表 2-4 に、発注者と受注者とのコミュニケーションに形式仕様記述を利用したか否かの結果を示す。表 2-5 に、形式手法の教育の実施状況を示す。

表 2-3: 形式手法を導入した主な動機

ヒアリング対象プロジェクト		仕様記述の品質向上	高信頼性	標準・規定への準拠	工期短縮	開発者間の共通認識	開発者の経験
HS01	FeliCa ファームウェア(1)	○	○			○	
HS02	FeliCa ファームウェア(2)	○	○			○	○
HS03	BPS1000 紙幣検査機				○		
HS04	SHOLIS		○	○			
HS05	iFACTS		○	○			
HS06	TradeOne	○	○		○	○	
HS07	オランダ軍 メッセージ分析				○		○
HS08	CAVA					○	
HS09	MULTOS CA		○	○			
HS10	Tokeneer	○	○	○			
HS11	オランダ花市場	○	○				

表 2-4: 発注者と受注者とのコミュニケーションに形式仕様記述を利用したか否か

ヒアリング対象プロジェクト		形式仕様記述を利用した	形式仕様記述を利用しなかった	該当なし
HS01	FeliCa ファームウェア(1)			N/A
HS02	FeliCa ファームウェア(2)			N/A
HS03	BPS1000 紙幣検査機			N/A
HS04	SHOLIS		○	
HS05	iFACTS	○		
HS06	TradeOne			N/A
HS07	オランダ軍 メッセージ分析		○	
HS08	CAVA			N/A
HS09	MULTOS CA		○	
HS10	Tokeneer	○		
HS11	オランダ花市場			N/A

表 2-5: 形式手法の教育の実施状況

ヒアリング対象プロジェクト		読み	読み書き	必要がなかった
HS01	FeliCa ファームウェア(1)		○	
HS02	FeliCa ファームウェア(2)			N/A
HS03	BPS1000 紙幣検査機		○	
HS04	SHOLIS		○	
HS05	iFACTS		○	
HS06	TradeOne		○	
HS07	オランダ軍 メッセージ分析			N/A
HS08	CAVA		○	
HS09	MULTOS CA		○	
HS10	Tokeneer	○		
HS11	オランダ花市場	○		

3. ヒアリング結果概要

実施したヒアリングに対する回答を、「作業項目１：形式手法を上流工程における記述法として導入する際の技術的解決策の抽出」のための計 13 項目(うち 2 項目は回答が他項目と不可分のため併合した)、および「作業項目２：日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出」のための計 8 項目(うち 2 項目は回答が他項目と不可分のため併合した)毎に整理する。

3.1「作業項目１:形式手法を上流工程における記述法として導入する際の技術的解決策の抽出」に関するヒアリング結果

①　上流工程でのバグ挿入等の諸問題の発見・明確化がどのようにして可能になったか。

表 3-1: ヒアリング項目①に対する各事例からの回答

ヒアリング対象プロジェクト		上流工程の問題が発見され明確化された主な工程						
		機能仕様	仕様アニメーション[3]	実装テストでの突合	機能仕様証明	機能設計証明	非形式的要求記述	評価仕様
HS01	FeliCa ファームウェア(1)	○	○	○				
HS02	FeliCa ファームウェア(2)	○	○	○				
HS03	BPS1000 紙幣検査機	○		○				
HS04	SHOLIS	○		○	○	○		
HS05	iFACTS	○		○	○	○		
HS06	TradeOne	○	○	○				
HS07	オランダ軍 メッセージ分析	○	○	○			○	○
HS08	CAVA	○	○	○				
HS09	MULTOS CA	○		○	○	○		
HS10	Tokeneer	○		○	○	○		
HS11	オランダ花市場		○	○				

[3]仕様の実行を通して検証を行う手法のこと

ヒアリングを行った全事例において、機能仕様を記述すること、ならびに仕様を実装テストの結果と突き合わせることによって、上流工程の問題が発見されたとしていた。その他にも、仕様アニメーション、機能仕様に対する証明、機能設計に対する証明によっても発見されたとする事例があった。[HS07]は、非形式的な要求を記述する工程と、評価仕様を記述する工程で、上流工程の問題が発見されたとのことであった。

② 曖昧さや不明確さのようなバグ挿入を許すような表現はどのようにして除去できたか。

表 3-2: ヒアリング項目②に対する各事例からの回答

ヒアリング対象プロジェクト		曖昧さや不明確さのようなバグ挿入を許すような表現に対する主な除去法						
		厳密な形式仕様記述言語	日本語識別子	適切な抽象化	問題領域の用語	準形式表現	図式モデル言語	形式記述と非形式記述
HS01	FeliCa ファームウェア(1)	○						○
HS02	FeliCa ファームウェア(2)	○	○					
HS03	BPS1000 紙幣検査機	○		○				
HS04	SHOLIS	○						○
HS05	iFACTS	○						○
HS06	TradeOne	○	○		○			
HS07	オランダ軍メッセージ分析	○				○		○
HS08	CAVA	○						○
HS09	MULTOS CA	○						○
HS10	Tokeneer	○						○
HS11	オランダ花市場	○					○	○

全ての事例において、仕様を厳密に記述するために形式仕様記述言語を採用することで、曖昧さや不明確さのようなバグ挿入を許すような表現に対する効果があったとしていた。また、多くの事例で、形式仕様記述とそれに対応する非形式的記述を突き合わせる形でレビューすることにより、非形式的記述の意味の曖昧性を解決したとのことであった。

その他の効果としては、“形式仕様記述の識別子に日本語の単語を用いることで、日本語としての読みやすさを保ちつつ形式仕様による厳密な記述を実現した”（[HS02][HS06]）、“適切な抽象化を行うことで仕様記述の複雑化を防いだ”（[HS03]）、“形式仕様記述の識別子に問題領域の用語を用いることで、問題領域での事象との関連付けを容易にした”（[HS06]）、“形式仕様言語を適用しない場合に、可能な限り形式度が高く厳密な記法(準形式表現)によってモデルを表現することで、記述の複雑化を防ぎ比較的厳密な記述を行った”（[HS07]）、“要求定義を図的記法で表現することで、自然言語からくる解釈の曖昧性を回避した”（[HS11]）といったものが挙げられた。

③ 上記②は、従来のレビュー・インスペクションとどう違うのか、および、どう効率的になったのか。

表 3-3: ヒアリング項目③に対する各事例からの回答

ヒアリング対象プロジェクト		従来のレビュー・インスペクションによる曖昧さや不明確さの排除法との主な違い						
		証明による正当性の検証	アニメーション	証明による妥当性の検証	レビュー	派生・転用	外在化	見積もり
HS01	FeliCa ファームウェア(1)		○		○			
HS02	FeliCa ファームウェア(2)		○		○			
HS03	BPS1000 紙幣検査機							○
HS04	SHOLIS	○		○	○			
HS05	iFACTS	○		○	○			
HS06	TradeOne		○		○		○	
HS07	オランダ軍 メッセージ分析		○		○			
HS08	CAVA		○		○			
HS09	MULTOS CA	○		○		○		
HS10	Tokeneer	○		○		○		
HS11	オランダ花市場		○		○			

“レビューを行う際に、自然言語による仕様記述と比べ、レビュー対象とする範囲を明確に設定することができる”とした回答が多くあった。また、“形式仕様記述によって、設計や実装が仕様を満たすかどうかを証明によって検証することができた点が、レビューとの違いであった”と回答した事例も多くあった。そのほかにも、“形式仕様記述によって、要求

を満たすかどうかを証明によって検証することができた”、“形式仕様記述をアニメーションにより実行することで、要求を満たすかどうか検証することができた”、“形式的記述の一部を複数の工程での記述に転用することができ、この結果、複数の工程で同じ記述を共有することができ、曖昧性を回避できた”、“文脈や問題領域知識の援用が必要な自然言語での仕様記述と違い”、“形式仕様記述では全ての条件や定義が記述されるため、何も隠された条件がないという確信を得ることができた”、“形式仕様では全てが記述され、後工程の成果物への影響も追跡可能であることから、形式手法による過去の経験に基づく見積もりをより正確に行うことができた”といった回答があった。

④ 上記③が、形式手法導入による効果であり、従来手法では得られなかったとヒアリング対象者が回答したとすると、両者の違いはどのようにメトリックスで把握されているか。

表 3-4: ヒアリング項目④に対する各事例からの回答

ヒアリング対象プロジェクト		形式手法と従来手法の手法に関するメトリックス上の違い		
		メトリックス収集が容易	責任範囲の明確化	派生表現による可読性向上
HS01	FeliCa ファームウェア(1)	○	○	
HS02	FeliCa ファームウェア(2)	○		○
HS03	BPS1000 紙幣検査機	○		
HS04	SHOLIS	○		
HS05	iFACTS	○		
HS06	TradeOne	○		
HS07	オランダ軍 メッセージ分析	○		
HS08	CAVA	○		
HS09	MULTOS CA	○		
HS10	Tokeneer	○		
HS11	オランダ花市場	○		

全てのヒアリング事例において、形式手法は従来手法と比較してレビューやインスペクション対象の単位と範囲が明確であることから、メトリックスの収集が比較的容易であったとのことであった。この他にも、従来手法ではレビューやインスペクションの責任範囲

が曖昧になりがちで、問題を引き起こしていた（[HS01]）、形式手法から表形式等の表現を派生することによって、可読性が向上した（[HS02]）との指摘もあった。

⑤ 上記③が、形式手法導入による効果であると認識されている場合、形式手法のどのような適用がそれを可能にしたのか。また、適用を支援するツールはどのように活用されたか。

表 3-5-1: ヒアリング項目⑤ 効果的であった形式手法の適用に対する各事例からの回答

ヒアリング対象プロジェクト		形式手法の主な適用					
		派生表現	構文検査･型検査	証明	コード自動生成	記述単位の明確化	アニメーション
HS01	FeliCa ファームウェア(1)		○			○	○
HS02	FeliCa ファームウェア(2)	○	○				○
HS03	BPS1000 紙幣検査機	○	○				○
HS04	SHOLIS		○	○		○	
HS05	iFACTS		○	○		○	
HS06	TradeOne		○			○	○
HS07	オランダ軍 メッセージ分析		○		○	○	○
HS08	CAVA		○			○	○
HS09	MULTOS CA		○	○		○	
HS10	Tokeneer		○	○		○	
HS11	オランダ花市場		○			○	○

多くの事例において、記述する度に構文検査や型検査による検証を行った、および、形式仕様言語が定義する記述単位に仕様をまとめた、とのことであった。仕様アニメーションによって仕様を実際に実行し動作を検証したという回答も多数あった。その他には、形式仕様から表形式などの表現を派生させることで可読性を向上させた、記述した形式仕様の性質を数学的に証明した、形式仕様から実行プログラムを自動的に生成した、といった回答があった。

表 3-5-2: ヒアリング項目⑤ 適用を支援したツールに対する各事例からの回答

ヒアリング対象プロジェクト		形式手法適用のために導入した主なツール				
		構文検査・型検査	実行処理系	証明支援系	コード生成系	自製ツール
HS01	FeliCa ファームウェア(1)	○	○			
HS02	FeliCa ファームウェア(2)	○	○			○
HS03	BPS1000 紙幣検査機	○	○			
HS04	SHOLIS	○		○		
HS05	iFACTS	○		○		
HS06	TradeOne	○	○			○
HS07	オランダ軍 メッセージ分析	○	○		○	○
HS08	CAVA	○	○			
HS09	MULTOS CA	○		○		
HS10	Tokeneer	○		○		
HS11	オランダ花市場	○	○			

全ての事例において、構文や型の正しさを自動的に検証するツールを使用していた。その他には、仕様をアニメーション実行するためのツール、仕様の性質を証明する作業を半自動で支援するためのツール、仕様から実行プログラムを自動的に生成するツール、あるいは当該開発のために開発したツールやフレームワーク、プラグイン等が挙げられた。

⑥ 上記①から⑤で導入された方法論およびツール導入に要したワークロードおよびコストはどのようなものであったか。

表 3-6: ヒアリング項目⑥に対する各事例からの回答

ヒアリング対象プロジェクト		形式手法の適用とツールの導入に要した主要なワークロードおよびコスト			
		言語教育	専任の開発支援担当者	兼任の開発支援担当者	コンサルタント
HS01	FeliCa ファームウェア(1)	○	○		○
HS02	FeliCa ファームウェア(2)	○	○		○
HS03	BPS1000 紙幣検査機	○			○
HS04	SHOLIS	○			○
HS05	iFACTS	○			○
HS06	TradeOne	○		○	
HS07	オランダ軍 メッセージ分析			○	
HS08	CAVA	○			○
HS09	MULTOS CA	○			○
HS10	Tokeneer	○			○
HS11	オランダ花市場	○			○

ほとんどの事例が、形式仕様記述言語の教育としてクラスルーム形式の座学と演習を計 1 週間実施していた。さらに、プロセスと成果物の改善レビューのためのコンサルタントを導入したとのことであった。その他には、仕様記述のためにツールやフレームワーク、プラグイン等を専任、あるいは兼任の担当者が作成したとのことであった。

⑦ 形式手法を有効に活用するためのスキル育成にはどのような方法が取られ、組織展開されたか。

表 3-7: ヒアリング項目⑦に対する各事例からの回答

ヒアリング対象プロジェクト		スキル養成		
		コンサルタントによる継続支援	クラスルーム形式	特になし
HS01	FeliCa ファームウェア(1)	○	○	
HS02	FeliCa ファームウェア(2)	○	○	
HS03	BPS1000 紙幣検査機	○	○	
HS04	SHOLIS	○	○	
HS05	iFACTS	○	○	
HS06	TradeOne		○	
HS07	オランダ軍 メッセージ分析			○
HS08	CAVA	○	○	
HS09	MULTOS CA	○	○	
HS10	Tokeneer	○	○	
HS11	オランダ花市場	○	○	

ほとんどの事例において、仕様のレビューや開発プロセスの改善を行うコンサルタントによる形式仕様言語の OJT を継続し、また仕様記述言語についての座学と実習を計 1 週間受講したとのことであった。[HS07]は、開発組織が既に形式手法に熟練しており、スキル養成のための教育は必要なかったとのことであった。

⑧ バグの挿入自体を少なくすること、および、挿入時点から近い時点でのバグの除去をシステム・ソフトウェア開発ライフサイクルプロセスの早い段階、即ち上流工程で可能にした管理方法論はどのようなものであったか。上流工程以降の工程における、形式手法のもたらした効果はどのようなものであったか。

表 3-8: ヒアリング項目⑧に対する各事例からの回答

ヒアリング対象プロジェクト		上流工程での主要なバグの除去法					
		仕様アニメーション			証明		適切な抽象
		下流工程での主要な効果					
		検証された仕様からのテストオラクル生成	検証された仕様を用いた設計	検証された仕様からのコード自動生成	設計に表明として埋め込み動的に検証	検証された仕様を用いた設計	整理された仕様を用いた設計
HS01	FeliCa ファームウェア(1)	○	○				○
HS02	FeliCa ファームウェア(2)	○	○				○
HS03	BPS1000 紙幣検査機		○				○
HS04	SHOLIS				○	○	○
HS05	iFACTS				○	○	○
HS06	TradeOne	○	○				○
HS07	オランダ軍 メッセージ分析	○	○	○			○
HS08	CAVA	○	○				○
HS09	MULTOS CA				○	○	○
HS10	Tokeneer				○	○	○
HS11	オランダ花市場	○	○				○

全ての事例が、適切な抽象化がされ整理された仕様記述を基礎にした設計をすることで、品質の高い設計を実現したことを挙げた。また、いくつかの事例では、仕様アニメーションにより妥当性が検証された仕様をテストオラクル生成に利用することで、効果的なテストが実現された、あるいは、設計の基礎となる仕様記述を設計以前に検証することで、より信頼性の高い設計が実現されたとのことであった。この他には、仕様記述を直接アニメーション実行することで、仕様の妥当性を検証した、安全要件やセキュリティ要件等の

要求項目を形式的に記述し、仕様記述がその要求項目を満たすかどうかを証明により検証した、仕様記述の対象を適切に抽象化し適切な粒度で記述することで仕様記述そのものの可読性や保守性を高めた、仕様アニメーションにより妥当性が検証された仕様からツールによりプログラムソースコードを自動生成することで開発工数を大幅に短縮することができた、設計において形式仕様記述中の制約条件や証明された性質を表明として埋め込み、実装テストによってその制約条件や性質が実際に満たされている事を検証することで、高い信頼性を実現した、などの回答が得られた。

⑨ 上流工程以降の工程における、形式手法のもたらした効果はどのようなものであったか。

表 3-9: ヒアリング項目⑨に対する各事例からの回答

ヒアリング対象プロジェクト		形式手法の主な効果							
		テスト仕様策定および実施	他文書の派生	良質なマニュアル	高い信頼性	仕様追加への対応	追跡性の確保	工期短縮	トレーニングが容易
HS01	FeliCa ファームウェア(1)	○		○	○				○
HS02	FeliCa ファームウェア(2)	○	○	○					○
HS03	BPS1000 紙幣検査機	○		○	○	○		○	○
HS04	SHOLIS	○	○		○		○		
HS05	iFACTS	○	○		○		○		
HS06	TradeOne	○			○				○
HS07	オランダ軍メッセージ分析	○			○		○		
HS08	CAVA	○							
HS09	MULTOSCA	○	○		○		○		
HS10	Tokeneer	○	○		○		○		
HS11	オランダ花市場	○			○				

全ての事例が、仕様を設計以前に検証しているため、テスト仕様の策定に早期に着手す

ることができたとしていた。また、仕様からテストオラクルの生成やテストケースの抽出ができたことで、テストを効率よく実施できたとの回答であった。高い信頼性を持つシステムが出荷されたと回答した事例も多くあった。この他には、仕様から表形式の文書を派生させることで可読性が向上し、仕様に書かれている記述を設計書の中で表明（アサーション、プログラムが実行時に満たすべき条件式）として派生することで効果的なテストが実施できた、マニュアル記述の基礎となる仕様記述を曖昧なく定義することで高品質なマニュアルが策定された、適切に抽象化と整理がされた高い品質の仕様を記述することで仕様追加に対して柔軟な対応をすることができた、仕様を曖昧なく定義することで要求への追跡性や下流工程の成果物からの追跡性が確保された、自然言語による仕様記述と異なり、開発や保守に必要な知識は全て仕様記述およびその周辺の文書に明記され、かつ、追跡性が確保されたため、開発途中から参加した開発者や開発後に参加した保守要員がシステムや対象領域に関する知識を学習することができた、といった回答が複数事例から得られた。出荷までの工期が短縮されたと回答した事例（[HS03]）もあった。

⑩ 上流工程で記述した仕様は、以降の工程で修正されたか。もし修正されたとすると、それはどのような理由でそうなったか。その修正は、従来手法をそのまま導入していた場合とどのような違いがあるか。

⑪ 形式手法を導入している場合の方が、導入していない場合と比べて、修正量が減少したと評価することができるか。それが、プロジェクトの QCD 向上効果に与える影響をメトリックス評価しているか。

表 3-10: ヒアリング項目⑩および⑪に対する各事例からの総合的な回答

ヒアリング対象プロジェクト		下流工程での仕様への修正				
		開発組織外からの修正要求		開発組織内での修正要求		
		要求提示側		設計		テスト
		従来手法よりも少なかった	柔軟に対応できた	実装上の制約からくる修正要求があった	曖昧性の指摘はほとんどなかった	テストによる問題指摘があった
HS01	FeliCa ファームウェア(1)	○		○	○	○
HS02	FeliCa ファームウェア(2)	○		○	○	○
HS03	BPS1000 紙幣検査機	○	○		○	○
HS04	SHOLIS	○			○	○
HS05	iFACTS	○			○	○
HS06	TradeOne	○			○	○
HS07	オランダ軍 メッセージ分析	○			○	○
HS08	CAVA	○			○	○
HS09	MULTOS CA	○			○	○
HS10	Tokeneer	○			○	○
HS11	オランダ花市場	○			○	○

多くの事例で、要求提示側からの仕様修正要求は他手法での開発と比較して少なかったとのことであった。[HS03]では、そのような修正要求が発生した場合においても柔軟に対応できるように、仕様記述の抽象化や整理が行われたとのことであった。また、開発組織内での修正要求に関しては、設計からの曖昧性の指摘はほとんどなかったとのことであった。[HS01]および[HS02]からは、実装上の制約からくる修正要求があったとの回答が得られた。

⑫　形式手法導入を決定した管理者あるいはプロジェクトリーダーはどのような思いで導入したのか。

表 3-11: ヒアリング項目⑫に対する各事例からの回答

ヒアリング対象プロジェクト		形式手法導入の思い						
		開発組織内の意思疎通	高品質な仕様記述	開発資産の継承	調達の要件	工期短縮	ツールの利用	プログラミング技術との親和性
HS01	FeliCa ファームウェア(1)	○	○					○
HS02	FeliCa ファームウェア(2)	○	○	○				
HS03	BPS1000 紙幣検査機		○			○		
HS04	SHOLIS				○			
HS05	iFACTS				○			
HS06	TradeOne		○			○		
HS07	オランダ軍メッセージ分析			○				
HS08	CAVA	○						
HS09	MULTOS CA				○			
HS10	Tokeneer				○			
HS11	オランダ花市場						○	

開発組織内の意思疎通に関して、[HS01]および[HS02]では、開発者が意思疎通する上で基準となる辞書的な文書が必要であり、そのような辞書を作成することを可能にする方法論として形式手法が導入されたとのことであった。[HS08]では開発者が遠隔地に分散しているため、システムに関する共通理解を構築するための基礎として形式仕様を位置づけたとの回答が得られた。

高品質な仕様記述に関しては、[HS01]および[HS02]ではアドホックな仕様規定では開発遂行が困難であり、明確で信頼できる仕様を規定することが重要であるという認識があったとのことであった。[HS03]では従来の仕様が複雑化しており、複雑性にどう取り組むかが遂行責任者の懸案であったとの回答が得られた。[HS06]では開発組織にとって新規領域で

の開発であり、形式手法によって対象領域の知識を整理することが必要であると認識されていたそうである。

開発資産の継承に関して、[HS02]は[HS01]を継承した開発であり、[HS01]の成果物を有効に利用したいという要望があったとのことであった。[HS07]では開発者が形式手法に熟練しており、その技術を有効に発揮するために形式手法を適用したとのことである。

この他にも、開発の要件として形式手法の適用が要求されていた、限られた期間の中で成果物を出荷するために形式手法を適用した、形式仕様記述のための記述環境や管理ツールを導入することによる開発効率の向上を図った、形式仕様言語はプログラミング言語と似ており、開発者に受け入れられやすいと期待した、などの回答があった。

⑬　自然言語処理の観点でどのようにして誤解を最小化したか。

表 3-12: ヒアリング項目⑬に対する各事例からの回答

ヒアリング対象プロジェクト		自然言語処理の観点から誤解を最小化するための主な工夫			
		抽象化による実装用語の排除	自然言語の用語を識別子として採用	自然言語の用語と識別子の対応付け	自然言語による説明の併記または注釈
HS01	FeliCa ファームウェア(1)			○	○
HS02	FeliCa ファームウェア(2)	○	○		○
HS03	BPS1000 紙幣検査機	○	○		○
HS04	SHOLIS	○	○		○
HS05	iFACTS	○	○		○
HS06	TradeOne	○	○		○
HS07	オランダ軍 メッセージ分析	○	○		○
HS08	CAVA	○	○		○
HS09	MULTOS CA	○	○		○
HS10	Tokeneer	○	○		○
HS11	オランダ花市場	○	○		○

多くの事例が、抽象化による実装用語の排除を工夫として挙げたが、特に[HS02]では仕様記述を隠蔽関数で抽象化することで、仕様記述から実装に関する用語を排除したとのことであった。また[HS01]以外からは、対象領域における自然言語での用語を識別子として採

用することで、対象領域の知識との結びつきを明確にしたとの回答が得られた。また[HS01]では、対象領域の用語が日本語であったが、識別子としては英語を用いたため、対象領域での日本語用語と識別子の対応関係を管理したとのことであった。全ての事例が、形式仕様は適切な単位で自然言語による説明と併記され、レビューされたと回答した。

3.2「作業項目２:日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出」に関するヒアリング結果

① 日本語をプロジェクトの上流工程でステークホルダ間の共通語として使用するにあたり、誤解を生じさせないようにコミュニケーションするためにどのような訓練を行ったか。

表 3-13: ヒアリング項目①に対する各事例からの回答

ヒアリング対象プロジェクト		誤解を生じないコミュニケーションのための訓練		
		最初に用語集をある程度用意した	関数名と述語間の対応関係表	日本語識別子の整合性
HS01	FeliCa ファームウェア(1)	○	○	
HS02	FeliCa ファームウェア(2)	○		○
HS06	TradeOne	○	○	

ヒアリングした全ての事例が、最初に用語集をある程度用意したと回答した。また、形式仕様内の関数名と日本語文書の述語間の対応関係表を保守し続けた、日本語識別子を用語集での用語と整合性が取れるように定義した。[HS02]では各種の述語も隠蔽関数という形で日本語の用語として形式仕様中に利用した、という回答もあった。

② 上流工程でステークホルダ間での誤解を最小化して仕様を記述したにもかかわらず、後工程において誤解に基づくと推定される問題点が発見された場合、それが何故、何処で見過ごされてしまったのか。また、その対応策をどのようにとっているか。

表 3-14: ヒアリング項目②に対する各事例からの回答

ヒアリング対象プロジェクト		誤解を生じた問題点	
		定義と注釈の乖離	仕様アニメーションでのチェック漏れ
HS01	FeliCa ファームウェア(1)	○	○
HS02	FeliCa ファームウェア(2)		○
HS06	TradeOne		○

全ての事例で、機能仕様の妥当性検証が仕様アニメーションによって行われていたが、まれにチェックで見逃される場合があり、設計へエラーが波及したものの、システムテストの時点でテストケースが拡充し、仕様上の誤りが発見されたとの回答が得られた。また[HS01]では、形式仕様記述内の定義に対して日本語の注釈を与えていたが、まれに注釈と形式仕様記述の間に齟齬があり、それがレビューをすり抜けた。理想的には形式仕様記述と注釈の両方を読んで設計者は開発を行うべきだったが、片方の記述だけを読んで設計し問題が発生したことがあったとのことであった。

③ 上記①と②で認識した「上流工程でステークホルダ間での誤解を最小化することが必要」という問題を解決する技術的解決策として、共通語として日本語を用いて仕様記述をしており、日本語自体が曖昧性を持つが故に、それの影響を最小化するために何らかの手法を用いているか。用いている場合、形式手法をベースにしているか。ベースにしている場合、形式手法をあまり意識しないで仕様記述できるようにこの方法論をどのようにまとめ、関係者が使用できるようにしているか。

表 3-15: ヒアリング項目③に対する各事例からの回答

ヒアリング対象プロジェクト		形式手法の採用を意識させない方策				
		形式手法	日本語注釈	日本語識別子	表の派生	ラベル付け
HS01	FeliCa ファームウェア(1)	○	○			
HS02	FeliCa ファームウェア(2)	○	○	○	○	○
HS06	TradeOne	○	○	○		

形式手法をベースにした解決策を導入した、形式仕様記述に対して日本語による注釈を行った、形式仕様中の識別子を日本語の用語を用いて付けた、形式仕様から表形式の記述を派生させた、条件式に日本語のラベルを付けた、といった回答が得られた。

④ 上記③の方法論は、何も工夫していない通常の日本語を使用して仕様を記述する場合と比較し、どのような成果を上げているか。

表 3-16: ヒアリング項目④に対する各事例からの回答

ヒアリング対象プロジェクト		通常の日本語仕様に対する③の方法論の効果				
		記述単位の明確化	正確な注釈の維持	不要な注釈の排除	暗黙的仕様の排除	機械的な検証
HS01	FeliCa ファームウェア(1)	○	○		○	○
HS02	FeliCa ファームウェア(2)	○	○	○	○	○
HS06	TradeOne	○	○		○	○

全ての事例が、形式仕様言語の導入によって記述単位と責任範囲が明確になった、定義に対して個別に日本語による注釈を行い、注意喚起を行うことで注釈の正確性を向上させた、構文や型の検査や仕様アニメーションによる自動テストを実施することで、仕様記述の信頼性が向上した、と回答をした。この他には、定義と注釈の齟齬を減らすために、仕様に対する不要な注釈を排除した、疑似コードによる機能仕様にはしばしば暗黙の知識を要求するが、形式的に定義することでそのような暗黙的な仕様を排除し一貫性が向上した、といった回答が得られた。

⑤ 上記③の方法論は、その導入における問題点として何が認識されており、それに対してどのように対応しているか。

表 3-17: ヒアリング項目⑤に対する各事例からの回答

ヒアリング対象プロジェクト		導入時に認識された問題点と対応策				
		新言語の導入		可読性の向上		教育リソース不足
		日本語注釈	言語機能の選択	日本語識別子	表形式の派生	経験者の招聘
HS01	FeliCa ファームウェア(1)	○	○			
HS02	FeliCa ファームウェア(2)		○	○	○	
HS06	TradeOne			○		○

新しい言語を導入し、これまで使った事のない仕様記述をきちんと読んでもらえるかが課題だった際には、形式仕様の中の式に日本語で注釈をつけてレビューしたとのことであった。[HS01]ではプログラミングから見慣れない構成要素を多用せず、簡易プログラミング言語のように利用して、仕様を記述し、[HS02]では形式手法に慣熟していたため、より仕様記述に適した言語機能を採用したとのことであった。

形式仕様の読み手からの可読性の課題に対しては、識別子に日本語を用いることで、問題領域の用語を直接用いて仕様を記述し、形式仕様からテストケースとして表形式の文書を

派生させ、これにより形式仕様言語を知らないステークホルダによるテスト項目のレビューが容易になったとのことであった。

日本語による教育リソースの不足に対しては、外部から経験者を招いて、サンプルの記述を依頼したとのことであった。

⑥ 背景に形式手法を持つ構造化された日本語での仕様作成のための方法論として、今後どのように展開しようとしているのか。

⑦ 特定のドメインに対する DSL 的アプローチ、及び一般的な方法論としてのアプローチは、それぞれどのようになされているか。

⑧ 技術的解決策としての今後の見通しはどのようなものか。

表 3-18: ヒアリング項目⑥、⑦および⑧に対する各事例からの総合的な回答

ヒアリング対象プロジェクト		形式仕様からの派生	関数型言語的側面を生かした活用	日本語語彙ベースの構造的仕様記述
HS01	FeliCa ファームウェア(1)	N/A	N/A	N/A
HS02	FeliCa ファームウェア(2)	○	○	
HS06	TradeOne			○

今後の展開に関して、[HS01]は[HS02]に引き継がれているが、[HS02]では、形式仕様記述から様々な情報を抽出・加工して、DSL を含む様々な形式の文書を派生させる、VDM は関数型言語の一種と見なす事もできるが、関数型言語としての特性を生かして仕様をより簡潔に記述する、といった回答が得られた。また[HS03]は、多くのプロジェクトでは疑似コードのようなもので機能仕様が記述されており、問題領域の用語から日本語識別子として形式仕様記述することで、疑似コードによる機能仕様を作成していた工程を日本語の語彙をベースにしつつ構造的な仕様記述として記述するように改善していくとのことであった。

4. 個別事例分析結果

本章では、個々のヒアリング対象事例毎に、

1. プロジェクトの特性
2. プロジェクト概要
3. 開発プロセス概要
4. 記述と支援ツール
5. 厳密な仕様記述の効果
6. 厳密な仕様記述を適用するための工夫と効果
7. 顧客とのコミュニケーション
8. 開発者内のコミュニケーション
9. 教育

という観点から、ヒアリング結果および当該事例に関する公開された情報を再整理した上で、個々の事例についての分析の結果を示す。

なお、本章における事例毎の説明は、出典を明記していない限りは全て、今回の対面ヒアリングにより得られた情報、および対面ヒアリング時に提示されたドキュメントから得られた情報である。

4.1 FeliCa ファームウェア(1)

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS01	FeliCa ファームウェア(1)	フェリカネットワークス 株式会社	開発責任者	2012/03/30	東京都

4.1.1 プロジェクト特性

工期	2004-2006
要求記述言語	日本語
仕様記述言語	VDM++, FSP, 日本語
設計言語	UML, 日本語
実装言語	C++
開発	日 フェリカネットワークス株式会社
ドメイン	組み込み

4.1.2 プロジェクト概要

FeliCa チップは首都圏の PASMO や全国の JR 系各社の IC カードに用いられたり、所謂お財布ケータイの内部に埋め込まれたりして用いられている中核素子である。

本プロジェクトでは初期の IC カード型の FeliCa チップのリリースに引き続き、新たに開発された移動機用の FeliCa チップ（所謂モバイル FeliCa チップ）の内蔵ファームウェアが開発対象となった。

図 4-1 に示したものがモバイル FeliCa の全体図である。本プロジェクトで開発対象となったモバイル FeliCa チップは、図の左下に水色で示された（モバイル FeliCa IC チップ）部分である（参考文献 17 より）。

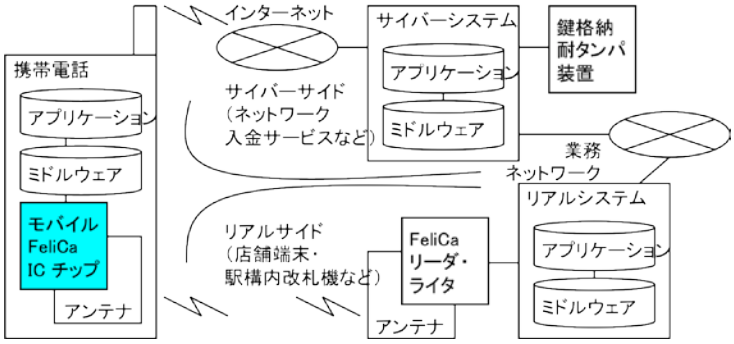


図 4-1: モバイル FeliCa 全体図

一種の組み込み機器であることに加えて、一度市場に出回ると回収が困難であること（事実上不可能）、金銭を扱うため間違いが社会的に大きな責任問題を引き起こすことなどから、極めて高い信頼性が要求される部品である。

第一世代（カード）の FeliCa チップは旧来の手法で開発されたものの、第二世代（モバイル）の開発に際して、機能の増強に伴い大幅な信頼性向上が求められていた。

プロジェクト本格開始に先立ち、開発リーダーは信頼性向上に関する様々な調査を行い、有力な手法の 1 つとして形式手法の存在に出会った。いくつか形式手法が存在する中で、VDM++が採用されたのは以下の理由からである。

- 実機のない仕様策定段階でもアニメーションによる仕様の検証ができる
- 開発上流工程における開発者同士のコミュニケーションと相互理解向上が可能
- 文法を持った言語として記述されていることから、解析が容易で仕様を多方面から精査できる
- 普通の開発者（特に数学的な訓練を受けていない技術者）でもある程度の努力で使うことができる

VDM++の採用を検討する際には、大学の形式手法専門家や VDM++コンサルタントの協力を得て、プロジェクトや開発対象の性質に対して検討が重ねられた。

4.1.3 開発プロセス概要

開発プロセスの全体像をデータフローダイアグラムで記述したものを下図に示す。

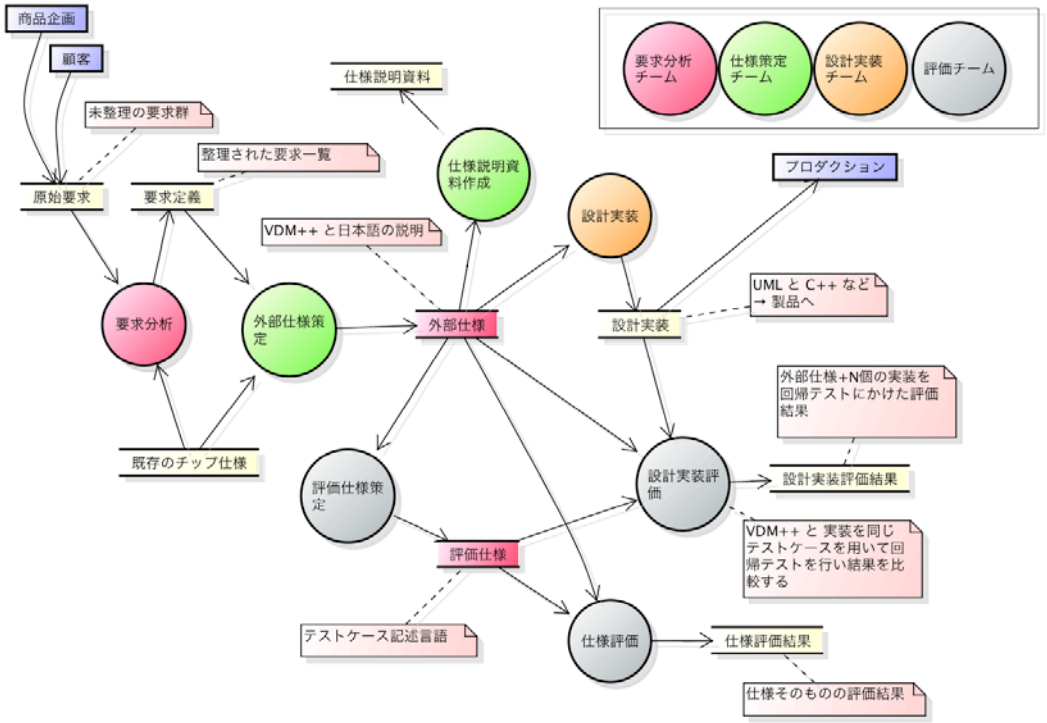


図 4-2: 開発プロセスの全体像[HS01]

最初の要求は「商品企画（社内）」または「顧客（社外）」からもたらされる。この段階では表などは用いられるものの完全に日本語のみで記述された未整理の要求記述である。

これらの記述を要求分析することにより、整理された「要求定義」へと展開する。この段階でも要求は表の形などに整理されているものの、特に厳密な定義がされているわけではない。

形式仕様作成が行われるのはこの次の段階である。図中「外部仕様」と書かれた部分がVDM++で記述された形式仕様の部分である。チップが提供するサービス（API）の仕様をVDM++で記述し、更に仕様アニメーションを可能にするために、陽仕様[4]が詳細に書き込まれたものとなっている。

もちろんこの VDM++で記述された「外部仕様」周辺には、全体の目的や構成、機能概要などを解説した日本語（自然言語）や UML などによる説明が付随している。プロジェクトを構成する主要成果物と形式仕様記述の適用箇所を以下の図に示す（水色の部分がVDM++による形式仕様適用箇所である）（参考文献 19 より）。

[4] その機能をどのように実現するのかを記述した仕様

図 4-3: プロジェクトを構成する主要成果物と仕様記述の適用箇所

このデータフローダイアグラムから読み取れるように、「外部仕様」はプロジェクトの中心に位置し、様々な用途に用いられている。

「外部仕様」そのものは単体での検証が行われている。ここでは記述された仕様の正当性の確認(verification)を仕様アニメーションによって仕様記述担当者が確認している。

本格的な検証は、独立した検証チームによって行われている。「外部仕様」をベースにまず「外部仕様」そのものをテストする評価仕様を作成し、十分仕様の妥当性確認(validation)を行ってから、同じ評価仕様を設計実装に適用し、仕様の評価結果と比較することにより設計実装の正当性確認(verification)を行っている。

顧客や企画に対して提示するための「仕様説明資料」は形式的に記述された「外部仕様」をベースに作成されている。元になる記述が形式的に厳密な記述であるため、正確な説明資料を作成することが容易になり、また作成時の抜け漏れも防ぐ事が可能となる。

設計実装作業に対する直接的な入力としても「外部仕様」は利用される。この場合設計者に対する極めて厳密な仕様が渡される事になる。本プロジェクトでは設計は人間が仕様を見ながら行っているが、記述のスタイルが統一され、既に「外部仕様」レベルで検証済の仕様が渡されることになる。

本プロジェクトでは、評価により問題が生じた場合、それが外部仕様に起因するものである場合には全て「外部仕様」の修正として反映されている。こうした開発プロセス上の規約により、常に最新状態の「仕様」がプロジェクトの成果物として反映され、多数の評価仕様を用いた回帰検証もそのたびに行われたため、製品の高い品質保持が可能となった。

4.1.4 開発体制

開発に際しては 3 つのチームが結成された

仕様チーム

設計チーム

評価チーム

の 3 チームである。以下の表にそれぞれのチームの役割を簡単にまとめた。

表 4-1: 各チームの役割

チーム	役割
仕様チーム	要求と概要仕様を受け取り、形式仕様として記述する。個別の仕様毎のアニメーションを行い、単体レベルでの検証を行う。単体レベルでの検証が終わった仕様を、開発チームと評価チームへ同時にリリースする。開発チームからは実装上の制約情報を受け取り、評価チームからは仕様に対する精査結果（仕様の評価）を受け取る。
開発チーム	仕様チームから形式仕様を受け取り、設計と実装を行う。実装プラットフォームが複数あるため、実際にはこのチームは複数存在する。設計上の制約から実現できない仕様がある場合には仕様チームへフィードバックを行う。成果物を評価チームに提示し、評価結果を待つ。
評価チーム	仕様チームから仕様を受け取ると、まず評価仕様の評価を行い、評価仕様自体の品質を確保したあと、外部仕様全体をアニメーションによって検証する。プロダクションレベルの回帰アニメーションを行い、仕様上の不備が見つかった場合には仕様チームへフィードバックを行う。 開発チームからは設計情報と実装を受け取り、評価を行う。このとき用いられる評価仕様は検証済のものである。

これら 3 つのチームの関係を以下に図示する。チーム間に描かれた矢印は主要な提示物である。

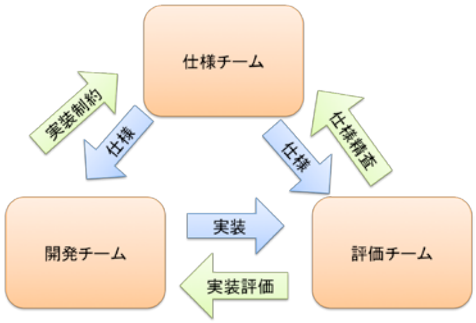


図 4-4: チーム間の関係

4.1.5 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた言語と主なツールを以下に挙げる。

表 4-2: 使用された言語と主なツール

言語	ツール	適用内容
VDM++	VDMTools	構文検査、型検査、仕様アニメーション
FSP	LTSA	モデル検査により設計の一部の安全性確認
自家製テストスクリプト	自家製ツール	テストケース記述言語を用いて、仕様(VDM++)と実装(C++)の両者を駆動。このテストケース記述言語は既存のプロジェクトでも利用されていてテストケースの資産が存在していた

VDMTools はインタプリタ方式で仕様アニメーションを支援する。このため記述や必要とするデータが大規模になるに従い、実行速度の低下などが課題になったが、幸いなことにVDM++コンサルタントならびに VDMTools[5]ベンダの協力により、ツールに対する様々な改良が適宜行われた。実行速度の改善は仕様アニメーションを用いた回帰テストの実行時間を 1 桁以上短縮し、より多量の評価仕様を適用することが可能になった。

4.1.6 厳密な仕様記述の効果

(1) 仕様の妥当性確認(validation)作業の早期化

VDM++による仕様アニメーションが可能な形にしていたため、外部仕様の評価仕様を直接評価してその結果を確認することが可能となった。このため新しい仕様が追加された際にもそのテストシナリオを反映した評価仕様を作成し、設計する前に仕様レベルでの確認を行うことができたので、仕様の妥当性確認を早い段階で行う事が可能になった。

特に最初の段階では、実チップは勿論、評価ボードなども存在しない段階であるため、仕様記述のみで記述と評価を行う事ができたことは価値が高い。

(2) 仕様説明資料作成の簡便化

外部に対して仕様を説明する際には、形式仕様記述そのものではなく、そこから半自動的に作成する形で、日本語や UML などの図版などが用意された。

こうした資料の作成時には、誤りや抜け等が混入しやすいものであるが、形式記述を元に作成する場合にはそうした間違いが混入しにくいという効果が期待される。実際、細かい定義部分等の文書は自動生成の対象となったため精度が高いものとなった。

[5] http://www.vdmtools.jp/modules/tinyd3/

(3) 仕様を開発関係者全員が共有する「辞書」とすることができた

「外部仕様」を中心において、要求が変更されたとき、評価で問題が報告されたとき、設計者などから実装制約による仕様の変更を指摘されたときなど、あらゆるシチュエーションで発生する仕様関連の変更の全てを 1 箇所で管理することが可能になった。

構成管理ツールによって「外部仕様」（すなわち VDM++によるテキスト）は管理されていたため、変更要求に応じて、どの仕様が変更されその結果がどの設計に反映されたかが追跡しやすい形になっていた。

ちょうど辞書を参照するように、仕様に関して調べたいときには「外部仕様」を手がかりに、全ての情報に辿り着き易い構造になっていたことが、プロジェクト内でのコミュニケーション向上に大きく役立っていた。

(4) 仕様に起因する不具合の低減

以下に示したのは仕様に関連した開発中の不具合を取り上げてその全体における割合を示したものである。

表 4-3: 不具合の分類と割合

分類	説明	全不具合中の割合
仕様記述漏れ	仕様策定者による記述漏れ	0.2%
仕様不明確	事前・事後・不変条件の記述ミス	1.8%
仕様見落とし	開発者による仕様見逃し	5.6%
仕様理解不足	開発者による誤読	10.7%
仕様変更通知不徹底	仕様策定者からの変更通知見逃し	0.2%

この仕様に関する不具合を集計すると、全体に占める割合は 18.5%である。これは従来の開発手法を採用した場合の平均 40%前後と比べると低い数値であるが、早い段階で仕様の誤りが取り除かれていることがこうした効果をもたらしたものでると考えられる。

ただ上の表の内部の内訳をみると、最初の 2 つ「仕様記述漏れ」「仕様不明確」に対して、次の 2 つ「仕様見落とし」「仕様理解不足」の割合がとても大きくなっていることがわかる。このことが意味していることは「仕様としてはきちんと書けているが、読み手にとっては読みにくい部分がある」ということである。

この「読みやすさの改善」というテーマに関しては別項である「4.2 FeliCa ファームウェア(2)」で取り上げる。

4.1.7 厳密な仕様記述を適用するための工夫と効果

(1) 仕様記述フレームワークの用意

API の外部仕様を記述するに際し、仕様記述のスタイルを統一化するために、仕様記述フレームワークが準備され、それを用いたサンプル仕様が作成された。

仕様記述フレームワークを用いた仕様が共通して利用する基盤部分の仕様も同時に策定され、そうした共通基盤があることを前提に作業が進められた。

仕様を読み設計を進めるためのいくつかのガイドラインが用意され、開発チーム内で共有された。

こうした工夫は、読み手書き手の双方に、スタイルを早く理解して貰うために役に立った。

(2) VDM++と日本語注釈の併記

VDM++とは別に日本語の説明資料が存在したことは既に述べたが、ここで言う日本語注釈は VDM++のソースコードに併記する細かいレベルのものであり、条件式などの意図を日本語で説明したものである。

厳密な定義は VDM++で記述されていたのであるが、条件式などに併記された日本語を抽出してレビューできることを狙ったものである。機械的に抽出が行えるように、その注釈には特別なマークアップが行われた。

こうした配慮が行われた理由は、VDM++の識別名や関数名がドメイン用語そのものではなく、英語で記述されていたため、一目で理解しにくいものであったからである。

当然日本語による注釈を併記することにより、仕様を議論する際の読みやすさは向上したが、日本語注釈と VDM++定義の微妙なズレが問題になることがときどき見受けられた。この問題は別項「4.2 FeliCa ファームウェア(2)」で再び取り上げる。

4.1.8 プロジェクト外部とのコミュニケーション

本プロジェクトでは顧客や商品企画などの外部（要求提示側）には VDM++による定義を提示していない。外部に対しては VDM++のソースから直接抽出整形された型やシグニチャの定義、ならびにソース内に注釈として埋め込まれた API の説明（日本語）などが「普通の文書」に埋め込まれて示され、その他に必要な日本語による説明や UML や様々な表が用いられた。

4.1.9 プロジェクト内部のコミュニケーション

要求を持ち帰った仕様チームは、要求内容を分析しながら VDM++によって「外部仕様」を記述した。

この仕様チームが書いた「外部仕様」は開発チーム、評価チームによって読まれ、「外部仕様」が共有資産として利用された。

4.1.10 教育

(1) 訓練形式

社内開発者ならびに外部委託開発者は全員 VDM++の読み方について訓練を受けた。仕様チームにアサインされたメンバーは社内、外部委託者を問わず書く為の訓練を受けた。これは OJT で行われており、特段記述に特化したクラスルームでの訓練を受けたわけではない。

(2) 職種や教育的背景の影響

開発に参加したエンジニアは、皆それまで特別に形式手法に関する訓練を受けた事がないメンバーばかりであった。仕様チームの最初のメンバーはドメイン知識を豊富に持っていた為、最初の仕様作成時の全体構成などに役立てることができた。

ドメイン知識をあまり持たないメンバーに関しても、「外部仕様」が形式的にまとめられていることにより、急速にドメイン知識を吸収できる教育効果が見受けられた。

4.1.11 プロジェクト参加者からの感想

以下に事後収集されたプロジェクト参加者からの「感想」を参考のために付記しておく。

(1) プロジェクトマネージャ

プロジェクトの進捗をより客観的に可視化できる：これまでは「仕様記述完了」と言われてもそれがどの程度の品質のものであるかを客観的に判断するのは難しかった。

仕様の早期検討による安定化：早い段階から仕様の検証を進めることができ、具体的で検証された振舞を使ってステークホルダと検討できるため最後の最後で仕様が大幅に変更される心配を減らすことができる。

(2) 仕様策定者

全体に共通することだが、自然言語による仕様書の場合まず一読してわからず、それを確認する作業を通して、厳密ではないレベルにも関わらず「わかったつもり」になることが多かった。これに対して形式仕様記述言語を用いる場合には、具体的な「振舞」を事前・事後・不変条件や陽仕様の中に記述しなければならないので、「なんとなくわかる」という状態がなくなる（減る）。

仕様に関する質問を受け、間違いや読みにくい記述を発見した場合には、仕様記述レベルで改善し全員で共有することができる。言葉で書いてあるだけではわかりにくくてもテストケースが添えてあれば動作の理解が促進される。こうした改善によって、同じ箇所に複数の開発者や評価者から繰返し似たような質問を受けることもなくなった。

(3) 開発者

開発者の視点から批判的に仕様を読むことにより、間接的に仕様策定に参加しているよう

な気持ちになれた。

(4) 評価者

自然言語による仕様書しか存在しない場合、評価者は頼れるものがないため非常に不安であり、かつテストの品質も上げにくい。形式仕様記述を用いることにより、早い段階からブラックボックステストのテストケースの検討を始めることが可能になり、かつ仕様エミュレータの構築と活用などにより、より網羅性の高い評価を行うことができた。

4.2 FeliCa ファームウェア(2)

ヒアリングセッション番号	ヒアリング対象プロジェクト名称	ヒアリング対象組織名	ヒアリング対象者の役割	ヒアリング日	ヒアリング場所
HS02	FeliCa ファームウェア(2)	フェリカネットワークス株式会社	開発責任者	2012/04/19	東京都

4.2.1 プロジェクト特性

工期	2008-2012
要求記述言語	日本語
仕様記述言語	VDM++, 日本語
設計言語	UML, 日本語
実装言語	C++
開発	日 フェリカネットワークス株式会社
ドメイン	組み込み

4.2.2 プロジェクト概要

別項「4.1. FeliCa ファームウェア開発(1)」で説明したプロジェクトは、製品開発として成功裏に終了した。その後の FeliCa チップの普及を受けて、より高度なサービスや頑健なセキュリティが求められるようになってきた。

先行する製品は VDM++による形式的な仕様も含めて、継続的に保守が続けられてきたが、新しい仕様を大量に投入したチップを開発する段階に至り、別項で取り上げた「読みやすさの向上」という課題を踏まえて、仕様の記述形式に関して改善を行うことになった。

本プロジェクトは多くの特性を別項「4.1 FeliCa ファームウェア開発(1)」と共有しているため、本項では主に先行するプロジェクトに対して加えられた変更の部分に関しての記述と分析を行う。

なお対象のチップはまずカード用のものが開発され、それを拡張する形でモバイル用のものが開発されている。報告書を記述している段階では、詳細設計と実装が進んでいる最中であるので、細かい定量的データは集められていない。

4.2.3 開発プロセス概要

開発プロセスの全体像をデータフローダイアグラムで記述したものを下図に示す。

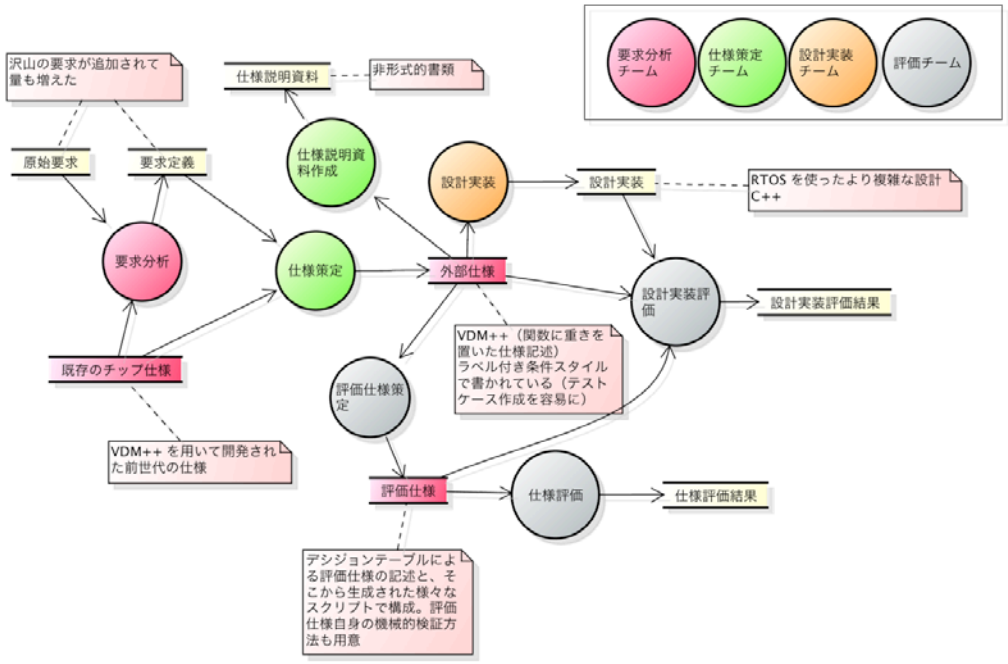


図 4-5: 開発プロセスの全体像[HS02]

大きな流れそのものは、別項「4.1 FeliCa ファームウェア開発(1)」で説明したものと変わらない。前回は「既存のチップ仕様」は自然言語で書かれた普通の仕様書であったが、今回は「既存のチップ仕様」が前回開発された VDM++によるものであるということが異なっている。

この既存の形式仕様と、新しい要求を用いて、新たな「外部仕様」が再び VDM++で策定された。

この新たな「外部仕様」は先行プロジェクトで課題になっていた読みやすさの改善へのアプローチを試みたものである。その意味で今回のプロジェクトにおける課題はプロジェクト管理上の工夫ではなく、あくまでも記述そのものに対する工夫であった。

以下に開発を進める際に掲げられた課題を挙げる。

表 4-4: 開発時に掲げられた課題

課題	工夫
実行仕様における仕様と設計の分離	「実行可能性の影響」の低減：仕様アニメーションのためのコードと仕様そのもののコードの明確な分離
	「実装の影響」の低減：仕様の記述そのものを問題領域に登場する用語だけを限定して用いる
レビューとテストの用途を考慮した仕様記述	レビュー用途：概要と詳細を階層化、簡潔な構造
	テスト用途：仕様記述から体系的にテスト項目を抽出する方法

本プロジェクトでは、評価により問題が生じた場合、それが外部仕様に起因するものである場合には全て「外部仕様」の修正として反映されている。こうした開発プロセス上の規約により、常に最新状態の「仕様」がプロジェクトの成果物として反映され、多数の評価仕様を用いた回帰検証もそのたびに行われたため、製品の高い品質保持が可能となった。

4.2.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた記述と主なツールを以下に挙げる。

表 4-5: 使用された記述と主なツール

記述	ツール	適用内容
VDM++	VDMTools	構文検査、型検査、仕様アニメーション
デシジョンテーブル	自家製ツール	VDM++記述の仕様（事前、事後条件）から同値類や境界値を体系的に抽出して、仕様のテストケースを網羅的に検討／確認できるようにした
自家製テストスクリプト	自家製ツール	テストケース記述言語を用いて、仕様(VDM++)と実装(C++)の両者を駆動。このテストケース記述言語は既存のプロジェクトでも利用されていてテストケースの資産が存在していた

今回は仕様からテストケースを体系的に抽出する仕掛けであるデシジョンテーブルの存在が大きい。

4.2.5 厳密な仕様記述の効果

(1) 仕様レビューの効率化

仕様記述形式の工夫により、以前に比べて仕様レビューの効率が一段と良くなった。

(2) テストケース検討の効率化

上記の仕様記述の工夫の一部にはテストケース抽出を行いやすくするためのものも含まれていて、それにより以前に比べて体系的かつ効率のよい仕様テスト項目の抽出が可能になった。

4.2.6 厳密な仕様記述を適用するための工夫と効果

(1) 仕様記述形式の改善 – 「実行可能性の影響」の低減

前回のプロジェクトにおける仕様記述では、仕様記述の中に、仕様アニメーションのための仕掛けが入り込んでいるところがあり、仕様を読む開発者はその点に関して注意を払う必要があった。

これは仕様として間違っていたという意味ではなく、過剰に設計者の発想を縛るような記述が入っている部分があったということである。この点に関しては純粋に「性質」を書く部分を関数として分離し、全体的に VDM++の関数型言語としての特性を生かす方向で記述を見直す事により改善が図られた。

(2) 仕様記述形式の改善 – 「実装の影響」の低減

上記の「実行可能性の影響」とも関連しているが、記述を仕様として設計者に読み取って欲しい「伝達部」、読んで欲しくない「非伝達部」に分離し、隠蔽関数という名前をつけた中間関数を用意することにより、あるレイヤーから上の記述が問題領域の言葉だけで構成されるように工夫した。

前回同様に仕様を読み設計を進めるためのいくつかのガイドラインが用意され、開発チーム内で共有された。

(3) 日本語識別子の利用

前回のプロジェクトでは識別子には全て英語を用いたため、設計への変換はそのまま素直に流用すれば良かった（それが設計者へ過剰に先入観を与える結果ともなった）が、今回は仕様内で利用される関数名に日本語識別子を利用した。

問題領域の用語を用いて述語を表現した関数は、問題領域の専門家にとって理解しやすいものであるためレビュー効率が向上した（日本語を使ったというよりも、日頃検討に使っている「問題領域の言葉」を使ったことによる効果が大きかった）。

特にプロジェクト内で「ラベル付き条件」という形式の採用がテストケースの体系的抽出とも関連している。

下図は旧来の記述方式と、新しい「ラベル付き条件」と呼ばれる記述の違いを簡単に示したものである。

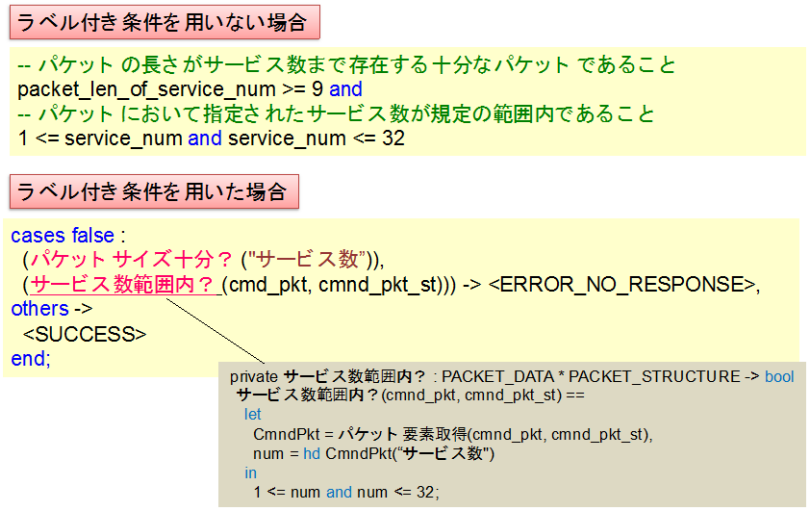


図 4-6: 記述の違い(参考文献 20 を元に作成)

ラベル付き条件を用いない場合には、条件が単にその場所に展開されていて、その意味を日本語の注釈と合わせて読みとる必要があるが、ラベル付き条件を用いた場合には関数名（＝述語）そのものがその意図を反映しているので、注釈などを添える必要はない。またその関数そのものの定義は簡単に参照でるような仕掛けが用意されていて、関数名をクリックするとその定義をすぐに見る事ができるようになっている。

4.2.7 プロジェクト外部とのコミュニケーション

プロジェクト外部とのコミュニケーションで大きく変わったところは、テストシナリオの検討の一環としてデシジョンテーブルが使えるようになったことである。デシジョンテーブルは既に説明したように、新しく工夫された仕様記述から体系的に抽出できるため、説明資料の作成の効率も高くなり、外部とのコミュニケーションも改善した。

もともと外部には VDM++そのものは見せていなかったので、これは形式記述に基づき「厳密かつ理解しやすい記述」を作り出した例として考えることができるだろう。

以下の図はデシジョンテーブルのイメージを示したものである。このデシジョンテーブルは前述のラベル付き条件記述から体系的に抽出することが可能である。

条件	同値分割	境界値	TC1	TC2	TC3	TC4
実行可能モード ?	TRUE	0..2	Y		Y	Y
	FALSE	3		Y		
パケット サイズ十分?	TRUE	10	Y	Y		Y
	FALSE	9			Y	
サービス数範囲内?	TRUE	1..32	Y	Y	N	
	FALSE	0, 33			N	Y
結果	正常応答		Y			
	応答なし エラー			Y		
	応答あり エラー				Y	Y

テスト ケースの網羅性を視覚的に確認することができる。レビューが容易になる。

図 4-7: デシジョンテーブルのイメージ(参考文献 20 を元に作成)

この図では「視覚的に確認」と書かれているが、現在はこのテーブルの網羅性などを機械的に検証し実行可能なテストケースを生成するツールが作成されている。

4.2.8 プロジェクト内部のコミュニケーション

内部でのコミュニケーションは引き続き VDM++と日本語を用いたものであるが、仕様記述方法の改善によりレビュー効率が改善されている（前回のプロジェクトでは、日本語の注釈と VDM++のコードが一致しているかどうかにも気を配らなければならなかったが、今回は問題領域の用語を使った関数そのものが記述されているので、レビューを行いやすくなっている）。

4.2.9 教育

教育という観点では、言語教育を施す際に、新しい仕様記述形式についての解説は行うものの、その意味は実際に現場で利用すれば程なく了解できるようなものであるため、特別な教程が用意されているわけではない。

またデシジョンテーブルの導出や、ツールの利用に関してはいずれも基礎的なテスト設計教育を受けていたり、実例をみて学習可能であったり、あまり難しい要素は含まれていない。

4.3 BPS1000 紙幣検査機

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS03	BPS1000 紙幣検査機	Aarhus University	技術コンサルタント	2012/04/10	オランダ

4.3.1 プロジェクト特性

工期	1997
要求記述言語	自然言語
仕様記述言語	VDM-SL, UML, 自然言語
開発	独 GAO 社
ドメイン	組み込み

4.3.2 プロジェクト概要

BPS1000 紙幣検査機のセンサー統合制御ソフトウェア SIC-2000 の開発に VDM-SL が適用された事例である。紙幣検査機ではより多種な紙幣の検査に対応するため、多くのセンサーを統合して扱う必要がある。新しいセンサーを導入するために制御ソフトウェアを変更するなどして、センサー制御が複雑になりがちである。そこで独 GAO 社は自社技術者が VDM-SL の教育を受け、その技術者が SIC-2000 を VDM-SL で仕様記述し、開発を遂行した。

他の多くの成功事例では形式手法導入の動機として安全性や信頼性、ミッションクリティカルであることが挙げられているが、この事例では開発コストの低減が主目的であった。すなわちこの事例は、複雑なシステムを経済的に開発するために形式手法を導入し成功した事例であると言える。

4.3.3 開発プロセス概要

この事例については開発プロセス全体に関する公開情報が不足しているため、データフローダイアグラムによるプロセスの記述は行わない。以下に、開発プロセスのうち形式仕様と関わった部分について概要を説明する。

この開発は受託開発ではなく、自社製品の制御ソフトウェア開発であり、既存製品での経験で直面した問題を克服することが課題であった。ここでの開発の焦点は、センサーに関する構成の自由度の高さからくる複雑性だった。紙幣検査のロジックを自由に定義できることは、多くの国、多くの種類の紙幣を扱う紙幣検査機の商品としての重要な魅力であり、また、サードパーティー製センサーを統合することも必要であった。この複雑性を仕様記述によって解決する必要があった。

そのため、センサー管理に関する自然言語で記述された要求仕様から、VDM-SL による形式仕様を記述した。システム内のデータトラフィックや処理速度等の性能面での制約を満たすよう仕様の改善が進められ、より抽象度のある仕様が策定されていった（参考文献 13 より）。

仕様の改善の例としては、数値表現の抽象化が行われた。このシステムの複雑さの要因の 1 つはセンサーの数値表現であった。センサーによって異なるフォーマットで数値データが表現されていた。初期の仕様ではこれを型変換によって表現していたが、これは仕様を複雑にすることが分かった。そこで、数値そのものとデータ表現を分離して定義すること

で仕様記述の多くの部分から数値表現というディテールを隠蔽した。これにより、抽象度の高い記述を可能にしながら、センサー毎に異なる数値表現を個別に定義できる自由度が実現された。

また、ユーザ定義可能なセンサー構成データ(センサー表)はメンテナンス時において誤動作の原因の 1 つであり、その表が「正しく」設定されていることが重要であった。このセンサー表が満たすべき条件は、VDM-SL の仕様では不変条件として記述された。これにより、テーブルの構成条件が散逸することなく記述することができた。そして、この不変条件はセンサー表の設定ルールとしてシステムの説明書に自然言語で記載された。

以下に各工程での工数を表で示す。

表 4-6: 各工程での工数

工程	工数
仕様記述	12 人月
実装	3 人月
制御ソフトウェア(SIC-2000)モジュールテスト	1 人月
システム統合(BPS1000)テスト	N/A

なお、モジュールテストでは数個の誤りが発見された。システム統合テストでは、形式仕様を適用した部分に関する誤りは発見されず、改善要請もなかった。

4.3.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた言語と主なツールを以下に挙げる（参考文献 13 より）。

表 4-7: 使用された言語と主なツール

	VDM-SL
ファイル形式	ソースファイル
構文/型検査	VDMTools
静的検証(証明等)	-
動的検証(テスト等)	VDMTools
自動生成	-

4.3.5 厳密な仕様記述の効果

(1) 抽象度が高く密度が高い仕様記述

数値表現の具体的フォーマットを隠蔽し抽象度の高い表現が可能になったため、簡潔で密度が高い仕様記述が可能になった。

(2) メンテナンスコストの低減

各モジュールが不変条件を明示しているため、メンテナンス時の改変等が比較的容易になる。

また、VDM-SL 仕様をプロトタイプとして実行することで、トラブルシューティングでどのような作業が発生するかを仕様記述の段階で評価することができた。それを参考に仕様記述を改善していくことで、メンテナンスコストの低いシステムの構築が可能になった。

(3) 早期の性能評価推定

データトラフィックや処理速度といった性能面の重要な評価を仕様記述段階から推定することが可能になり、実装からの手戻りを事前に防ぐことができた。

4.3.6 厳密な仕様記述を適用するための工夫と効果

形式手法による開発コスト低減を狙うため、軽量な開発プロセスを採用した。証明や段階的詳細化といった工程を経ずに、抽象度が高く、かつ、厳密な表現が可能で、インタプリタ実行が可能な形式仕様を中心に開発を進め、ツールによるテストと合わせて、開発工数を削減することができた。

4.3.7 顧客とのコミュニケーション

センサー表に関する制約条件の記述を自然言語に翻訳し、それを説明書に記載することでシステム中の不変条件（センサー表が満たすべき条件）が顧客であるユーザに伝えられた。

4.3.8 開発者間のコミュニケーション

VDM-SL 仕様は開発プロセスの工程を超えて活用され、通常であればより後の工程において行われる作業が、VDM-SL による仕様記述を使って前倒しされて遂行された。例えば、センサー表の不変条件はマニュアル作成工程において自然言語に翻訳されて説明書に記載された。また、メンテナンス作業の策定は VDM-SL 仕様を実行することで、実装を待たずに策定された。さらに、性能評価についても実装される前に VDM-SL 仕様からデータトラフィックや処理速度が見積もられた。

4.3.9 教育

プロジェクト開始時には、開発組織には形式手法に熟達した技術者はいなかった。VDM-

SL 専門家から 1 週間のコースを受け、その後、その VDM-SL 専門家のコンサルテーションを受けた。

4.4 SHOLIS

ヒアリングセッション番号	ヒアリング対象プロジェクト名称	ヒアリング対象組織名	ヒアリング対象者の役割	ヒアリング日	ヒアリング場所
HS04	SHOLIS	Altran Praxis Limited	Principal Engineer	2012/04/03	英国

4.4.1 プロジェクト特性

工期	1995-1997
要求記述言語	英語
仕様記述言語	Z 記法，英語
設計言語	SPARK (Ada サブセットに表明を加えたもの)
実装言語	SPARK
開発	英 Praxis 社
ドメイン	艦載ヘリコプターの航空管制

4.4.2 プロジェクト概要

英国海軍の艦載ヘリコプターを運用する際の時間制約などを管理するためのシステムの開発プロジェクトである。軍用航空機の安全性に直結するシステムであり、英国国防省の安全規格 Def Stan 00-55[6]に準拠するための高い安全性と検証可能性が要求されていた。すなわち、仕様が安全要件を満たすこと、および、実装が仕様を満たすことを形式的に論証可能であることが求められた。この要求に応えるために、Z 記法による仕様記述と SPARK 言語による設計および実装が採用された。すなわち、形式手法を採用したのは顧客側の要求によるものだった。

開発企業はそれまでも Z 記法による開発の経験があり、社内に形式手法の専門家がいた。開発企業による過去の開発では Z 記法による証明はあまり行われなかったが（参考文献 4 より）、このプロジェクトでは Z 記法による証明を行うことで、誤りを早期に発見することができた。これは結果として、Z 記法による証明を通して、開発側が開発対象に関する正確な理解をより早期に深めることができた事例であると言える。

[6] 英国国防省が定めている安全規格で、安全に関わるソフトウェア開発への形式手法の適用について規定している。

1995 年当時としては非常に挑戦的なプロジェクトであり、開発プロセスに関してもシンプルな構成で工夫をし、形式手法の効果的な応用を切り拓いたプロジェクトである。

4.4.3 開発プロセス概要

開発プロセスの全体像をデータフローダイアグラムで記述したものを下図に示す。

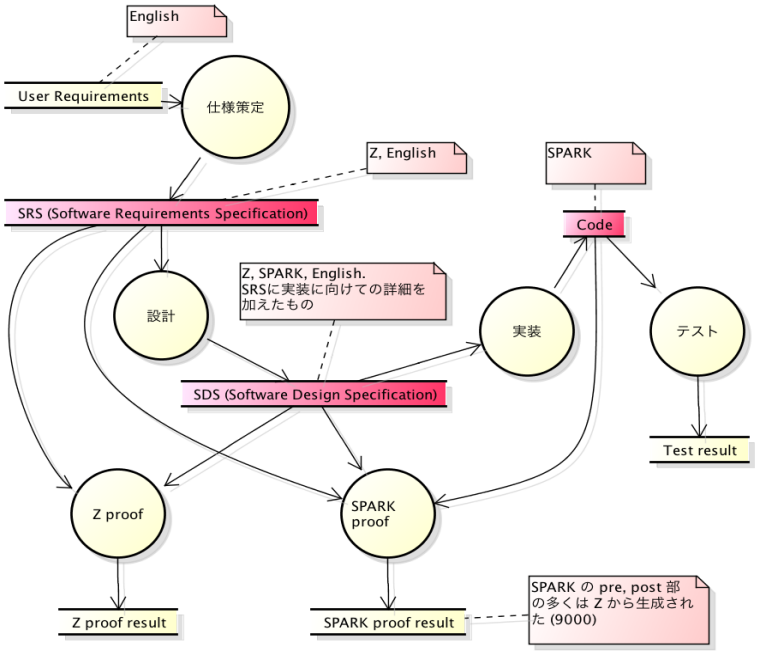


図 4-8: 開発プロセスの全体像[HS04]

ユーザからの要求(図中では User Requirements)が英語で仕様策定者に伝えられ、Z 記法およびその補足説明となる英語による仕様(図中では Software Requirements Specification, SRS)が記述された。仕様は、同じく Z 記法で記述された安全項目に対して証明による検証が行われた。約 130 個の証明が Z 記法による仕様記述に対して行われた。

Z 記法による詳細化の技術によって、Z 記法での設計を派生し、更に SPARK で pre, post, assert, return といった論理的表明や、global, derives, own, inherit といった情報のフローに関する表明の注釈記述を伴う設計を派生させた。詳細化の各ステップや、Z 記法による設計から SPARK による設計への派生に関する正当性の証明が行われた。Z 記法による設計に対しては約 20 の証明が行われ、SPARK による設計に対しては、3,100 個の機能要件や安全要件に関する証明が行われた（参考文献 4 より）。また、SPARK の実装に関しても、スタックサイズや入出力帯域、ループ停止条件が静的分析により検証された。

Z 記法および SPARK に関する証明はピアレビューの対象となった。また、SPARK の実装に関する証明は、ピアレビューに加えて、開発社内の別チームによるレビューを受けた。

従来手法と同様の単体テストおよび結合テストも行われ、全工程のうち約 37%の工数が当てられた。テストにより、証明や静的分析では検出できなかった誤りを検出し、最終的には安全性を確保して実行時検査を取り除き、出荷した。

4.4.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた言語と主なツールを以下に挙げる。

表 4-8: 形式手法に関して利用された言語と主なツール

	Z 記法	SPARK
ファイル形式	LaTeX ファイル	ソースファイル
構文/型検査	Fuzz	SPARK toolset
静的検証(証明等)	HOL/Z	SPARK toolset
動的検証(テスト等)	-	SPARK 実行時検査
自動生成	-	証明課題

4.4.5 厳密な仕様記述の効果

(1) 高品質

Z 記法の仕様記述により主要な安全項目を満たすことが検証され、また、SPARK による設計および実装が Z 記法による仕様記述を満たすことも証明および静的解析により検証された。これにより、Def Stan 00-55（英国国防省の安全規格）およびその SIL4[7]で要求される高い安全基準を満たすことができた。

(2) 誤りの早期発見(1)

Z 記法による仕様記述は型検査器にかけられ、記述者によって誤りの早期発見が行われた。データや機能や操作に関する型の矛盾を早期に発見することで、仕様記述の基本的な整合性を確保することができた。

[7] Security Integrity Level 4

(3) 誤りの早期発見(2)

Z 記法による仕様記述に関する証明によって、重大な誤りを早期に発見することができた。下に各工程における誤りの発見に関するデータを引用する。

Project phase	Faults found (%)	Effort (%)
Specification	3.25	5
Z proof	16	2.5
High-level design	1.5	2
Detailed design, code & informal test	26.25	17
Unit test	15.75	25
Integration test	1.25	1
Code proof	5.25	4.5
System validation test	21.5	9.5
Acceptance test	1.25	1.5
Other*	8	32

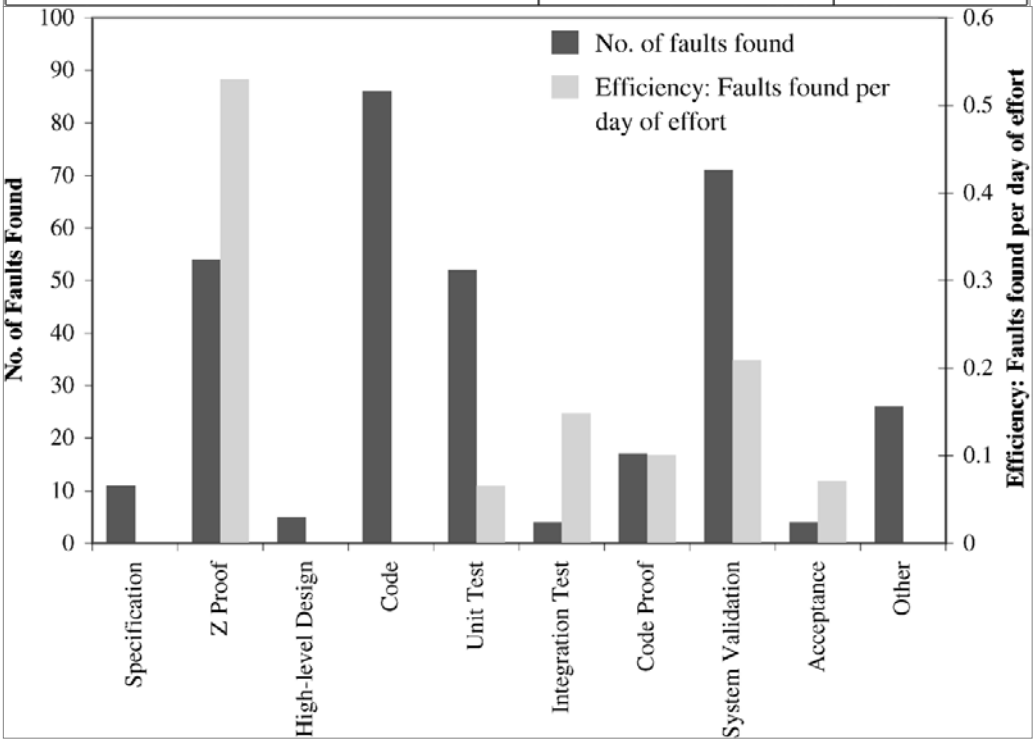


図 4-9: 誤りの発見に関するデータ(参考文献 4 より引用)

4.4.6 厳密な仕様記述を適用するための工夫と効果

(1) 異なる言語間での追跡性の確保

仕様記述と設計で異なる言語を用いたために、異なる言語間での追跡性を確保する必要があった。また、仕様記述言語と設計／実装言語では使用できるデータ型が異なっているために、そのギャップを埋める必要があった（参考文献 4 より）。

追跡性を確保するため、仕様記述で用いられた Z 記法上の識別子と設計および実装で用いられた SPARK 上の識別子をできる限り統一させた。双方の言語の間で同じ変数について異なる型が用いられた場合には、SPARK 上での表明としてその制約条件が記述され、ほとんど誤り混入の元とはならなかった（参考文献 4 より）。

(2) **検証コストの低減**(1)

システム全体が安全基準 SIL4 の対象であったのではなく、SIL4 対象部分と非 SIL4 部分があった。そこで SIL4 準拠部分と非 SIL4 部分を機能的に分離し、静的検証によって分離を検証することで、機能要求に関する検証を SIL4 部分に対してのみ行った。

(3) **検証コストの低減**(2)

型検査を Z 記法の仕様および SPARK による設計全てに対して行うことで記述時の誤り発見に大きな効果を得た一方で、仕様記述および設計に対する検証項目をキーとなる安全項目に絞った。

4.4.7 顧客とのコミュニケーション

顧客と開発者の間では自然言語が用いられた。顧客はシステムが満たすべき仕様および安全要件を自然言語で示し、開発者はそれらを Z 記法で記述し、Z 記法で仕様が主要な安全要件を満たすことを検証した。ここで開発者は仕様に関する証明を行うことでドメインおよびシステムについてより深い理解ができ、それを顧客とのコミュニケーションに役立てることができた。

4.4.8 開発者内でのコミュニケーション

Z 記法による仕様記述から SPARK による実装まで異なる詳細化レベルでの記述があり、開発者はその作業工程に応じた詳細度での記述を参照した。

4.4.9 教育

仕様記述および証明を行ったのは、プロジェクト開始時には既に形式手法に熟達した技術者だった。実装者に対して 1 週間の形式手法のコースを受講させ、形式的記述の読み方の教育を行い、現場での指導を行った。

4.5 iFACTS

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS05	iFACTS	Altran Praxis Limited	Principal Engineer	2012/04/03	英国

4.5.1 プロジェクト特性

工期	-
要求記述言語	英語
仕様記述言語	Z 記法，状態機械(UI 記述)，英語
設計言語	SPARK (Ada サブセットに表明を加えたもの)
実装言語	SPARK
顧客	英 NATS 社
開発	英 Altran-Praxis 社
ドメイン	航空管制

4.5.2 プロジェクト概要

英国上空の航空路は欧州内外からの航空機が通過する交通量の大きな空域である。英国および北大西洋東部の航空管制を担う英 NATS 社は次世代の航空管制システムの一部として iFACTS の開発を開始した。飛行経路予測と中期衝突検出に基づいて航空管制官の業務を支援しすることで管制官の管制処理能力の向上が期待されている（ホームページ[8]から引用）。iFACTS は単独の航空管制システムではなく、既存の航空管制システムと併設統合される。2011 年 11 月に全機能が完成した。

開発企業はそれまでも Z 記法による開発の経験があり、社内に形式手法の専門家がいた。このプロジェクトでは社内の形式手法専門家だけでなく、顧客のドメイン専門家等が形式手法を学習し、仕様のレビュー等に参加した。

このプロジェクトはまだ詳細は公開されておらず、本調査においても開発内容については非常に限定的な情報しか得られなかった。一方、このプロジェクトでは顧客も含め大量の関係者に形式手法の教育が行われたことから、開発プロジェクトにおける形式手法教育として重要な事例であると考えられる。よって、本報告では教育を中心にした分析を記す。

[8] http://www.altran-praxis.com/atm.aspx

4.5.3 開発プロセス概要

開発プロセスの全体像をデータフローダイアグラムで記述したものを下図に示す。

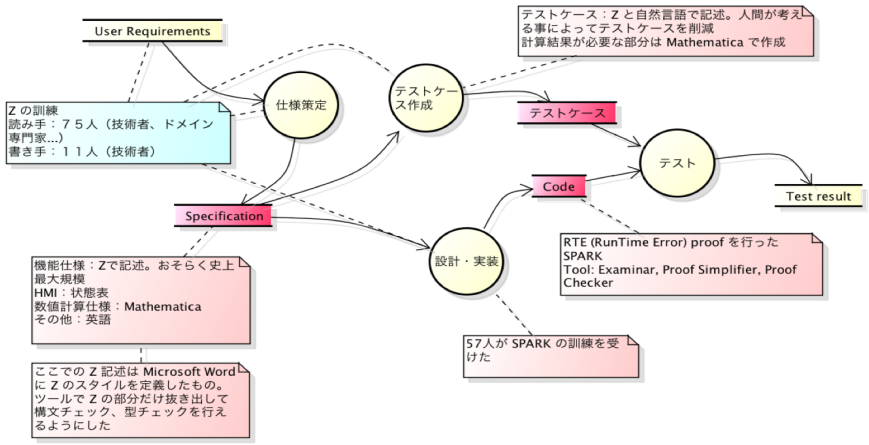


図 4-10: 開発プロセスの全体像[HS05]

ユーザからの要求(図中では User Requirements)が英語で仕様策定者に伝えられ、Z 記法およびその補足説明となる英語による仕様(図中では Specification)が記述された。ここで記述された仕様は非常に規模が大きく、Z 記法で記述された仕様としては最大規模と言われる。また、ユーザインタフェースの仕様としては状態表が用いられた。

実装は大部分が SPARK により記述された。Z 記法による仕様記述は SPARK コードに詳細化され、ソースの規模は約 15 万行となった。SPARK コードに対しては実行時例外に関する証明が行われた。

実装をテストするための参照実装が、5 名の技術者チームによって Mathematica[9]で構築された。また、Z 記法による仕様からテストケースが人手により作成され、参照実装の動作と付き合わせる形で実装に対するテストが行われた。テストケースは人手による慎重な吟味を経て、仕様記述から機械的に求まる状態数に比較して非常に少ないケース数に抑えられた。

[9] 数式処理システム
http://www.wolfram.com/mathematica/

4.5.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた言語と主なツールを以下に挙げる。

表 4–9: 使用された言語と主なツール

	Z 記法	状態表	SPARK
ファイル形式	Microsoft Word	–	ソースファイル
構文/型検査	fuzz	–	SPARK toolset
静的検証(証明等)	–	–	SPARK toolset
動的検証(テスト等)	–	–	Mathematica, SPARK 実行時検査
自動生成	–	–	–

このプロジェクトにおける形式手法に関する問題点としてツールの不足が挙げられる。Microsoft Word(以降、MS Word と略す)による Z 記法の仕様記述は記述量が増えるにしたがってパフォーマンスの低下が問題となった。また、バージョン管理や構成管理も困難であり、このプロジェクトで要求された仕様記述の規模に対して不足していた。

4.5.5 厳密な仕様記述の効果

(1) 誤りの早期発見(1)

Z 記法による仕様記述は MS Word から簡単な操作で型検査器にかけることができた。また、型検査で型エラーが発見されると MS Word 文書にエラーメッセージへのハイパーリンクが設定された。データや機能や操作に関する型の矛盾を早期に発見することで、仕様記述の基本的な整合性を確保することができた。

(2) 誤りの早期発見(2)

SPARK で記述された設計および実装の型安全性の証明は必須とされ、毎夜再証明が行われた。

4.5.6 厳密な仕様記述を適用するための工夫と効果

(1) 受け入れられやすい編集環境

このプロジェクトでは顧客が形式手法を学習し開発に参加した。そこで MS Word へのア

ドオンとして、Z 記法による形式的仕様記述を MS Word ドキュメントに埋め込んだ型検査ツールへの簡単なインタフェースを提供した。これにより、より多くの人に受け入れられる仕様記述編集環境を提供することができた。反面、仕様記述の規模が大きくなると処理速度の問題が顕在化した。

(2) Z 記法と英語の併記

MS Word 文書に Z 記法による仕様記述と英語による補足説明を併記することができた。単に併記できるだけでなく、Z 記法での仕様記述で定義された識別子名などへのリファレンスを埋め込むことができた。

4.5.7 顧客とのコミュニケーション

顧客は Z 記法および SPARK のトレーニングを受け、顧客側のスタッフが開発に参加した。形式的仕様記述や設計／実装について直接読み書きを行った。

4.5.8 開発者間のコミュニケーション

外部委託開発者は Z 記法および SPARK のトレーニングを受けて開発に参加した。形式的仕様記述や設計／実装について直接読み書きを行った。

4.5.9 教育

このプロジェクトでは非常に多くの顧客および外部委託開発者に対して Z 記法および SPARK の教育が行われた。以下、表にまとめる（参考文献 8 より）。

表 4-10: 実施された教育

	職種	人数	コース期間	習熟期間
Z 記法読み	技術者,ドメイン専門家,航空管制官	75	3 日	1 週間
Z 記法書き	技術者	11	3 日	3 ヶ月
SPARK	技術者、顧客側技術者	57	-	即使えた。但し、実行時例外に関する証明作業には 2 ヶ月かかった

(1) 職種や教育的背景の影響

このプロジェクトでは所属として顧客や外部委託、役割として開発者やドメイン専門家など、非常に多種多彩な背景を持つ関係者に形式手法の教育が行われた。本調査のヒアリン

グでは、学習者の教育的背景(大学での専攻など)は形式手法の習得には大きな障害にはならならず、むしろドメイン知識を持っていることは習得に非常に役立った、というコメントを得た。

(2) 実作業による習熟

SPARK を習得した技術者は全員 Z 記法を読むための教育も受けた。SPARK による設計および実装についてはクラスルーム形式の教育を受講後、直ちに実務に着いて作業を遂行することができた。

SPARK を扱う作業としては、実行時例外に関する証明があった。この作業にあたるためには、2 ヶ月の習熟期間を要した。

(3) Z 記法／SPARK 以外の記法やツールに関する教育

Z 記法および SPARK 以外には、このプロジェクトではユーザインタフェースに関する仕様記述として状態表が用いられた。状態表については特に教育は必要なく、直ちに理解された。

また、テスト用の参照実装のために Mathematica が使われたが、5 人のみがトレーニングを受けた。

(4) 教育コストの影響

開発側企業は、教育コストは大きな問題ではなかったとしている。多種多様な背景を持つスタッフに対して教育が行われたが、大きな困難はなかったとしている。例えば Z 記法のコースは 3 日間であり、一度受講すればそれで十分であり、コストとしても安上がりだったとしている。

4.6 TradeOne

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS06	TradeOne	SCSK 株式会社 (旧 日本フィッツ株式会社)	開発責任者	2012/04/27	東京都

4.6.1 プロジェクト特性

工期　　2003-2004

要求記述言語　　日本語

仕様記述言語　　VDM++, 日本語

設計言語	UML，設計用共通フレームーク利用，日本語
実装言語	C++
開発	日　SCSK 株式会社（旧 日本フィッツ株式会社）
ドメイン	証券業務

4.6.2 プロジェクト概要

証券会社のバックオフィス機能を提供する TradeOne パッケージの開発に際し、一部サブシステムの開発に形式仕様記述言語 VDM++が採用された。

開発対象となったのは

- マル優
- オプション取引

の 2 つのサブシステムである。パッケージ全体の開発としては、普通のオブジェクト指向開発手法（UML+繰り返し開発）が採用されていたが、上記のサブシステム開発を引き受けたメンバーの発案により形式仕様記述の採用が決定された。

基本的に外販用パッケージを開発する目的でプロジェクトが進んでいたため大部分の要求は社内の証券ドメイン専門家を交えた商品企画部門から出されていたが、最初の顧客（証券会社）のためのカスタマイズ要求も一部含まれていた。

当時、証券業務の複雑な処理は永年の経験によって暗黙的に蓄積され、設計、実装、検証されていたが、形式仕様記述を利用することによって、業務ルールが明示的に記述され、それまで証券業務を知らなかった設計実装者も、急速に問題領域の知識を得て、仕様の矛盾を指摘できるようになった。

4.6.3 開発プロセス概要

開発プロセスの全体像をデータフローダイアグラムで記述したものを下図に示す。

マル優の開発とオプション取引の開発は異なる所もあるが、全体として似通った部分の方が多いため、以下 1 つのプロジェクトとして説明を行うことにする。

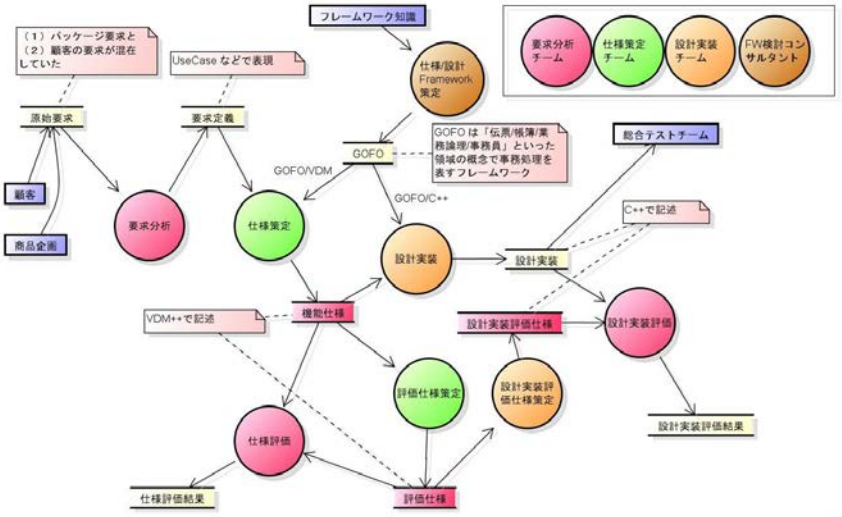


図 4-11: 開発プロセスの全体像[HS06]

原始要求は大部分社内の商品企画からきているが、一部最初のパッケージ顧客からのカスタマイズ要求も取り込まれたものになっていた。こうした要求は要求分析チームによって主にユースケース記述としてまとめられたが、まだ厳密さには欠けていたので、仕様策定チームがユースケースに対応するシナリオを起点に VDM++を用いた仕様策定を行い、仕様アニメーションの形で検証を行った。

記述された機能仕様をもとに、設計実装チームが C++を用いたシステムとして実装を行った。すなわち仕様を見ながら設計実装チームが手で設計実装を行ったわけであるが、この作業に際して、単純な仕様から設計への展開ガイドラインを作成する代わりに、仕様から設計実装への橋渡しを行いやすくするフレームワーク Gofo を準備して、実際の開発作業に適用した。

要求分析チームは証券業務の知識はあるが、形式仕様の経験はない。仕様策定チームは一部古い証券業務の知識はあるものの特に問題領域知識に対する強みはない形式仕様記述の専門家であり、設計実装チームは様々な実現手段を持っていたものの、証券業務の知識も形式手法の経験もなかった。FW 検討コンサルタントは形式手法の知識と事務処理業務の知識を持っていたが、証券業務の知識は持っていなかった。

表 4-11: チームとその特性

チーム名	特性
要求分析チーム	証券業務の知識を持つ専門家。形式仕様の経験はなし。抽象化を通したモデリングの経験もほとんど持っていなかった。
仕様策定チーム	形式仕様記述の専門家。 一部のメンバーは形式仕様記述に関して深い知識は持つものの証券業務知識は皆無であり、また一部のメンバーは古い証券業務知識をもっていたが、形式仕様記述に関し

	ては当初深いレベルには達していなかった。
設計実装チーム	設計／実装の専門家。実装言語を問わず様々な実現手段を身につけていた。 形式仕様記述の経験も証券問題領域の専門知識も持ち合わせていなかった
FW 検討コンサルタント	設計実装のためのアーキテクチャを検討し、形式仕様から設計実装の流れの中で利用しやすいように、アーキテクチャをフレームワークの形に落とし込んだものを提供した。ある程度の形式仕様記述の知識と、幾つかの事務処理問題領域に関する知識は持ち合わせていたが、証券業務そのものの知識は持ち合わせていなかった

4.6.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた記述と主なツールを以下に挙げる。

表 4-12: 使用された記述と主なツール

記述	ツール	適用内容
VDM++	VDMTools	構文検査、型検査、仕様アニメーション
Gofo	フレームワーク	設計実装のためのフレームワーク。当時はまだ珍しかったウェブアプリケーション開発のためのフルスタックのフレームワークをまず設計し、それと VDM++仕様との対応関係を示すことにより仕様から設計への円滑な橋渡しを可能にした。 Gofo は事務処理に特化した問題領域の知識を表現する語彙（伝票、帳簿、業務論理、事務員など）を用いるため、仕様と設計の対応関係をとりやすい構造になっている。

4.6.5 厳密な仕様記述の効果

(1) 仕様の早期検証による妥当性確認（validation）の効率化

仕様アニメーションを通して早い段階でシナリオを通した仕様の検討ができたため、要求を出す側との意見調整と仕様に対する意思決定を素早く行う事ができた。

(2) 設計への円滑な橋渡し

既に説明したように、設計実装フレームワーク（Gofo）が用意され、仕様との対応関係を明確にするようにしてあった。VDM++による仕様に書かれた内容が、設計のどの部分に反映されるべきかが決まっていたため、仕様を設計に反映する作業も円滑に進み実装上のつまらないミスも発生しにくい形になっていた。

(3) 業務知識（問題領域知識）の移転

表 4-11：チームとその特性 からもわかるように説明したように、開発が始まった段階では要求分析チームのメンバー以外は証券業務に関しては深い知識を有していなかった。

しかし開発が進むにつれて、少なくとも開発対象業務の性質については深く理解できるようになっていった。これは要求に現れる業務を抽象化して厳密な記述として、皆が読めるような形になったことによるものであった。

その意味で形式仕様記述は、業務知識の効果的な移転ツールの役割も果たしたと言う事ができる。

(4) 高品質なシステムの実現

仕様アニメーションを用いた早期からの検証により、仕様レベルでの品質はとても高いものになった（設計、実装レベルの品質はフレームワークに作り込まれた）。仕様策定段階でも、繰り返し仕様アニメーションによる回帰テストが実施され、仕様の間違いが探され続けた。

このため最初の本番システムリリース後、本サブシステムに関しては本質的不具合が 1 件も報告されなかった（1 件だけマイナーな不具合が帳票レイアウト上に発見されたが、その部分は形式仕様記述の範囲外であった）。

(5) 低保守コストの実現

TradeOne には複数のサブシステムが存在していたため、他のサブシステムとの生産性が比べられた。この計算を最初に行ったときには、形式手法を用いた当サブシステムと、通常の開発手法を採用した他の 2 つのサブシステムの間では「リリースまでの生産性」は似通っていた。

このため形式手法はあまり効果がなかったのかという意見も出されたが、結局リリース後形式手法を用いたサブシステムでは 1 件も不具合が発生しないのに比べて他のサブシステムでは複数の不具合が発生し、パッケージ改修費用などがコストとして発生していた。

この意味で保守までを含んで考えた場合、本プロジェクトに関しては形式手法を用いたほうが保守コストを低くできることがわかった。

なお最初の開発における生産性には、それまで存在していなかった様々な準備（仕様記述の形式、フレームワークの検討と作成、仕様検証の為の枠組みの作成、仕様ライブラリの整備）などが含まれている。上記理由から更に生産性が高くなると予想される。

4.6.6 厳密な仕様記述を適用するための工夫と効果

(1) 日本語識別子の利用

2 つのサブプロジェクトのうち、2 つ目のオプション取引開発プロジェクトでは、VDM++の記述に日本語識別子が用いられた。1 つ目の英語名（と一部ローマ字表記）を用いた仕様記述に比べて格段に読みやすさが向上した。

このため問題領域の専門家にも読んでもらいやすくなった。

(2) 設計実装フレームワークとの対応

ソフトウェアシステムは最後には動作しなければならないため、仕様から設計に大きな負荷がかかるようでは、採用しにくいと感じるプロジェクトも多くある筈である。

本プロジェクトでは設計実装フレームワークと仕様の対応関係を明快に示してやることができたため、仕様の量に対する設計実装の工数の見積も行いやすくなった。

まだ仕様を起点にした作業の進捗度も計算しやすくなるため、プロジェクトのリスク管理なども比較的行いやすくなった。

4.6.7 プロジェクト外部とのコミュニケーション

プロジェクト外部とは、要求分析チームが日本語の文書によるコミュニケーションを行っていた。この文書は当初仕様の裏付けのない、図式や文章表現に過ぎなかったが、開発が進むにつれて VDM++を用いた仕様アニメーションによって検証されたシナリオを反映するようになっていった。

4.6.8 プロジェクト内部のコミュニケーション

プロジェクト内部でのコミュニケーションは当初ユースケースなどの UML モデルも併用しようとしていたが、次第に VDM++記述や Gofo を使った構造を基礎とするものに変わっていった。

VDM++記述は日本語識別子を使ったために、そのまま読んでも理解しやすいものになっていったが、意図に関する説明（「何故」こうするのか）に関しては日本語（自然言語による）注釈を併用していた。

仕様レベルで意図が説明されていれば、設計レベルではその意図を「どのように」実現するのかをフレームワークの業務論理部品などで表現するだけなので、あまり詳しい注釈は不要であった。

4.6.9 教育

この当時 VDM++に関しては日本語の教科書やセミナーも存在していなかったため、海外のコンサルタントを招いて英語で一週間のセミナーを受講した。英語の問題により、このセミナーでは若干名の脱落者が出たが、そののちの内部フォローにより読みに関しては何

とかできるレベルとなった。

他の多くの事例でも見られるように、一週間程の研修を受けた後は、読みに関しては実務投入後一週間ほど、書きに関しては 2〜3 ヶ月で円滑にこなせるようになっている。

4.7 オランダ軍メッセージ分析

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS07	オランダ軍 メッセージ分析	C.H.E.S.S. Group	開発責任者	2012/04/10	オランダ

4.7.1 プロジェクト特性

工期	1997-1998
要求記述言語	自然言語
仕様記述言語	VDM-SL,自然言語
実装言語	C++
開発	蘭 C.H.E.S.S.社
ドメイン	軍事, テキストマイニング

4.7.2 プロジェクト概要

この事例は、テキストマイニングシステムの一部として、複雑かつ自由度が高いテキストメッセージを定型データに加工しデータベースに格納するサブシステムの開発プロジェクトである。アーキテクチャとしては入力メッセージを解析するパーサ、定型データに変換するトランスレータ、定型データをデータベースに格納するデータベースバックエンドの 3 つのモジュールから構成された。

この開発では仕様記述言語および設計言語として VDM-SL が用いられた。開発者は顧客に形式手法を説明し受け入れさせるのではなく、開発内部で用いる仕様記述として VDM-SL を適用した。顧客との間では UML やデザインパターンを通して要求獲得や顧客レビューを行った。

実装は主に VDM-SL からの C++コード自動生成により行われた。システム全体のうちこのプロジェクトで開発したサブシステムのみが期日内に出荷され、計画予算内で終了した。また、出荷以来このサブシステムについての誤りは全く発見されていない。品質、コスト、出荷を高度に両立させた成功事例であると言える。

また、本報告書に挙げられる成功事例には、他にも顧客とのコミュニケーションを UML や自然言語等を用いた事例があるが、顧客側には安全基準やセキュリティ基準からくる形式手法への承認があった。本事例は、形式手法に関して顧客から直接の要求はなく、あく

まで開発者内部の記述として用いられ、成功した事例と言える。

4.7.3 開発プロセス概要

開発プロセスの全体像をデータフローダイアグラムで記述したものを下図に示す。

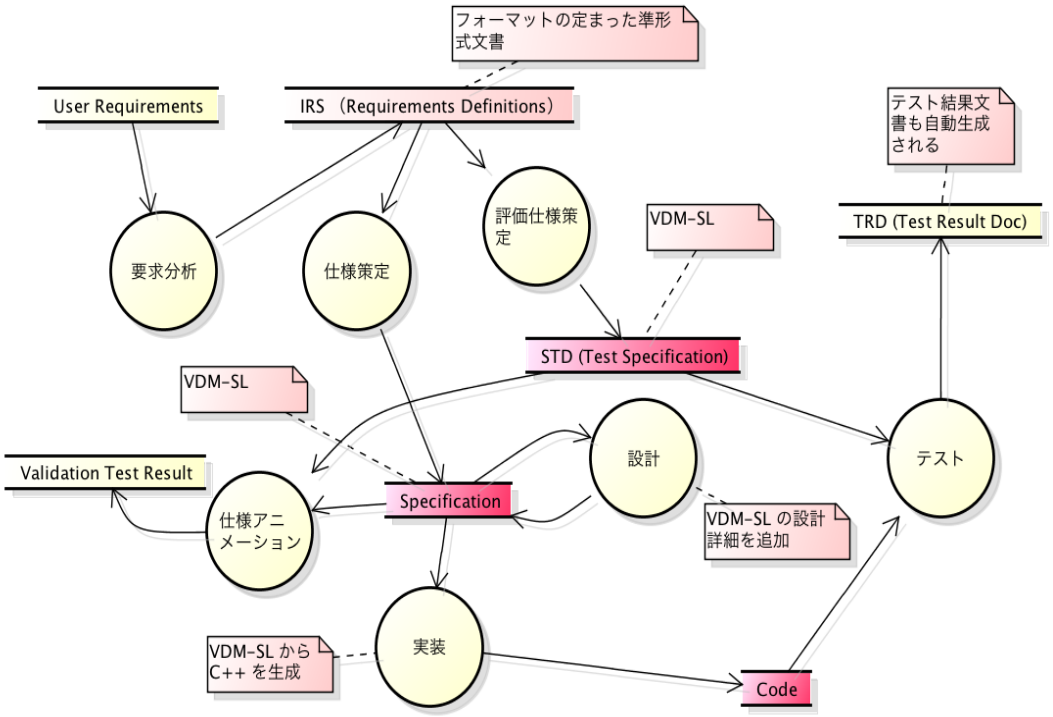


図 4-12: 開発プロセスの全体像[HS07]

ユーザからの要求(図中では User Requirements)が自然言語で要求記述者に伝えられ、インタフェース要求仕様(図中では IRS)が定義された。インタフェース要求仕様は定型的なフォームによる自然言語で記述された。すなわち、インタフェース要求仕様は形式的仕様記述ではないが、一定の定型化がされていたと考えられる。

インタフェース要求仕様からは、VDM-SL による形式的仕様記述が作成された。VDMTools には仕様を直接実行するインタプリタが搭載されており、記述された仕様のプロトタイプ利用や、後述のテストケースを実行して仕様の妥当性の検証が行われた。

テストケースは 100 万件のホワイトボックステストケースと、インタフェース要求仕様からブラックボックステストケースが VDM-SL で作成された。すなわち、同一のインタフェース要求仕様から、形式仕様としての VDM-SL 記述とブラックボックステストケースとしての VDM-SL 記述がそれぞれ独立に作成され、テストに用いられた。テスト結果から自動的にテスト報告が作成された。また、実装が行われると、VDM-SL 仕様を実装に置き換えてテストが行われた。

実装は主に、VDM-SL から C++コードが自動生成された。自動生成以外では、VDM-Tools とデータベースの間のインタフェースが人手で作成された。このインタフェースは自動生

成された C++コードのランタイムとしても用いられた。コードの自動生成の利用により、実装の工数を大幅に削減することができた。

4.7.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた言語と主なツールを以下に挙げる。

表 4-13: 使用された言語と主なツール

	VDM-SL
ファイル形式	ソースファイル
構文/型検査	VDMTools
静的検証(証明等)	-
動的検証(テスト等)	VDMTools
自動生成	VDMTools

このプロジェクトでは既存ツールだけでなく、VDMTools とデータベースのインタフェースが新たに作成された。このインタフェース作成に要した工数は 94 人時であり、このプロジェクト全体の工数 1,196 人時と比較すると 8%弱である。

4.7.5 厳密な仕様記述の効果

(1) 効果的な要求獲得

入力となるテキストメッセージは複雑な構造をしており、かつ、運用上様々な例外的な用法があった。インタフェース要求仕様ではそれらのテキストメッセージについて定型化された変換規則が記述されていたが、具体的な規則は自然言語で記述されていた。

開発者は、インタフェース要求仕様を参照して VDM-SL で実行可能仕様を記述する作業を通して、また、記述された VDM-SL 仕様を実際に実行することで、インタフェース要求仕様の自然言語による記述では不明確な点を探し出すことができた。顧客とのインタビューでそうした点を顧客側の専門家に問い合わせることで、顧客側専門家も意識していなかった例外的な規定を引き出す質問ができ、効果的な要求獲得を行うことができた。

(2) 実装コードの自動生成による生産性と信頼性の向上

VDM-SL で記述された形式仕様から、C++のソースコードを自動生成することで、生産性が大幅に向上した。実装コードは全体で約 94,000 行作成され、うち約 90,000 行は VDMTools により自動生成された（数値は参考文献 15 より）。これにより生産性が大幅に向上するとともに、形式仕様に対するテストケースがそのまま実装に対して適用され、信頼性の高い実装が得られた。出荷以来、このサブシステムからは誤りは発見されていない。

各工程と記述量、工数、生産性を以下に示す。

表 4-14: 各工程と記述量、工数、生産性

工程	記述量(行)	工数(人時)	生産性(行/人時)
仕様記述	15,000	1,196	13
人手による実装	4,000	471	8.5
自動生成による実装	90,000	0	-
テスト	-	612	-
全体	94,000	2,279	41.2

4.7.6 厳密な仕様記述を適用するための工夫と効果

(1) UML およびデザインパターンによる顧客とのコミュニケーション

開発者には VDM の専門家がおり、開発者内部では VDM-SL による仕様記述というプロセスや仕様の実行によって問題に対する理解を深めていく意図を持っていた。しかし顧客は形式手法には興味はなく、顧客とのコミュニケーションは UML およびデザインパターンを交えた自然言語が中心となった。この 2 つのコミュニケーションを分断せず、VDM 仕様によって得られた開発者の知見を顧客とのコミュニケーションに有効に生かすことができた。また、インタビューで判明した問題点を極めて短時間に修正することができた。これによって、形式手法の利点を生かしつつ、顧客の満足と信頼が得られた。

(2) VDMTools とデータベースのインタフェースの作成

このプロジェクトでは VDMTools の拡張機能として、データベースへのインタフェースが実装された。このインタフェースを使って、VDM-SL による仕様記述およびテストケースの作成を非常に早期から進めることができた。このインタフェースの作成には 94 人時がかけられたが、上記顧客との良質なコミュニケーションおよび VDM-SL 仕様の早期着手によるフロントローディングの効果は大きかった。

4.7.7 顧客とのコミュニケーション

顧客は形式手法に対する直接的な興味はなく、UML およびデザインパターンを交えた自然言語によるコミュニケーションが中心となった。

顧客からの定型的自然言語によるインタフェース要求仕様と、開発者からの UML やデザインパターンによるコミュニケーションが行われる一方で、開発者内部では VDM-SL による仕様記述が開発者にとっての問題の理解に非常に重要な働きを示した。そして開発者はインタフェース要求仕様の曖昧な点や矛盾などを指摘し、顧客を驚かせることがあった。そのような良質な対話を通して、インタフェース要求仕様が劇的に改善されていった。

4.7.8 開発者間のコミュニケーション

開発者は全員 VDM-SL の読み書きを行った。VDM-SL による仕様記述とテストケース作成は別々の開発者によって行われた。開発者は VDM-SL を通して開発対象に対する共通認識を持つ事ができた。

4.7.9 教育

このプロジェクトでは既に VDM-SL に熟達した開発者が参加した。

4.8 CAVA

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS08	CAVA	Aarhus University	技術コンサルタント	2012/04/10	オランダ

4.8.1 プロジェクト特性

工期	1998
要求記述言語	自然言語
仕様記述言語	VDM-SL,自然言語
開発	デンマーク Baan 社
ドメイン	言語処理系,制約解消系

4.8.2 プロジェクト概要

この事例は、制約記述言語 CAVA の処理系の開発に VDM-SL による仕様記述を適用した事例である。CAVA は製品販売時の構成部品の組合せに関する制約条件を記述する言語であり、このプロジェクトではその処理系が実装された。

この開発では、開発対象の複雑さと、開発チームが遠隔地に分散されているというリスクを持っていた。VDM-SL による仕様記述は複雑な開発対象を高い抽象度で厳密に定義することで、遠隔地にいる開発者間での共通認識の基盤となった。また、実行可能な VDM 仕様はプロトタイプとしても用いられ、共通認識の構築に役立った。

この成功事例は、オフショア開発やニアショア開発など開発拠点が分散された開発において大いに参考になると考えられる。

4.8.3 開発プロセス概要

この事例では開発プロセスの詳細に関する公開された情報は非常に限定されているため、開発プロセス全体のデータフローダイアグラムはこの事例については記述しない。

開発は米国、オランダ、デンマークの３ヶ所に分散して行われた。３ヶ所に分散しながらVDM-SL による仕様をテストケースと共に策定した。VDM-SL 仕様は VDMTools により実行されテストされた。また、自動生成されたプログラムのテストには、仕様の実行結果がテストオラクルとして使われた。

4.8.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた言語と主なツールを以下に挙げる。

表 4-15: 使用された言語と主なツール

	VDM-SL
ファイル形式	RTF (MS Word に埋め込み)
構文/型検査	VDMTools
静的検証(証明等)	-
動的検証(テスト等)	VDMTools
自動生成	VDMTools

4.8.5 厳密な仕様記述の効果

(1) 共通認識の確立

開発は 3 ヶ所の遠隔地に分散して行われた、VDM-SL 仕様の厳密な定義と実行可能仕様を通して、開発対象に対する共通理解を構築することができた。

(2) 実装コードの自動生成による生産性と信頼性の向上

VDM-SL で記述された形式仕様からコードが自動生成され、形式仕様のインタプリタ実行の結果と突き合わせる形でテストが行われた。

4.8.6 厳密な仕様記述を適用するための工夫と効果

(1) VDM 仕様の MS Word への埋め込み

VDM-SL で記述された仕様を RTF 形式で出力し、MS Word ドキュメントから参照される形で作成された。これによって仕様の各コンポーネントを別ファイルとして管理しながら、

1 つの MS Word ドキュメントとして見せることができた。

4.8.7 顧客とのコミュニケーション

VDM-SL で記述された仕様は MS Word に埋め込まれ、自然言語による説明と併記する形でまとめられた。VDM-SL で記述された部分の表示／非表示を切り替えて表示することでより広いステークホルダとのコミュニケーションが可能になった。

4.8.8 開発者間のコミュニケーション

開発者は全員 VDM-SL の読み書きを行った。VDM-SL による仕様記述を中心に遠隔開発拠点間での共通理解が得られた。

4.8.9 教育

開発者はコンパイラ作成などの対象ドメインに関する知識を持つものがいたが、VDM についてはこのプロジェクトが初の経験だった。4 日間のクラスルーム形式の教育と 1 日の実習が行われた。実習はコンパイラ作成をテーマとすることで、プロジェクトに直接役に立つ経験を得ることができた。

また、VDM 専門家のコンサルテーションを受けることで初めての VDM-SL プロジェクトを進めていくことができた。

4.9 MULTOS CA

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS09	MULTOS CA	Altran Praxis Limited	Principal Engineer	2012/04/03	英国

4.9.1 プロジェクト特性

工期	1997-2000
要求記述言語	UML, 構造化操作定義, 自然言語
仕様記述言語	Z 記法
設計言語	Z 記法, CSP
実装言語	SPARK(Ada サブセット+表明), Ada, C++, C, SQL
開発者	英 Praxis 社
ドメイン	セキュリティ

4.9.2 プロジェクト概要

この事例は Multos スマートカードの認証局を開発したプロジェクトである。スマートカードの認証局というシステムの特性上、非常に高いセキュリティ要求と処理量、操作性、そして分散システムからくる複雑性を同時に満たすことが求められた。特にセキュリティ要求として、英国情報技術セキュリティ評価基準(Information Technology Security Evaluation Criteria, ITSEC)レベル E6、すなわち形式手法の早期からの適用が求められた。その一方でプロジェクト予算の関係からハードウェアおよび OS は既製品を利用する必要があった。

この事例では、非常に多くのシステム記述技術が組み合わされて適用された。自然言語を除いても、要求獲得および仕様記述においては形式的記述と非形式的記述が併用され、設計では 2 つの異なる形式手法と非形式的設計により作成された。また、実装も SPARK, Ada, C++, C, SQL と多彩な言語で行われた。

4.9.3 開発プロセス概要

このプロジェクトは高いセキュリティ基準から要請される構成的正当性(Correctness by Construction)と呼ばれる詳細化と証明を基盤とする形式手法と、経済性から要請される既製品利用を両立するために、システム開発の各工程においてシステムの様々な側面をそれぞれ異なる記法で記述した。それぞれの記法は選択可能な中で最大限形式化された記法が選択された。多数の種類の記述が複雑な派生関係を持つ中で、形式手法の利点である検証可能性から各工程の各側面の記述の正しさの確信を得ながら開発が進められた。

開発プロセスを表わすデータフローダイアグラムを以下に示す。

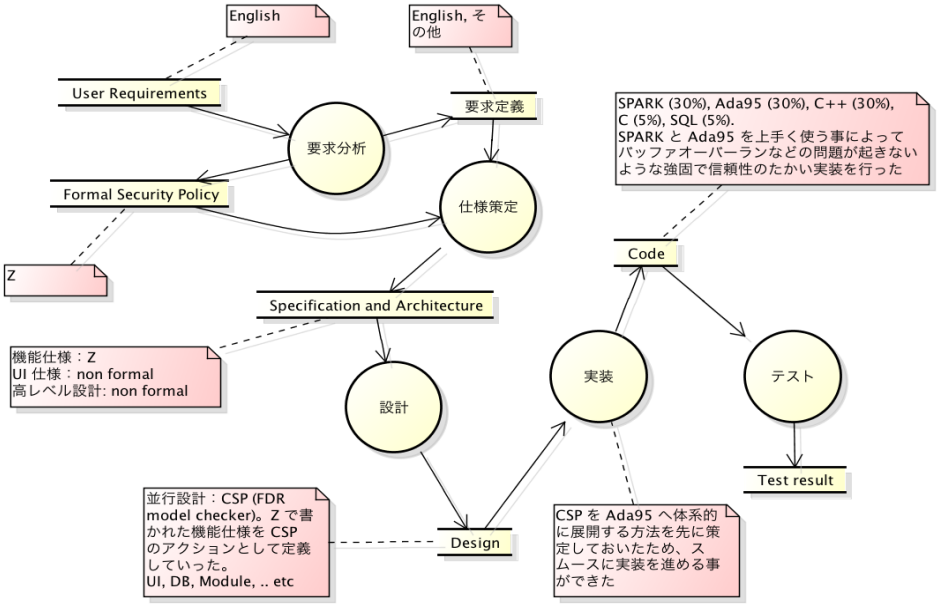


図 4-13: 開発プロセスの全体像[HS09]

要求分析においては、認証局の環境条件や目的を定義し、それに基づいてシステム要件を定義した（参考文献 6 より）。顧客の要求(図中では User Requirements)はまず英語で記述され、補足として UML や構造化操作定義が記述された。各要件は実装コードまで追跡

可能なようにラベル付けがされた。また、各セキュリティ要件から脅威まで追跡可能にし、これらの追跡性は全ての開発文書を通して実装コードまで確保された。

仕様策定においては、情報技術セキュリティ評価基準(ITSEC) E6 および情報セキュリティ国際評価基準(Common Criteria)の要請に基づいて、顧客からの非形式的セキュリティポリシーが定義する資産および脅威、対抗策が形式的セキュリティポリシーモデル(FSPM)として Z 記法で記述された。形式的セキュリティポリシーモデルは計 27 個の論理的表現として記述された（参考文献 6 より）。

また、ユーザインタフェース仕様ではシステムの表示などがプロトタイプ作成を通して定義され、そのプロトタイプは顧客側の認証局オペレータースタッフにより妥当性が検証された。

そして、要求定義や形式的セキュリティ基準とユーザインタフェースプロトタイプから、Z 記法によるトップレベル形式仕様(FTLS)が策定された。形式仕様の正しさは、型検査器や顧客レビューにより検証された。

高レベル設計として、計算機やプロセス上への機能の配置、データベース構造と保護機構、トランザクションおよびコミュニケーションの機構、などの視点からシステムの構成や構成部品間のコラボレーションが定義された（参考文献 6 より）。

高レベル設計の要点はセキュリティ要求を満たすところにあった。すなわち、既製品ハードウェアや基本ソフトを使いながらも高いセキュリティ要求を達成するために、セキュリティを既製品の機能に頼ることなく、ハードウェア分離、暗号化、メッセージ認証、セキュリティクリティカルプロセスの分離という高レベル設計上の手段で実現した。

設計工程では、Z 記法で記述されたトップレベル形式仕様に加えて実装に必要な情報として、プロセス設計などが記述された。プロセス設計は形式言語である CSP により記述され、また、CSP のアクションとして Z 記法で記述されたトップレベル形式仕様の機能仕様を参照した。CSP のモデル検査器である FDR によってデッドロックや並行実行に関する特性が検証された。

実装する上で重要だったのは製品のライフスパンであった。認証局として約２０年間稼働する必要があるため、商用ミドルウェアのような既製品部品の採用は、実用の範囲内でできる限り避けた。例えば RPC 機構の実装は DCOM や CORBA ではなくスクラッチから実装した（参考文献 6 より）。また、可用性 99.999%の達成（参考文献 6 より）や、設置場所のアクセス性(認証局という事情から、非常に堅牢な場所に設置された)といった要因から、高度な安定性が求められた。

セキュリティ要求や可用性要求と既製品 OS 利用を両立させるために、実装言語はそれぞれの実装箇所の適材適所で決められた。システムの基盤は Ada で実装され、セキュリティ要件が重要な箇所は形式手法での検証に適した SPARK(Ada サブセットに形式的な表明を拡張した実装言語)で記述された。GUI 部分は高レベル設計工程でセキュリティ要件への影響が無いよう設計され、マイクロソフト社の MFC を利用して C++で記述された。また、暗号関連のデバイスドライバやライブラリの C 言語ソースコードをレビューした上で利用した。これらの組合せによる開発にはリスクが想定されたが、小規模の実証コードを作成し検証することでクリアしていった。

4.9.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた主な言語と主なツールを以下に挙げる。

表 4-16: 形式手法に関して利用された言語と主なツール

	Z 記法	CSP	SPARK	Ada	C++/C
ファイル形式	LaTeX	ソース	ソース	ソース	ソース
構文/型検査	fuzz	FDR	SPARK toolset	ada	Visual C++
静的検証(証明等)	-	-	SPARK toolset	SPARK Examiner for Ada	BoundsChecker, PC-Lint
動的検証(テスト等)	-	FDR	SPARK 実行時検査, Visual Test	Visual Test, AdaTest	Visual Test
自動生成	-	- (但しシステマチックな手順で Ada に書き換え可能)	-	-	-

4.9.5 厳密な仕様記述の効果

(1) セキュリティ基準の達成

形式手法の導入により、情報技術セキュリティ評価基準(ITSEC) E6 の要請をみたすことができた。

(2) 信頼性の向上

出荷された製品は初年に 4 件の誤りが発見されたのみである。0.04 件／KLOC は新規開発としては非常に低い数字である（参考文献 6 より）。

(3) フロントローディングと手戻りの少なさ

要求分析および仕様記述という、いわゆる上流工程にかけられた工数と比較して、短時間の工数で実装が行われ、プロジェクト全体としての生産性は 28LOC／人日（数値は参考文献 6 より）という高い数値が達成された。誤り修正にかけられた工数の少なさ(全体の 6%)

は特筆に値する。また、各工程で混入された誤りは比較的早期に発見された。以下に図を引用する。

Figure 2. Life-cycle phases where defects were introduced and where they were detected and removed.

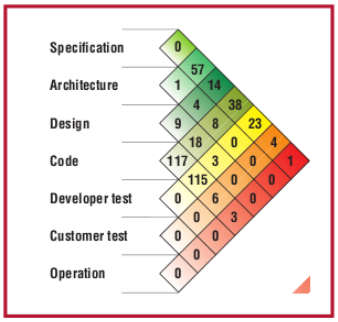


Table 1

Distribution of effort.

Activity	Effort (%)
Requirements	2
Specification and architecture	25
Code	14
Test	34
Fault fixing	6
Project management	10
Training	3
Design authority	3
Development- and target-environment	3

図 4-14: 各工程で実施された誤り修正

(Anthony Hall 他: Correctness by Construction: Developing a Commercial Secure System, IEEE Software, v.19 n.1, p.18-25, Jan 2002 より引用)

4.9.6 厳密な仕様記述を適用するための工夫と効果

(1) 機能とユーザインタフェースの分離

ユーザインタフェースのためのライブラリに既製品を用いながらもシステム全体としてのセキュリティ要求を満たすために、ユーザインタフェースからセキュリティに影響する機能を分離することでセキュリティ基準に適合する検証を行うことができた。

(2) システムの記述側面ごとの適材適所による形式手法の適用

システムの機能面を検証可能にするために、システムを様々な側面から記述し、それぞれの側面ごとに異なる記述言語や異なる手法を適用した。そして、各側面について、システムの一部を形式的に記述／検証するのではなく、例えばプロセス設計はシステム全域を CSP で記述し、システムの機能的仕様はその全域を Z 記法で記述した。そして既製品部品を利用せざるを得なかったユーザインタフェースについては、その側面での最大限の形式化を行った。この適材適所の方針に基づいて検証可能な側面はその全域を検証可能な言語で記述することで、システムのより多くの部分を検証可能にした。

4.9.7 顧客とのコミュニケーション

顧客とのコミュニケーションはまずは英語による要求獲得が行われ、自然言語やプロトタイプによるレビューが行われた。

4.9.8 開発者間のコミュニケーション

機能面に関しては Z 記法による仕様および CSP によるプロセス設計を参照して、Ada および SPARK のパッケージ仕様が作成され、そこから実装コードが作成された。英語による要求定義から実装コードまで完全な追跡性が確保された。

4.9.9 教育

このプロジェクトの教育的側面については公表されていない。

4.10 Tokeneer

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS10	Tokeneer	Altran Praxis Limited	Principal Engineer	2012/04/03	英国

4.10.1 プロジェクト特性

工期	2003
要求記述言語	自然言語
仕様記述言語	Z 記法
設計言語	INFORMED, SPARK(Ada サブセット+表明),
実装言語	SPARK(コア機能), Ada (周辺部, シミュレータ)
開発	英 Praxis 社
ドメイン	セキュリティ

4.10.2 プロジェクト概要

この事例は米国国家安全保障局(NSA)が情報セキュリティ国際標準(Common Criteria, CC) EAL5 に適合する高信頼システム開発による実装として、既存の生体認証情報システム(Tokeneer)の一部を形式手法を用いて再開発したものである。

開発主体となった英国 Praxis 社は形式手法による開発に実績があり、また、既にセキュリティクリティカルな高信頼性情報システムの開発経験があった。開発は新手法の導入ではなく、Praxis 社が発展させてきた数々の高信頼性ソフトウェア開発プロセスと開発技術が適用された。

開発は顧客側のドメイン専門家や社外技術者も形式手法の教育を受け、遠隔地に分散し開発者側の形式手法専門家の助けを受けながら、形式仕様を直接レビューした。開発は大別してカーネル部分と周辺部分に分けられ、カーネル部分は英国にて証明を含む構成的正当性(Correctness by Construction)が適用され、周辺部分は米国ニューメキシコ州にて、通常のソフトウェア工学的手法で開発された。本報告では、カーネル部分の開発を対象とする。

システム開発は 3 名の技術者によりパートタイムで合計 260 人日で遂行された（参考文献 7 より）。独立したシステム信頼性検査においても、また、出荷後においても、開発されたシステムに誤りは発見されていない。

プロジェクト終了後、成果物も含め全開発文書がインターネット上で一般に公開されている。本プロジェクトの目的はセキュリティ基準の達成だが、セキュリティ基準適合に限らず形式手法による信頼性の高いソフトウェア開発についての成功事例としての価値は非常に高いと考えられる。

4.10.3 開発プロセス概要

このプロジェクトは、高いセキュリティ基準から要請される構成的正当性(Correctness by Construction)と呼ばれる詳細化と証明を基盤とする形式手法を用いて実際にセキュリティクリティカルな高信頼性システムをスクラッチから開発するための参照となるべき開発として開始された。既に形式手法に熟達し実績のある専門家がおり、その形式手法による開発経験を生かして開発を進めつつ、顧客への形式手法の教育を提供し開発プロセスに受け入れて開発を遂行した。

開発プロセスの原則として、下記の項目を掲げた（具体的な記述は参考文献 7 を参照）。

- 正しい記述：必要な情報を混乱や曖昧や煩雑さなく記述する
- 飛躍なき進行：開発の各作業は小さなステップで進める。大きなステップは誤りを生み、また、誤り発見を困難にする。
- 単一の記述：未だ記述されていない情報やデザインデシジョンを、明確な目的とともに文書化する。同じ情報の繰り返しは不必要な作業を生み、本当のデザインデシジョンを不明にする。
- 確認してからの実行：各記述は可能な限り早急に検証する。それまでの記述と突き合わせてレビューすることで検証を容易にし、簡明な記述を得る。
- 論証の記述：デザインデシジョンの根拠とそれが正しいと考える理由を文書化する。理由付けは後の分析に役立ち、誤り発見につながる。
- 正しい道具：検証したい特性に最も適した方法で検証する。方法には公式レビュー、非公式ピアレビュー、証明、検査などが含まれる。
- 頭脳：作業全てについて頭を働かせる。顧客とのレビューでの注意深さ、矛盾を探すための知性、機能仕様を記述する時の正しい構成と明瞭さ、変更で

の思慮深さと文書化での注意深さを持つ。

開発プロセスをデータフローダイアグラムとして記述したものを下図に示す。

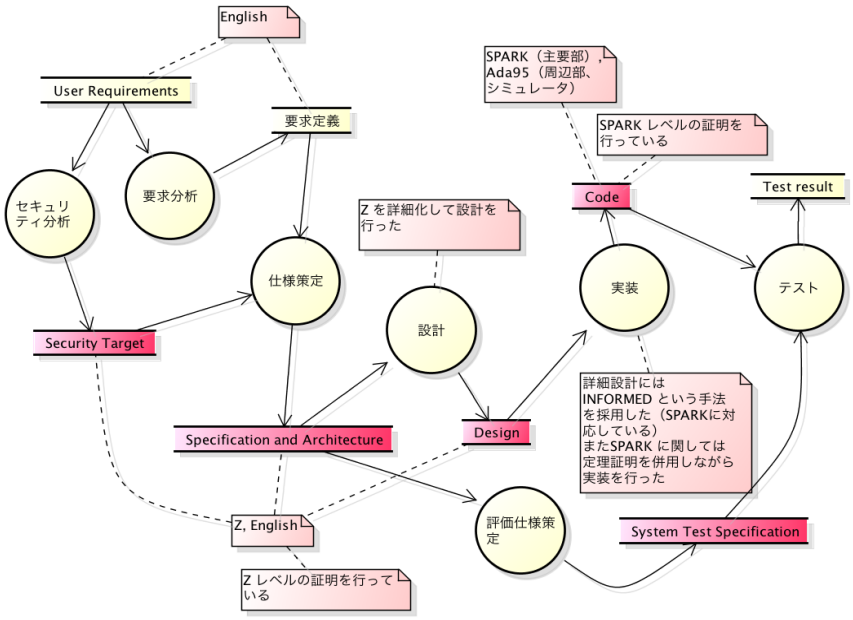


図 4-15: 開発プロセス概要[HS10]

4.10.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた主な言語と主なツールを以下に挙げる。

表 4-17: 使用された言語と主なツール

	Z 記法	SPARK
ファイル形式	LaTeX	ソース
構文/型検査	fuzz	SPARK toolset
静的検証(証明等)	-	SPARK toolset
動的検証(テスト等)	-	SPARK 実行時検査
自動生成	-	-

4.10.5 厳密な仕様記述の効果

(1) セキュリティ基準の達成

構成的正当性(Correctness by Construction)により情報セキュリティ国際標準(Common

Criteria, CC) EAL5 に適合する高信頼システム開発が可能であることを実証した。

(2) 遠隔地の顧客側レビュー

Z 記法で記述された仕様に対するレビューを米国メリーランド州にいる Z 記法の専門家と米国ニューメキシコ州にいるドメイン専門家の間で、電話によるレビューセッションを行い、生産的な結果を得ることができた。これは、Z 記法の基本的な読解スキルを身につけることで同一の仕様記述について解釈の曖昧性なしに複数人で討議することができたことを意味する。

4.10.6 厳密な仕様記述を適用するための工夫と効果

(1) 顧客数名と開発者 1 名のチームによるレビュー

ドメイン知識豊富な顧客側スタッフが Z 記法の専門家による補足説明を受けながら Z 記法による仕様記述を直接レビューすることができた。

(2) 構成的正当性による堅実な開発

前述の開発プロセス概要で示した通り、開発者は形式手法による高信頼性ソフトウェアの開発を数多く手がけており、そこで得られた教訓から開発プロセスで守るべき原則を掲げた。

4.10.7 顧客とのコミュニケーション

顧客から英語による要求を獲得した後、開発側が英語による要求を Z 記法による仕様記述として策定した。Z 記法による仕様記述は顧客のレビューを受けた。レビューでは Z 記法専門家が顧客のアシスタントを努めることで、1 週間程度の教育コースを受けただけの基礎的な読解を持つ顧客側ドメイン専門家もイディオム等を恐れずにレビューすることができた。

設計に関しては顧客による詳細なレビューは必要なかった。設計は Z 記法による仕様と同様の構成をしており、対応する項目間の比較は容易だった。

4.10.8 開発者間のコミュニケーション

各工程でそれぞれの詳細度での記述が行われたが、前工程での記述の構造を大きく変えることなく、必要な詳細の追加をひとつひとつ理由や根拠を示しながら記述を重ねていった。これによって、前工程と後工程の間の関係を把握することが容易になり、後工程が前工程の正当な詳細化であることを検証するのに役立った。

4.10.9 教育

顧客側のドメイン専門家が 1 週間の Z 記法の教育コースを受けた。その後、Z 記法の専門家がアシスタントとして Z 記法で記述された仕様の理解を手助けすることで、Z 記法の顧

客レビューを効果的に実施することができた。

4.11 オランダ花市場

ヒアリング セッション 番号	ヒアリング対象 プロジェクト名称	ヒアリング 対象組織名	ヒアリング 対象者の役割	ヒアリング日	ヒアリング 場所
HS11	オランダ花市場	C.H.E.S.S. Group	開発責任者	2012/04/10	オランダ

4.11.1 プロジェクト特性

工期	1998
要求記述言語	UML, 自然言語
仕様記述言語	VDM++
設計言語	VDM++
実装言語	C++
開発	蘭 C.H.E.S.S.社
ドメイン	商取引, レガシー拡張

4.11.2 プロジェクト概要

オランダ・スキポール空港近くに大規模な花市場があり、オークションが開催されている。オークション運営者はオークションでの競りによる商取引を行うシステムを数十年運用してきた。このシステムにネットワークを介して拡張を加え、遠隔地からもインターネットでオークションに参加できるようにするための拡張が VDM++により開発された。

開発主体となったのは、それまで既存のオークションシステムを維持管理してきた技術者チームである。技術者チームのメンバーは主たる業務として既存システムの維持管理や障害対応の修正を行ってきたことから、モデリング技法や形式手法に関する知識や経験はなかった。そこで形式手法 VDM の専門家を技術コンサルタントとして招き、上流工程である要求定義および VDM による仕様記述を委託し、下流工程はオークション主催者の技術者チームが、VDM 専門家の技術コンサルタントの指導を受けながら VDM を実作業を通して学習し、実装およびテストを行った。

このプロジェクトは、形式手法の専門家が上流を担い、形式手法を初めて導入する開発組織が下流を担当した事例であると同時に、形式手法によらない既存システムの拡張を形式手法で行った事例である。

4.11.3 開発プロセス概要

本プロジェクトは既存システムへの拡張だが、拡張部分は既存システムとネットワーク越しに接続されるものであり、また、いわゆる e-commerce としての要求は既存システムにはないものであったことから、拡張部分も新規開発と同様に要求定義から仕様記述を経て実装、テストが行われた。

要求はオークション主催者とのインタビューで獲得され、UML のユースケース、コラボレーション図、クラス図としてまとめられ、それに対応する形で VDM++仕様が記述された。UML と VDM++仕様は相互に繰り返し改良が行われていった。また、既存システムおよびそのインタフェース仕様はそれまで既存システムの維持管理をしていた技術者チームが持っており、その仕様をベースに拡張部分のインタフェースに関する VDM++仕様が定義された。

VDM++で記述された仕様は、オークション主催者の技術者チームにより C++で実装された。テストは既存システムの動作を模擬するシミュレータによって行われた。テストにおいて新規開発部分だけでなく、既存システム側のインタフェース仕様の更新漏れが発見された。

開発プロセスをデータフローダイアグラムとして記述したものを下図に示す。

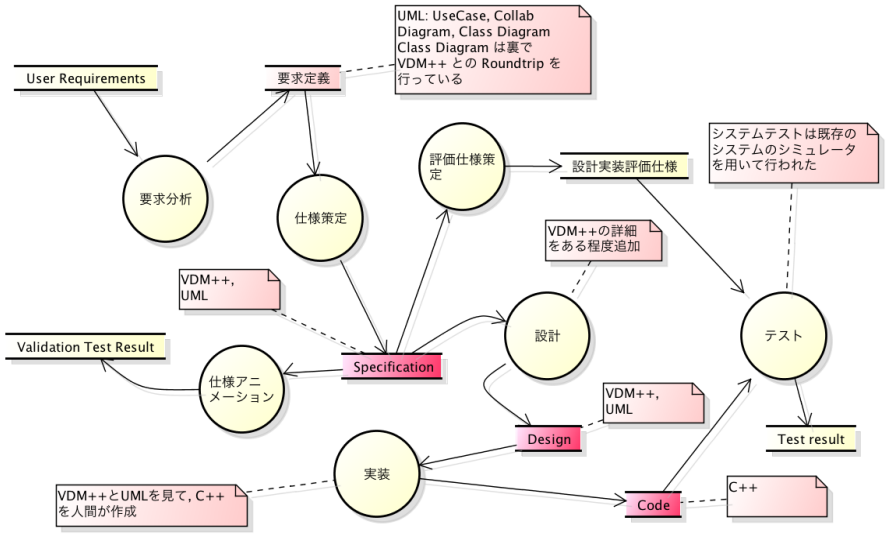


図 4-16: 開発プロセス概要[HS11]

4.11.4 記述と支援ツール

このプロジェクトで形式手法に関連して用いられた主な言語と主なツールを以下に挙げる。

表 4-18: 形式手法に関して利用された言語と主なツール

	UML	VDM++
ファイル形式	Rational Rose	LaTeX
構文/型検査	Rational Rose による作図	VDMTools
静的検証(証明等)	-	
動的検証(テスト等)	-	VDMTools
自動生成	-	VDMTools と Rational Rose の連携機能

4.11.5 厳密な仕様記述の効果

(1) 顧客のコミットメント

UML および VDM++で記述された仕様を繰り返し改善しレビューを行うことで、オークション運営側からの十分なコミットメントを受けることができた。

(2) 十分なテスト

VDM++による厳密な仕様とそこからくるテストケースにより、良質なテストが可能になった。その結果、既存システムのインタフェース仕様と実際の動作に違いがあることも発見された。

4.11.6 厳密な仕様記述を適用するための工夫と効果

(1) 形式手法のツールと他のツールの連携

UML ツールである Rational Rose と形式手法のツールである VDMTools を連携させることで、UML による要求獲得と VDM++による仕様記述を連携させることができ、繰り返し改善による仕様記述の質の向上と顧客レビューを通しての顧客のコミットメントにつながった。

4.11.7 顧客とのコミュニケーション

原始要求を提供するオークション主催者とは UML を使ったコミュニケーションが行われた。UML と VDM が相互に派生し合い、繰り返し改良を行った結果をオークション主催者がレビューすることで、オークション主催者側のコミットメントが得られた。

4.11.8 開発者間のコミュニケーション

下流工程を遂行したオークション主催者の技術者チームは、UML と VDM 両方の記述を参照し、C++による実装を行った。UML の各図を理解する上で、ツール連携を通して VDM++の記述を参照できることが非常に役に立った。

4.11.9 教育

オークション主催者の技術者チームは VDM++についてクラスルーム形式による教育を受けずに、実作業を通して VDM 専門家である技術コンサルタントから VDM++を学んだ。

5. 総合分析結果

本章では、事例を横断的に分析し、考察した結果を論じる。5.1 では、ヒアリングの結果から明らかとなった形式手法の適用箇所について、成功している事例に共通する点、あるいは類似した点という観点から説明を行う。5.2 では、同じくヒアリング結果から明らかとなった形式記述と自然言語（日本語）をはじめとするそれ以外の記述との併存について、共通する点あるいは類似した点という観点から説明を行う。形式手法の導入に関しては、1990 年にアンソニーホールが「形式手法の 7 つの神話」として、典型的な形式手法に関わる誤解を 7 つ列挙している。5.3 では、ヒアリング結果を踏まえた上で、これらの誤解に対する実証的な反証を列挙する。

5.1 形式手法の適用箇所についての成功事例全体に通じる共通点と類似点

5.1.1 サマリ

４章で詳述したように、事例調査をした各プロジェクトでの、形式手法を始めとする厳密な記述の適用範囲は要求、仕様、設計、実装、検証（そしてそれに続く保守）を含む全工程にまたがっていることがヒアリングの結果明らかとなった（表 5-1 参照）。

厳密な記述はその前後の開発フェーズにおける他の記述や成果物との間に明快な対応関係を保っているものが多い（相手が自然言語による一塊のドキュメントの場合は対応が難しい場合もある）（図 5-1 参照）ことがわかった。

また、プロジェクトごとに記述する対象は異なっているものの、いずれの場合にも、ある核となる記述対象を定めて、プロジェクト内での記述を全てその方式に統一し、かつ保守対象として扱っている（図 5-4 参照）こともヒアリング結果より明らかとなった。

粒度が揃っていて網羅性のある記述を導入することにより、プロジェクトの状況を定量的に解析することが可能になり、変更に対するコスト（工数、工期）などの情報も蓄積しやすくなる（図 5-5 参照）と考えられる。

5.1.2 適用箇所

今回事例調査をした各実プロジェクトでの、厳密な記述適用（ならびに応用）箇所を以下に表としてまとめた。ツールを使って構文を整えることができるという意味で UML なども挙げてある。

また、たとえ形式的な記述手法を使っている場合でも、全ての記述には必ず自然言語による注釈が伴っている（5.2 節参照）。

表 5-1: 各事例における厳密な記述適用箇所

事例	要求	仕様	設計	実装	検証
FeliCa(1)	UML	VDM++ 記述	FSP/LTSA, UML	C++	テスト設計、テストオラクル生成、仕様アニメーション
FeliCa(2)	UML	VDM++ 記述	FSP/LTSA, UML	C++	テスト設計（デシジョンテーブル）、テストオラクル生成、仕様アニメーション
BPS1000	UML	VDM-SL	UML	C++	テスト設計への利用
SHOLIS	-	Z	Z, SPARK	SPARK	Z proof, SPARK proof
iFACTS	-	状態表, Z,Mathematica	SPARK	SPARK	Z proof, SPARK proof, Mathematica
TradeOne	UseCase	VDM++	Gofo	C++	仕様アニメーション、テストオラクル生成
オランダ軍メッセージ分析	IRS	VDM-SL	VDM-SL	C++（大部分を自動生成）	STD（VDM-SL で記述されたテストケース）、仕様アニメーション、テストオラクル生成
CAVA	-	VDM-SL	VDM-SL	-	仕様アニメーション、テストオラクル生成
MULTOSCA	Z（セキュリティ）	Z	Z, CSP	SPARK, Ada95, C++, C, SQL	CSP モデルチェッカー, Z proof, SPARK proof
Tokeneer	Z（セキュリティ）	Z	Z,INFORMED, SPARK	SPARK, Ada95	Z proof, SPARK proof
オランダ花市場	UML	VDM++（UML とラウンドトリップ）	VDM++, UML	C++	仕様アニメーション、テストオラクル生成

成功したプロジェクトはいずれも開発工程内の単一のフェーズに適用しているのではなく、必ずその前後のフェーズとの密な追跡性の確保が行われている。

例えば個別事例のなかの TradeOne の事例を見てみよう。ここでは形式仕様記述 VDM++ と C++を用いた設計実装の連携の例を見る事ができる。

以下に個別分析で掲げた TradeOne のデータフローダイアグラム をもう一度示す。

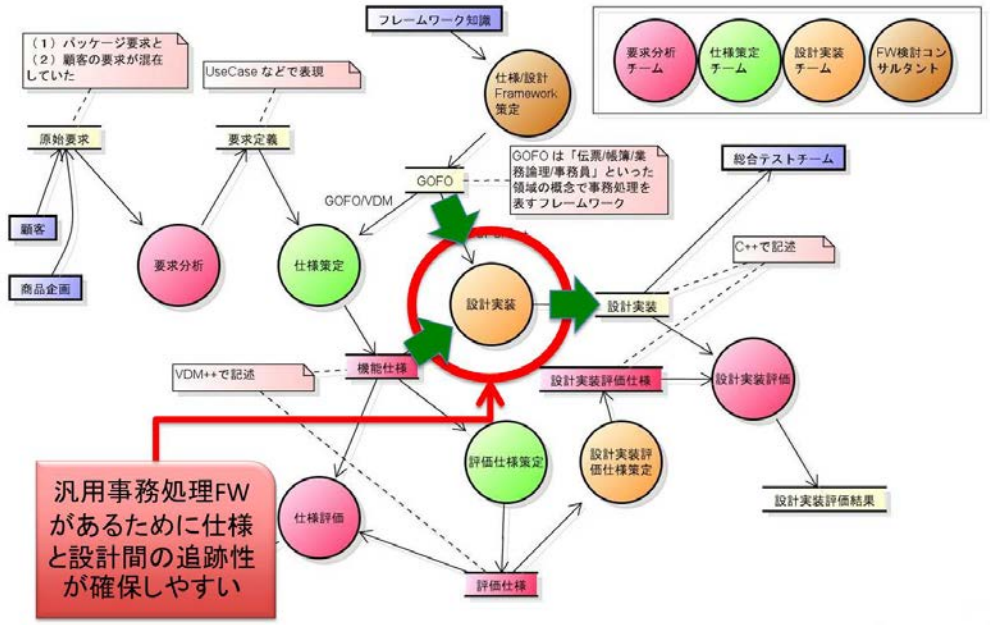


図 5-1: プロセス全体像（再掲）

図中に注釈を加えてあるが、この事例では仕様から設計への展開を円滑にするために、汎用事務処理フレームワーク Gofo を利用している。このフレームワークは事務処理特有の概念（入力伝票、出力伝票、業務論理、帳簿、事務員、etc…）を構成要素として包含していて、通常の事務処理（業務論理の適用）と事務フロー（伝票の交換）を同時に表現できるような構成になっている。

以下に示すのが Gofo を構成するユニットの概念図である。この図における「事務員」は計算機の処理能力を抽象化していて、機能仕様そのものの定義時にはあまり関係してこない（非機能仕様や設計を考えるときに登場する）。業務論理の仕様そのものは以下の 3 要素で定義される。

(1) 事前条件：入力伝票、業務論理適用前の帳簿

(2) 事後条件：業務論理適用後の帳簿、出力伝票

(3) 不変条件：帳簿の内容に関する制約

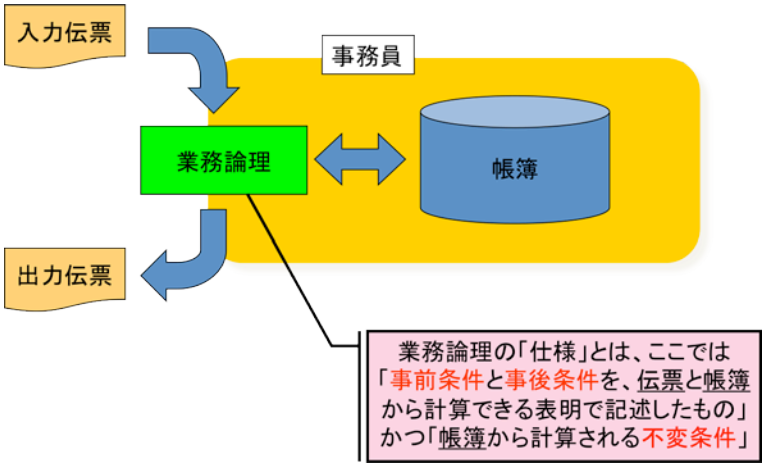


図 5-2: Gofo を構成するユニット概念図

図 5-2 からもわかるように、一般的な事務フローも、このような形式手法で記述できることがわかる。

このような設計実装を可能にする抽象クラスが用意されているため、仕様を設計へ反映する際には VDM++等で記述された機能仕様を対応付けて行く事になる。

このように、事務処理における機能仕様=業務論理を素直に設計して配置することができるようになっているため、機能仕様に対する設計実装を参照したい場合にも容易であり、機能仕様の事前事後条件が適切に反映されているかの設計レビューも行いやすい形になっている。

別の例を取り上げてみよう。MULTOS の事例である。以下に個別分析で示したデータフローダイアグラムを再掲する。

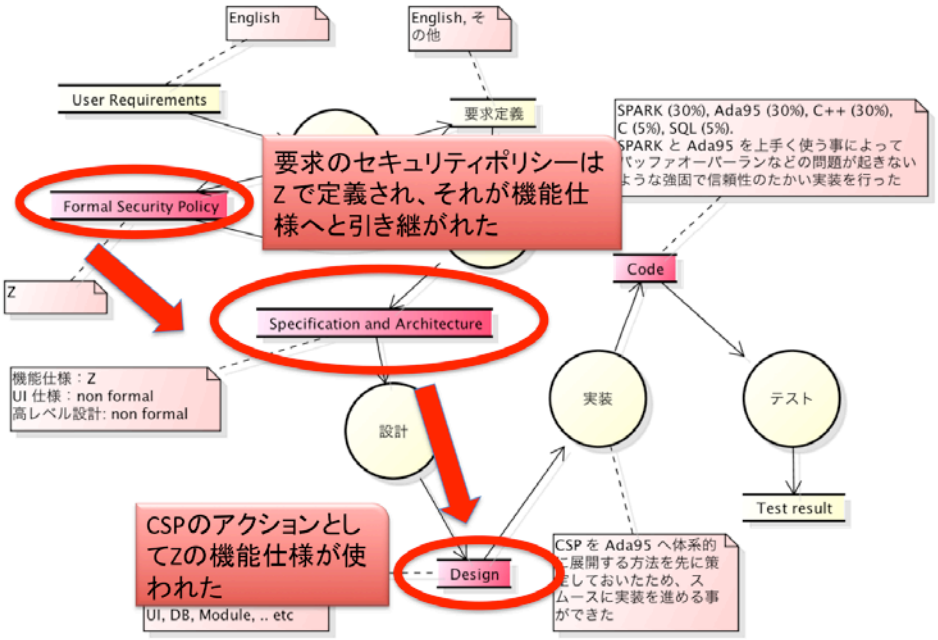


図 5-3: データフローダイアグラム（再掲）

注釈で示したように、要求段階での Z 記法で記述されたセキュリティポリシーが機能仕様に受け継がれ、証明の対象となった。また機能仕様として定義された Z 記法による仕様は設計時に CSP のアクションの仕様として用いられた。

CSP を Ada95 へ体系的に展開する方法を事前に策定していたために、実装も円滑に進んでいる。

このように形式的な記述は、一度きちんと記述してしまえば、それに続く様々な開発フェーズで利用していくことが可能である。多くの開発現場では全く同じ内容の情報が、単純なワードやエクセルのドキュメントであるが故に、複製されつつ微妙に版が変わりながら利用されている。

厳密な記述（特に形式的記述）はその界面と意味がはっきりしているため、他の開発ドキュメント中に仮に埋め込まれてしまったとしても、劣化コピーを作成することなく再利用できる可能性を持っている。もちろんこうした使い方は、開発フェーズ間の成果物の関連をきちんと考慮しておかなければ手に入らない。

なお、今回は調査対象になっていないが、形式手法適用技術の、もう 1 つの大きな流れとして B ならびに Event-B と呼ばれる言語を中心とした開発手法が存在する。この開発手法では、まず要求を満たす抽象機械を設計したあと、正しさを確保しながら詳細化を繰り返し、最終的に実装へと導く。この場合には自動的に網羅性と追跡性も確保されていることになる。

5.1.3 記述対象に関する考察

前節でも述べたように調査対象の事例はプロジェクトの開発ドキュメント全てを形式的に記述しているわけではない。前節では開発フェーズ毎の適用の様子を分析したが、本節では記述の対象がどのようなものであったのか、そしてその記述の際の留意点（図 5-4 参照）は何であったのか、更には形式仕様が情報のハブという役割を果たす、という点について考察する。

以下に示したのは、各調査対象事例で主に「厳密な記述の対象」となったものの表である。

表 5–2: 各事例における厳密な記述の対象

事例	記述対象
FeliCa(1)	FeliCa チップファームウェア外部仕様(API の仕様)
FeliCa(2)	FeliCa チップファームウェア外部仕様(API の仕様)、API 仕様から派生したデシジョンテーブル（テストケース）
BPS1000	銀行券鑑別仕様（センサデータを抽象化した）
SHOLIS	機能仕様、詳細設計(SPARK)
iFACTS	状態遷移、機能仕様、詳細設計(SPARK)
TradeOne	業務論理（入力伝票、出力伝票、帳簿の状態）仕様
オランダ軍メッセージ分析	要求仕様：IRS（入出力の要求。形式の定まった自然言語） 機能仕様：IRS を満たす入出力の関係 評価仕様：IRS を満たす評価仕様
CAVA	AST の不変条件 AST の変換機能仕様
MULTOS CA	セキュリティ要件 機能仕様 並列性設計(CSP)
Tokeneer	セキュリティ要件 機能仕様
オランダ花市場	システムの振舞要件（UML クラス図、コラボレーション図を要求定義に利用。この内容を Rose を使って VDM++とラウンドトリップ） ソフトウェアモジュール機能仕様（VDM++）

調査対象になった事例の記述対象を分析すると、いずれの記述もそのプロジェクト内の全ての範囲に渡って記述を行っていることがわかる。これは「要求から実装まで全てのレベルを詳細かつ厳密に記述している」という意味ではなく、「ある意味のレイヤーを定め、そのレイヤーに関してはプロジェクト全体に渡って記述している」という意味である。

例えば FeliCa の場合、ファームウェアの外部仕様を記述するという決定を下して、全ての外部仕様を VDM++で記述している。ここで全ての仕様を記述しているため、仕様アニ

メーションを用いた仕様の Validation ならびに Verification も、この記述を起点として全て始めることが可能になったのである。

以下に FeliCa (1)の個別分析で示した図を再掲する。

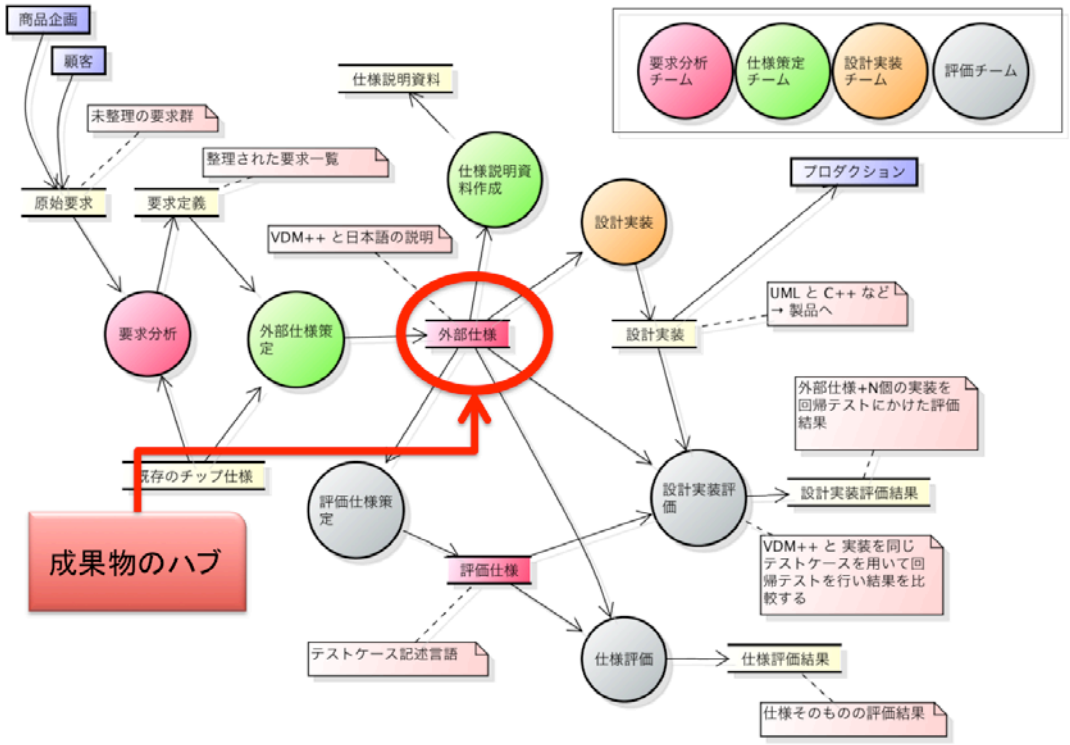


図 5-4: プロセス全体像（再掲）

この図中に注釈で示したが、FeliCa プロジェクトの「外部仕様」は、全開発情報を辿る為の「ハブ」としての役割を果たしている。「要求定義」、「設計実装」、「仕様説明資料」、「評価仕様」などが全て「外部仕様」を中心に関連付けられているのである。評価側でもし仕様上の不具合が見つかれば、その結果は直接設計実装には戻されず、必ず「外部仕様」に反映されて、改めて「仕様評価」が施された後、「設計実装」へとつなげられる。

こうして、常に「外部仕様」を（仕様アニメーションを介して）評価済の状態にしておくことにより、開発者は常に何が最新で検証済の仕様かを知る事ができるようになるのである。

「厳密な記述」にこのような「情報のハブ」という役割を果たさせるためには、当然プロジェクト全域にわたる記述が必要となる。一部の情報が別の形式で書かれていたり、粒度が違っていたりした場合こうしたハブとしての利便性が損なわれることになるだろう。

もう１つの例を見てみよう。

オランダ軍のメッセージ分析の場合も IRS という定型的な入出力要求文書の形式を定め（これは自然言語ではあるが）全ての要求を書き出している。これを用いて、機能仕様と評価仕様をそれぞれ網羅的に記述したために、仕様アニメーションならびに、実装テストを通して全ての仕様評価が行えたのである。

以下に個別分析の際に示したデータフローダイアグラムを再び示す。

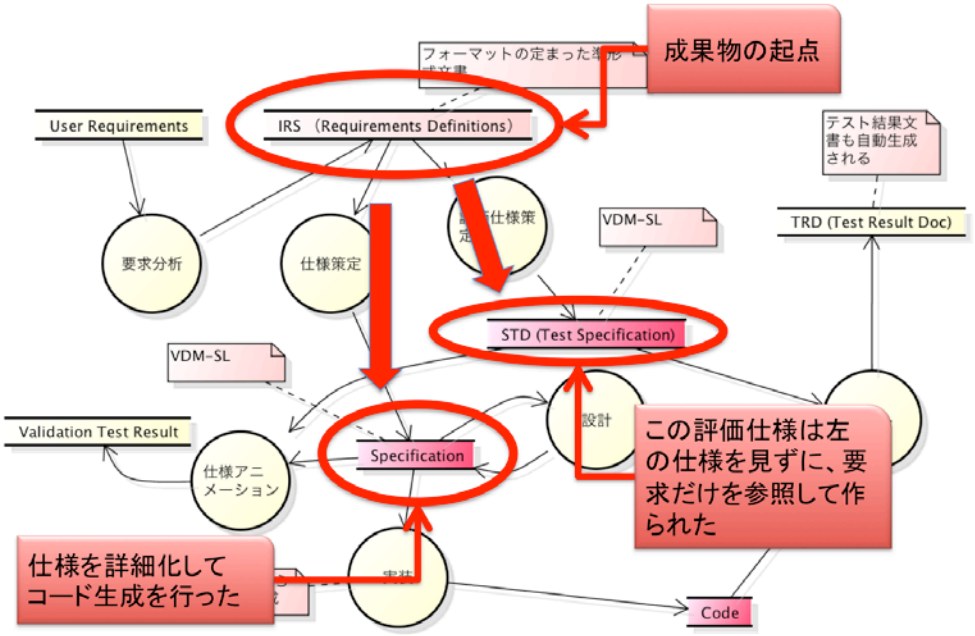


図 5-5: データフローダイアグラム（再掲）

図中に注釈で示したが、このプロジェクトでは FeliCa プロジェクトで「外部仕様」が果たしていたような情報の要の役割を、IRS というこのプロジェクトのために策定した形式の要求記述（記述の枠組みは決まっているが、内容は自然言語）が果たしている。

個別分析でも述べたように VDM-SL で書かれた機能仕様(Specificatin)と、やはり VDM-SL で書かれたテスト仕様(STD)が、お互いを参照することなく記述されたことが特徴である。この特徴により、妥当性検証のための仕様を記述する人間が、機能仕様として定義されたものに影響されずに、評価仕様をまとめることができたのである。

この例では IRS を「成果物の起点」として扱っていて、当然このレベルで記述されるべき内容は全て網羅されている。こうした網羅性と追跡性はプロジェクトの進捗状況を知る上でも重要である。

前節でもとり挙げたように、ある記述は他の記述との関連付けが大切である。記述のレイヤーは抽象化のまとまりを表しているが、こうした同じ抽象度を表すべきレイヤーの記述方法がバラバラでは、保守を行う際に大変な困難を引き起こす事が予想される。

新規開発途中でもステークホルダ側からの要求の変更は継続的に発生するので、既に保守の問題は始まっているとも言える。

ここでは個別事例のうち FeliCa とオランダ軍のメッセージ分析の例を取り上げたが、厳密な記述(IRS)を使う場合でも形式仕様記述(VDM++)を使う場合でも、その記述による粒度を揃えた網羅性が確保できるならば、開発情報の「ハブ」もしくは「起点」として利用できることが可能になることがわかった。

そこから更に粒度の揃った網羅的記述へと繋いで行く事ができるならば、常にプロジェクトの状況をそうした記述から得られるメトリクスをもとに計算できることになる。

このようにして粒度が揃っていて網羅性のある記述を導入することにより、プロジェクト

の状況を定量的に解析することが可能になり、変更に対するコスト（工数、工期）などの情報も蓄積しやすくなる。

5.2 形式記述とそれ以外の記述の共存についての成功事例全体に通じる共通点・類似点

5.2.1 サマリ

4 章で詳述したように、ヒアリングの結果より、事例調査をしたいずれのプロジェクトでも、形式仕様記述だけではなく、それを補う記述（自然言語による普通の文章、図、表など）を併用していることが明らかとなった。事例横断的に分析を行うにあたっては、こうした利用形態を分類するために、まずは大きく分けて 2 通りの状況を区別することにした。

開発チーム内における形式記述とそれ以外の記述の併用

開発チームと外部の間における形式記述とそれ以外の記述の併用

3 章および 4 章で述べたように、両方の記述を併用することで、 (1) そもそも形式手法では表現できない情報を記述することができる、(2) 形式記述の可読性を上げることができる、といった点で有効であることが、ヒアリング結果より知見として得られた。

以下では、それぞれの状況下でどのようなことが行われていたかについての観点から分析を行う。

5.2.2 開発チーム内での併用

いずれの開発プロジェクも、既に個別に示したように形式記述でてを書き切ったわけではない。そもそも形式記述で書きにくい、概要（システムの目的や、全体の構造を概説したもの）、非機能要求などは自然言語（図や表なども含む）で書かれている。

こうした記述もその内部に仕様に関わる用語を包含しているので、理想を言えば形式記述に照らし合わせて管理すべきなのであるが、一度書かれるとあまり変更されることもなく、保守上も大きな問題になりにくい（その記述だけをみても保守はできない）ために、いずれの事例でも積極的に形式仕様との整合性を図ろうとしてはいなかった。

よって、ここで取り上げたいのは、形式仕様と直接対応付けられる記述の扱いである。以下の表に示すのは各事例における開発チーム内での形式記述とそれに伴う形式的ではない記述の扱いである（なお設計以降の記述は省いている）。

表 5-3: 各事例における形式記述と形式的ではない記述の扱い

事例	形式記述	形式的ではない記述
FeliCa(1)	UML VDM++ FSP	日本語による UML への注釈 日本語による VDM++式への注釈
FeliCa(2)	UML 日本語識別子を用いた VDM++ FSP	日本語による UML への注釈 日本語による VDM++式への注釈（隠蔽関数利用） デシジョンテーブル
BPS1000	UML VDM-SL	英語による UML への注釈。 英語と VDM-SL を使った文芸的プログラミング
SHOLIS	Z 仕様記述	英語と Z を使った文芸的プログラミング
iFACTS	Z 仕様記述 状態表 Mathemaica	英語と Z を使った文芸的プログラミング
TradeOne	UseCase 日本語識別子を用いた VDM++	日本語による VDM++への注釈
オランダ軍メッセージ分析	IRS VDM-SL 機能仕様 VDM-SL 評価仕様	英語と VDM-SL を用いた文芸的プログラミング
CAVA	VDM-SL	英語と VDM-SL を用いた文芸的プログラミング
MULTOS CA	Z 要求記述 Z 仕様記述	英語と Z を使った文芸的プログラミング
Tokeneer	Z 要求記述 Z 仕様記述	英語と Z を使った文芸的プログラミング
オランダ花市場	UML VDM++	VDM++式への英語による注釈

表 5-3 の一番右のカラムに現れるものを以下に説明する。

【形式記述式への自然言語による注釈】

形式仕様記述言語は厳密な定義を許すが、書き方によっては読みにくくなってしまう。人間がこうした仕様を読み取って設計等を行う場合、読みにくさは誤りの基となる。

このため簡便な方式としては、読みにくそうな形式仕様記述の式に対して自然言語による注釈を併記するという方法が考えられる。この方式をとることにより、仕様レビューが円滑に進む効果が期待されるからである。

実際 FeliCa(1)の事例では、単に VDM++の式だけを書いていた場合に比べて読みやすさが向上し、レビューの効率も上がった。これは日本語として書かれている意味をまず頭に入れながら、VDM++の式の意味をレビューし双方に間違いがないことをチェックできるためダブルチェックとしての役割を果たすことにもなった。

しかし、いわば同じ定義が二重に書かれていることから、間違いも発生した。仕様を読む開発者が片方（日本語だけ）の説明を読み、それが実は VDM++の定義とは食い違っていたことに気が付かず、結果的に設計ミスとなった事例が発生したのである。

FeliCa(2)ではこの反省に立ち、日本語の注釈を減らし、VDM++の式そのものを読みやすくするための工夫が行われた。

- 日本語識別子の採用
- 隠蔽関数の利用による問題領域の用語の利用
- ラベル付き条件式による記述形式

といったものである。個別分析の所でも示したが改善の結果以下のようなイメージとなった。

ラベル付き条件を用いない場合

```
-- パケットの長さがサービス数まで存在する十分なパケットであること
packet_len_of_service_num >= 9 and
-- パケットにおいて指定されたサービス数が規定の範囲内であること
1 <= service_num and service_num <= 32
```

ラベル付き条件を用いた場合

```
cases false :
  (パケットサイズ十分？("サービス数")),
  (サービス数範囲内？(cmd_pkt, cmnd_pkt_st))) -> <ERROR_NO_RESPONSE>,
others ->
  <SUCCESS>
end;
```

```
private サービス数範囲内？ : PACKET_DATA * PACKET_STRUCTURE -> bool
 サービス数範囲内？(cmnd_pkt, cmnd_pkt_st) ==
  let
    CmndPkt = パケット要素取得(cmnd_pkt, cmnd_pkt_st),
    num = hd CmndPkt("サービス数")
  in
    1 <= num and num <= 32;
```

図 5-6: ラベル付き条件を用いる場合と用いない場合(参考文献 20 を元に作成)(再掲)

図中「ラベル付き条件を用いない場合」が旧来の記述方法であり、VDM++に式に対して日本語の注釈が付けられている。

これに対して改善版の方では日本語の注釈がなくても、問題領域の知識を持つ読み手なら内容を理解しやすいような形になっているのである。

【自然言語と形式仕様記述言語を用いた文芸的プログラミング】

最近は日本ではあまり耳にしないが「文芸的プログラミング」というのは D. Knuth 博士が提唱した、「ドキュメントとプログラムコードの交ぜ書きスタイル」のことである。

この交ぜ書きされたソースは web（ウェブ）と呼ばれ、1 つのファイルとして管理されるが、ツールを用いることによって、コンパイル可能なソースコード（当初はパスカル）と、清書可能なドキュメントの部分に分離させることが可能になる。

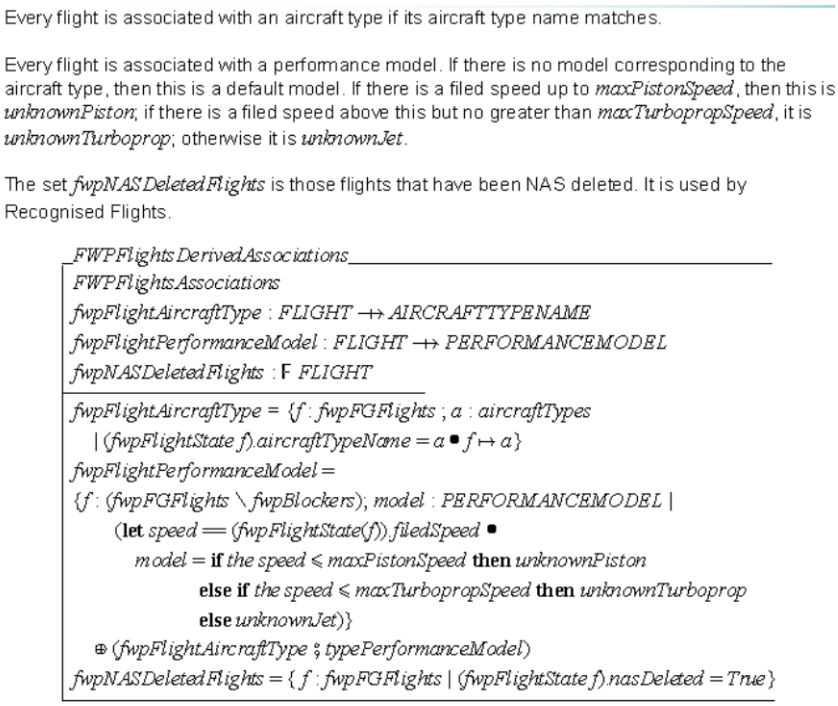
清書可能なドキュメントの部分には、プログラムコードそのものも出力され、非常に美しい文書をタイプセット品質で得ることができる。現在アカデミックな世界で広く使われている TeX 清書システムはこうした文芸的プログラミングスタイルを支援するツールプロジェクトの副産物である。

上記の表の中に挙げた「文芸的プログラミング」はこの流れに沿ったものである。かつては TeX 形式で行われていた「文芸的プログラミング」であるが、iFACTS などの新しいプロジェクトでは、MS Word を用いて、ドキュメントとコードの共存を図っている。

文芸的プログラミングの場合、ある程度のまとまりのある仕様に対して、自然言語による説明を付加するスタイルが一般的である。

以下に例示するのは iFACTS プロジェクトの文芸的プログラミングスタイルを用いた機能

仕様の例である。



Every flight is associated with an aircraft type if its aircraft type name matches.

Every flight is associated with a performance model. If there is no model corresponding to the aircraft type, then this is a default model. If there is a filed speed up to *maxPistonSpeed*, then this is *unknownPiston*; if there is a filed speed above this but no greater than *maxTurbopropSpeed*, it is *unknownTurboprop*; otherwise it is *unknownJet*.

The set *fwpNASDeletedFlights* is those flights that have been NAS deleted. It is used by Recognised Flights.

FWPFlightsDerivedAssociations

$$
\begin{array}{l}
FWPFlightsAssociations \\
fwpFlightAircraftType : FLIGHT \pfun AIRCRAFTTYPENAME \\
fwpFlightPerformanceModel : FLIGHT \pfun PERFORMANCEMODEL \\
fwpNASDeletedFlights : \mathbb{F}\ FLIGHT \\
\hline
fwpFlightAircraftType = \{f : fwpFGFlights ; a : aircraftTypes \\
\quad | (fwpFlightState\ f).aircraftTypeName = a \bullet f \mapsto a\} \\
fwpFlightPerformanceModel = \\
\{f : (fwpFGFlights \setminus fwpBlockers); model : PERFORMANCEMODEL \mid \\
\quad (\textbf{let}\ speed == (fwpFlightState(f)).filedSpeed \bullet \\
\qquad model = \textbf{if}\ the\ speed \leqslant maxPistonSpeed\ \textbf{then}\ unknownPiston \\
\qquad\qquad \textbf{else if}\ the\ speed \leqslant maxTurbopropSpeed\ \textbf{then}\ unknownTurboprop \\
\qquad\qquad \textbf{else}\ unknownJet)\} \\
\quad \oplus (fwpFlightAircraftType \fatsemi typePerformanceModel) \\
fwpNASDeletedFlights = \{f : fwpFGFlights \mid (fwpFlightState\ f).nasDeleted = True\}
\end{array}
$$

図 5-7: iFACTS プロジェクトの文芸的プログラミングスタイルを用いた機能仕様の例
（参考文献 8 より引用）

ここでは上半分に仕様の説明が記述され、下半分にその形式仕様記述が記載されている。

iFACTS プロジェクトは、評価仕様の記述にも「文芸的プログラミングスタイル」を用いている。以下に示すのがその例である。

2.2.1.18 TPDeviationRequests

Summary

Requests the required deviation trajectories.

This is a non-conditional schema.

Partitions

There are two equivalence classes:

1 Flight is not radar supported, so no information.

2 Flight is radar supported.

The output condition in the first equivalence class is that there is no request. This can also occur when there are no deviation trajectories, so that input condition should be tested as well.

It is stated within the FPM process specification that the number of deviation requests will be either none, one or two ([4] section 13.2.13.2). We should test for each of these conditions separately (since 0 and 2 are boundary conditions).

Test Conditions

TPDeviationRequests	1	2	3	4
fpmData!? = nil	●	○	○	○
(the fpmData!?).fpmDeviationTrajectories = ∅		●	○	○
deviationReqs = ∅	●	●	○	○
#deviationReqs = 1	○	○	●	○
#deviationReqs = 2	○	○	○	●

図 5-8: iFACTS プロジェクトの文芸的プログラミングスタイルの例(参考文献 8 より引用)

この例では、上半分にテストの説明が記述され、下半分にその厳密な定義が記述されている。厳密な記述の左の方の式を見てみると、これは Z 記法の記述そのものである。

【デシジョンテーブル】

これは FeliCa(2)での事例だが、ラベル付き条件式の記述形式から、テストケースを検討しやすいようにデシジョンテーブルという記述が工夫された。これは形式仕様記述から派生させた文書に具体的なテストケースを埋め込んで使うための形式である。

以下にその例を示す。

条件	同値分割	境界値	TC1	TC2	TC3	TC4
実行可能モード？	TRUE	0..2	Y		Y	Y
	FALSE	3		Y		
パケットサイズ十分？	TRUE	10	Y	Y		Y
	FALSE	9			Y	
サービス数範囲内？	TRUE	1..32	Y	Y	N	
	FALSE	0, 33			N	Y
結果	正常応答		Y			
	応答なしエラー			Y		
	応答ありエラー				Y	Y

テストケースの網羅性を視覚的に確認することができる。レビューが容易になる。

図 5-9: デシジョンテーブルの例(参考文献 20 を元に作成)(再掲)

【その他の形式】

上記に紹介した形式以外にも、HTML を生成してウェブ上でソースコードのハイパーテキストを実現したり、用語の一覧を作ったり、データベースのスキーマの基礎を生成したり、仕様の情報を解析してデータベースに格納しレポートライタ機能を用いてドキュメントを生成したりと、簡単なスクリプトプログラミングによって、形式仕様記述以外の文書は気軽に作成されている。

MS Word の場合には通常のテキスト処理は行えないものの、マクロを駆使して必要となる情報を抽出するという試みは行われていた。

5.2.3 開発チームと外部とのコミュニケーションにおける併用

外部とのコミュニケーションを考えた場合、更に大きく 2 つの状況が区別された。それは

- 開発チーム外部（要求提示者、評価者など）に形式仕様記述そのものを見せてコミュニケーションを図る状況
- 開発チーム外部（要求提示者、評価者など）には形式仕様記述そのものは見せず、代替の記述を用いてコミュニケーションを図る状況

ということである。

傾向として安全性に関して非常にクリティカルなシステムほど(SHOLIS, MULTOS CA, Tokeneer, iFACTS など)要求提示側が自ら形式仕様そのものを読む（場合によっては書く）姿勢になっていた。

Dependable Software Forum（DSF)[10]が取り上げるような高度にミッションクリティカルなシステムの場合にも、要求提示側が形式仕様記述そのものを読み（書き）する効果は高

[10] ソフトウェアの信頼性と安全性向上のために、6 社と 1 機関が共同で研究開発活動を行う団体（http://www.nttdata.co.jp/dsf/index.html）

いと思われるが、これはこの先の実証が待たれる範囲である。

外部が形式仕様を読まない場合には、形式仕様記述を使わない分、厳密さを何かの記述で補填しなければならない。以下に示すのは開発チーム外部と形式仕様記述以外の記述でコミュニケーションしたプロジェクトが行った工夫である。

表 5–4: 外部とのコミュニケーションのための工夫

事例	行った工夫
FeliCa(1)	VDM++記述を見ながら人間が説明資料を作成 VDM++の記述からマークアップした場所や日本語注釈を抽出してマニュアルの一部を生成
FeliCa(2)	FeliCa(1) に加えて、デシジョンテーブルを使ったテストケースの検討
TradeOne	VDM++記述を見ながら人間が説明資料を作成
オランダ軍メッセージ分析	IRS を用いた要求記述。要求記述の単位と範囲を明確にしたため、機能仕様、評価仕様と対応つけやすくなった。
オランダ花市場	UML ツール（Rose）を用いて VDM++コードをラウンドトリップし、UML クラス図を生成。他にシナリオ検討時にコラボレーションダイアグラムを利用

興味深かったのが、インタビュー対象の 1 人である Peter Larsen 博士の以下の言葉であった。

「一部の人は形式仕様を決して見たいとは思わない。その場合私は裏で形式仕様モデルを作りながら、表では自然言語や UML のダイヤグラムを使ってコミュニケーションを行う。そして大抵の場合、素早く問題領域上の問題を指摘して、顧客にとても驚かれる場合が多いのだ。」

（実際には形式手法が使われていたのだが）普通の開発手法しか使っていないと思っていた顧客は、なぜあまりよく知らない問題領域の問題を易々と相手が指摘できたのか不思議だという話である。対象システムの規模が大きくなればなるほど内部の矛盾を見つけるのは指数関数的に難しくなっていくが、形式手法を上手く使うことにより他者に抜きん出た振舞をなすことができる可能性を、これらのヒアリング結果は示唆していると言える。

5.3 形式手法に関するしばしば言及される誤解への反証

形式手法の普及を阻害する、形式手法に対する典型的な誤解の一覧が、Anthony Hall による「形式手法の 7 つの神話」(Seven Myths of Formal Methods)として 1990 年に発表されている。それから 20 年以上が経過した今でも、「形式手法の 7 つの神話」は多くの開発現場を覆ったままである。以下「形式手法の 7 つの神話」それぞれについて本調査での分析結果に基づき実証的な反証を列挙する。

神話(誤解)1. 形式手法はソフトウェアが完全である事を保証できる。

本調査の分析対象となった事例のうち、SHOLIS, iFACTS, MULTOS CA, Tokeneer の 4 事例は高信頼性を目的に形式手法を導入し、非常に信頼性の高いソフトウェアを出荷した。しかしながら、これは「完全であることを保証できる」ことを意味しない。これら高信頼性ソフトェアの開発についてヒアリング対象の 1 人は、「我々は完全なソフトウェアという言葉で語った事はない」と語り、また、別の 1 人は、「我々の開発は、高確信性という言葉で表わすのが適切である」「我々は出荷する実行コードのあらゆる断片について、それがなぜそのようになっているのか、追跡可能にする」と語った。すなわち、完全さを最初から求めるのではなく、欠陥を減らすための堅実で地道で細やかなステップを刻みながら、それらステップを追跡し、理由付け、その理由が正しいのかどうか検証可能にすることが形式手法による高信頼性の核心であるというのが、本調査のヒアリングでの回答である。

また、形式手法を導入する目的は高信頼性だけではない。工期短縮を目的にした事例、開発対象に関する開発者間の認識を合わせることを目的にした事例、複雑で例外の多い要求から簡明で見通しの効く仕様記述を得ることを目的にした事例などがある。前述の高信頼性目的の事例もあわせて、成功した形式手法の事例には、真っ先に完全な仕様記述を確定して完全な実装を得たというものはない。成功事例に共通していることは、むしろ仕様を改善していく姿勢であった。すなわち、(1) 仕様を曖昧なく記述し、(2) 形式仕様から多くの記述を派生させ、(3) 多くの工程で形式仕様を参照し、(4) 他の工程からのフィードバックにより仕様記述の品質を高める、という作業をひとつひとつ着実に実行した、ということである。

神話(誤解)2. 形式手法はプログラムの検証に関するものである。

確かに形式手法は証明やテスト等の、プログラムの検証に様々な効果がある。しかし、プログラムの検証以外にも様々な効果があり、「検証に関するものである」という限定的な言及は正しいとは言えない。

この神話の反例としては、BPS1000 紙幣検査機の事例が挙げられる。この事例では、形式手法を導入した動機は工期短縮であり、その工期短縮の手段とは、複雑な要求から簡明で見通しのよい仕様記述を策定することであった。そうして得られた良質の仕様記述に基づいて開発し、テストを経て、結果として欠陥の少ないシステムの開発に成功した。

BPS1000 紙幣検査機の事例においてもテストという形での検証は行われた。形式仕様を使ったテストにより、効果的なテストが実現された。この点からは、形式手法により、より効果的なテストが実現されたとは言える。しかし、効果的なテストは良質な形式仕様がもたらす好影響の 1 つに過ぎない。したがって、形式手法はプログラムの検証に関するものである、という限定的な言及は正しいとは言えない。

神話(誤解)3. 形式手法は高信頼性システムの開発にのみ役に立つ。

誤解 1, 2 で述べた通り、形式手法は高信頼性システム以外にも適用され、成功している。工期短縮や効果的な要求獲得はソフトウェア開発において広く求められている事柄であるが、本調査の対象となった事例においても、それらを目的に形式手法を適用し成功している。

神話(誤解)4. 形式手法を用いるためには高度な教育を受けた数学者が必要である。

本調査で調査対象となった事例では、形式手法の教育は 1 週間程度のコース受講により基礎的な能力は身に付き、残りは作業中の指導により 1 週間程度で仕様を読む事ができ、3 ヶ月程度で仕様を記述することができるようになる。本調査のヒアリング対象の複数が「形式手法を適用した開発に参加するのには、高度な数学教育は必要ない」「数学よりも開発対象となっているドメインの知識が重要である」と語った。ただし、形式手法に適性が全くないわけではない。ヒアリング対象の複数が「実装経験が豊富な技術者が、命令的なプログラミングから抽象的記述や抽象的思考にシフトするための柔軟さを欠いていると、形式手法を身につけるのが困難になることがある」と語った。しかし、これは形式手法特有の問題ではなく、新しい技術を習得する上で必ず伴う関門である。

神話(誤解)5. 形式手法は開発のコストを増加する。

BPS1000 紙幣検査機の事例をはじめ、調査対象となった成功事例では開発企業は形式手法を使って開発のコストをコントロールしている。ヒアリング対象の 1 人は「我々は、顧客との合意に形式仕様を使っている。そして、顧客が追加要求や仕様変更を申し出た時には、元の形式仕様からどれだけの分量の変更が必要で、その変更により開発コストがこれぐらい必要である、という提示をすることができる」と語った。形式仕様から工数を見積もり、それに基づいて顧客との協議も含めて、コストをコントロールしている。

神話(誤解)6. 形式手法はユーザには受け入れられない。

本調査では、ユーザとの協議に形式手法を直接用いない事例と、ユーザが形式手法を学習し開発に参加した事例の両方を調査対象に含めた。形式手法を直接ユーザに見せずに開発する場合においても、形式手法は有効である。例えば、オランダ軍メッセージ分析の事例では、開発者は顧客との対話に形式仕様を使わずに、従来の自然言語や UML やデザインパターンを使ってシステムに関する議論を行っている。そのような場合であっても、開発者側が形式仕様の記述や実行を通して対象ドメインをより良く理解し、ドメイン専門家である顧客でも気付かなかったような例外事項に気付き、効果的な要求獲得につなげた。また、顧客が形式手法を学習する場合であっても、たとえ顧客がシステム開発に関する知識や経験がなくとも約 1 週間のコース受講と形式手法専門家の補助があれば形式仕様のレビューを行うことが可能である。

神話(誤解)7. 形式手法は実際の巨大なソフトウェアには用いられていない。

iFACTS 事例のような、巨大でミッションクリティカルなシステム開発に対しても形式手法は有効である。また、オランダ花市場事例からは、いわゆるレガシーシステムへの拡張にも適用可能であることがわかる。BPS1000 紙幣検査機事例は複雑かつ自由度の高いシステム構成や制約条件からくる複雑さに対しても形式手法が有効であることを示している。

以上により、Anthony Hall の「形式手法の 7 つの神話」として知られる、形式手法の効果に関してしばしば言及される誤解が、本調査でのヒアリング結果の分析によって反証することができたと考える。

6. 技術的解決策の抽出

本章では、5 章までに説明したヒアリング結果の分析に基づき、

1. 形式手法を上流工程における記述法として導入する際の技術的解決策の抽出
2. 日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出

という 2 つの項目を実施した結果を報告する。

6.1 形式手法を上流工程における記述法として導入する際の技術的解決策の抽出

6.1.1 サマリ

形式手法を上流工程における記述法として導入する際の下記のような要件が、ヒアリング事例結果の分析を通して明らかとなった。

- 問題領域の用語をそのまま記述したり、自然言語記述と形式言語記述の併記を支援したりできる。
- ある観点におけるプロジェクト全体に渡る仕様を記述することができる。
- ある程度の検証（準備、実施、確認）を（半）自動的に行うことができる
- 仕様そのものや仕様に付随する情報を様々な形（図、表、自然言語）で抽出・加工したり、逆に取り込んだりするための仕掛けをもつ。
- 複数の形式手法間の役割分担を支援できる。
- 過去の開発資産を簡単に参照できる。

以下本章では、これらの要件を軸に、それらをどのように実際の開発プロジェクトの中で技術的に解決していけるかについて、ヒアリング結果の分析から得られた知見を踏まえながら考察を行う。

この考察を行うために、まず開発における仕様の位置付けと、要求を仕様化する際の手順について確認し、上記の要件とその技術的解決策について考察する。

6.1.2 形式仕様の位置付けと仕様化の手順

(1) 仕様の位置付け

以下に示すのは仕様記述に形式手法を取り入れた際の、仕様の位置付けを示した図である。

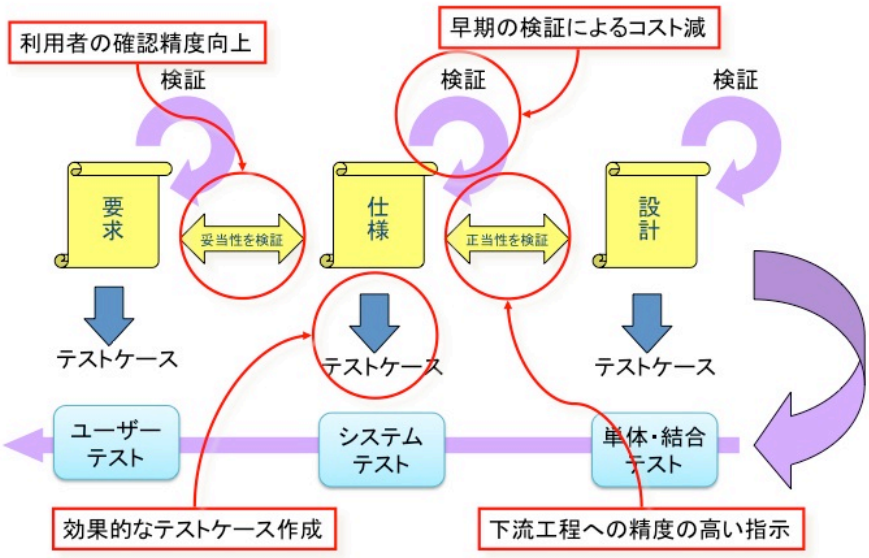


図 6-1: 仕様の位置づけ

この図は仕様を形式的に書くことによる各方面への波及効果について示している。この効果を大きく 4 つにわけて解説する。

第一は妥当性検証の支援である。要求と仕様の間では、妥当性検証(Validation)を行う必要があるが、それを効果的に支援するためには、要求の中に現れる概念（例えば用語）が、仕様の中のどの部分に対応するかを追跡しやすい形になっている必要がある。

仕様記述の中に、問題領域の用語をそのまま記述したり、仕様の中の記述に自然言語による説明を付加したりすることにより、要求と仕様の対応関係はとりやすくなる。

なお、ただ対応関係がとれただけでは、場所を見つけやすいというだけに留まるので、実際に書かれたものが正しく書かれているか否かを判断するには内容に関する検証を行う必要がある。

第二は仕様の検証である。仕様はそれ自体が設計へ渡される前に検証を行うことができる。記述された仕様そのものに矛盾や間違いがないかどうかを検証するのである。

この検証には事例の Z 記法を使用したプロジェクトで行われていたような証明や、VDM を使用したプロジェクトで行われていたような仕様アニメーションによるテストなどが含まれる。もちろん人間によるレビューも必須であるので、仕様記述の読みやすさも大切である。

なおここで言う「検証」には 2 通りの意味がある。要求に対して仕様は満足しているかを問う「妥当性検証（Validation）」と、仕様記述そのものの内部に矛盾がないか否かを確認する「正当性検証（Verification）」の 2 種類である。開発のなるべく早い段階で正当性検証を進めることによって、作られつつあるシステムが利用者の要求を満たしているか否かを早い段階で判定することができるため、実装が終わって最終段階になってから「思っていたものと違う」といった大きな手戻りが発生することを未然に防ぐことが可能となる。

第三は設計（下流工程）への精度の高い仕様の引渡しである。仕様は対象とするものが「どのようなものであるか」の性質を規定するものであり、設計はその規定された性質を、具体的な手段を用いて「どのように実現すればよいか」を案出する作業である。

このため設計から見た仕様には「どのような前提条件下で」「どのような結果が得られれば良いのか」「制約条件は何か」を厳密に定義するという役割が求められている。「形式手法」を使わなくともこうした記述を日本語で行うことは原理的には可能である。

しかしながら、これは扱う対象がほんの小さな問題である場合に限られるのである。少しでも規模が大きくなったり、関わる人間が 2 人以上になったりしただけで、すぐに日本語だけの仕様記述の管理は大変手間がかかるものとなる（もちろん厳密に行おうとしなければ、そこまでの苦労は発生しない）。

形式仕様記述を用いることにより、こうした仕様の管理の手間をある程度機械的に行うことができるようになる。もちろん仕様そのものを考えるのは人間の仕事であるが、複数の記述の間の「用語」統一や、アニメーションによる値の計算などの（人間にとっては）退屈で間違いやすい作業を機械任せにできる効果は大きい。

第四は仕様の検証に使った評価仕様を、実装の評価にも使うことによるテストの精度の向上である。たとえ形式仕様を使ったとしても、最終製品に対するテストの必要性は無くなることはない（テストの位置付けは変わる可能性はある）。

今回事例調査した全てのプロジェクトも、最終製品に対する厳重なテストを行っている。形式手法を使わない旧来のやり方では、テストケースは人間が文書をもとに仕様策定者や開発者に対して質問を繰り返しながら作り上げて行く場合が多かった。

しかしこうしたやり方は、多くの場合評価責任者にとって大きな負担を強いるものであった。仕様そのものが厳密に検証されているわけではないので、テストケースの切り出しも、正しい振舞の定義も、あまり頼るものがない場所から始めなければならないのである。

形式仕様を用いた開発の場合、仕様を検証した際のテストケースを実装のテストに対して再利用することが可能である。特に仕様アニメーションを使った検証を行っている場合には、その移行は円滑である。仕様アニメーションを行って仕様のテストを行うことにより、テストケースそのもののテストも先に行っていることになるため、実装テストの段階では相当量の質のよいテストケースを手にしていることができるからである。

なおここで述べているテストケースは皆仕様に対するテスト（ブラックボックステスト）であり、設計に対するテスト（ホワイトボックステスト）は別途設計しなければならない（設計段階にも形式手法を用いることもできる）。

(2) 仕様化の手順

ここでは何らかの要求記述（通常の日本語や、ユースケース記述、UML の図など）を読み、それを形式仕様記述として仕上げて行く際の、概要手順を示す。

もともと VDM++を使ったモデル化の手順として想定されたものだが、他のいわゆるモデル規範型の仕様記述（Z 記法や B など）にも適用可能である。更に言えば UML などを用いるいわゆる「オブジェクト指向開発」にも応用可能なものである。

なお手順の進め方は、厳密にこの順序でででなければならないいうものではなく、複数の手順を同時に行ったり、一部先に進んでは少し戻ったりを繰り返しながら行われる。

(1) 要求記述の内容のうち、問題領域自体の性質（システムが変えることのできない部分）と、構築対象（システムが影響を及ぼす部分）への記述を区別する

(2) 要求記述を読み、用語を切り出す

(3) クラスもしくは型と、関数もしくは操作の候補を見つける

(4) クラスの構造（型、属性、関連）に関する概略を書く

(5) 関数／操作に関する概略を書く（シグニチャを決める）

(6) 要求記述から不変な性質を抽出して定義する

(7) 事前条件、事後条件もしくは定義本体を記入することによって、関数／操作定義を完成させる、必要ならば型定義を変更する

(8) 作成された仕様のモデルが要求中に現れる項目や制約を満たしているかどうかを検証する

(9) 以上のステップを必要なだけ繰り返す

以下それぞれのステップを簡単に説明する。

(1) 要求記述の内容のうち、問題領域自体の性質（システムが変えることのできない部分）と、構築対象（システムが影響を及ぼす部分）への記述を区別する

要求記述の中には様々な事柄が書かれているが、その中には「事実」（システムの責任範囲ではなく、その内容を変えることができないもの）と「願望」（構築したいシステムによって実現したいと思っている振舞）が混ざっている。

例えば今、特急券の予約システムを考えたいと思っているとしよう。要求記述の中に

“出発駅と到着駅を指定してある列車の座席を予約する”

という記述があったとする。この場合「予約システム」が行うべきことは、ある列車が提供するある座席を、指定された駅の間確保されたものとして関連付けることである。しかし、「ある列車」「ある座席」「出発駅」「到着駅」という部分は実はこの予約システムの中に存在しているのではなく、路線、運行といったもの（事実）の中に存在していて、内容を変更すること(願望)は普通できない。

最初の段階であまり細かく分けると扱いにくくなってしまうが、この区分を意識することにより、仕様記述を適宜分割し保守しやすいレイヤーに分類することが可能となる。

(2) 要求記述を読み、用語を切り出す

要求記述の中に含まれる「用語」を切り出す。この用語は、あるものは型や属性に、また別のあるものは関数の候補となっていく。

例えば予約の例で言うならば、「座席」「列車」「予約」「出発駅」「到着駅」などは用語の候補である。

なお日本語の「予約」という言葉は「予約する行為」もしくは「予約情報」の両方の意味を含んでいる。仕様記述言語の中に記述する際にはどちらの意味で使っているかを区分して記述することになる。

(3) クラスもしくは型と、関数もしくは操作の候補を見つける

上記で抽出した用語は、ステークホルダが理解しやすい問題領域の用語であり、基本的に要求記述の話をする際にはその用語を利用する。一方形式仕様はこの用語を出発点に記述の単位と範囲を決めて行く必要が出てくる。

要求を満たすために必要な仕様を考える際に、その説明に必要な用語を候補として選び出す。

例えば対象とする仕様全体に「特急券予約仕様」という名前をつけてこれを管理単位とし、その中に「新規予約」という機能があるとする。もちろんこれらは要求記述の中に存在する用語である。こうした用語が仕様として記述される候補となる。

(4) クラスの構造（型、属性、関連）に関する概略を書く

候補として選び出した用語を、型、属性、例えば VDM++の場合ならこの全体を「特急券予約仕様」クラスとして定義し、その中に「新規予約」関数があるものとして定義できる。

また先に述べたように、列車や、座席、駅などは他の場所から取り込むことを考えた方が良いかもしれない（とはいえ実用上、最初のうちは 1 つの世界の中で記述した方が簡単である。もし強力なツールの支援があれば最初から分離して記述した方が良いかもしれない）。

(5) 関数／操作に関する概略を書く（シグニチャを決める）

ここまでの段階で、基本になる型や属性そして関数の候補は出されているので、それを並べてみる。

なおここでも構文は VDM++のものを使っているが、定義をとりまとめる単位、とその中に含まれる単位などの概念を適宜他の形式仕様記述言語でも利用することが可能である。

```
class 特急券予約仕様

  新規予約(rs:予約一覧, rr:予約要求) rrs:予約一覧, rsv:  予約

end 特急券予約仕様
```

ここでは、特急券予約仕様の中に「新規予約」という機能定義されたことを示している。ここでは「予約一覧」と「予約要求」を受け取って、「予約一覧」と「予約」を返しているということがわかる。

なおここでは、言語のキーワード(class, end など)をそのまま使っているが、これを隠したり、あるいは表のような形式で記述したりすることも（利用者が望めば）可能である。

(6) 要求記述から不変な性質を抽出して定義する

構築対象の問題領域で、必要となる不変条件を定義する。これは個別の機能仕様と直交していて仕様の検証に役立つ性質である。

ここで、要求記述の中にある制約を表すと思われる述から用語を選んでおくことができる。例えば「同一の利用者が同じ日に 2 つ以上の予約をとることはできない」という制約が

あったとすると、それを不変条件として定義しておくことができる。このとき制約条件を判定する関数を

不変条件_同一利用者は同じ日に 2 つ以上の予約をとることはできない(予約一覧)

結果:bool

といった名前で用意しておくと、あとからその部分を要求記述と対応つけてレビューしやすい。

(7) 事前条件、事後条件もしくは定義本体を記入することによって、関数／操作定義を完成させる、必要ならば型定義を変更する

各関数に事前事後条件を定義しておくことによって、機能仕様として完成させる。事前事後条件を定義した段階で、設計が満たすべき性質は記述できたことになるが、この段階で仕様アニメーションを行うなら、更に関数の本体部分を記述しておく必要がある。

(8) 作成された仕様のモデルが要求中に現れる項目や制約を満たしているかどうかを検証する

モデル規範型の形式仕様記述言語は証明か仕様アニメーションといった手段を用いて検証される。証明の場合には、ツールの手助けを使いモデルから証明課題(Proof Obligation)を生成して、それを人間の手で証明するというスタイルが一般的である。今回の事例でも Z 記法を使ったものはそちらのアプローチをとっていた。

一方仕様アニメーションは通常の実行テストのように入力を与え出力を検査することによって振舞の正しさを見ようとするものである。

仕様アニメーションは 2 つの側面を持っている。1 つは入力を与えて実際の値を計算させることであり、もう 1 つは、その際に各種表明が満たされるかを検査することである。

実際に値を計算させる場合には、通常のテストと同様に計算結果とテストオラクルとの比較を使って成功／失敗を見ることができる。そして同時に事前条件、事後条件、不変条件が満たされているかを検査して矛盾を探すことになる。

こうした検証を経て、モデルの「正しさ」（妥当性、正当性両者の意味で）が検査される。

(9) 以上のステップを必要なだけ繰り返す

ここまで辿ってきた手順は一度で終了するのではなく、必要な範囲を必要な精度で定義し尽くすまで繰り返し行われる。

なお必要な範囲とは、要求記述が想定する記述が漏れ無く取捨選択されて、ここで記述された仕様に写像された状態になったときの範囲のことであり、必要な精度とは入力のパターンや機能の適用のパターンの組合せが必要十分な粒度で洗い出された状態を指している。

6.1.3 形式手法を上流工程における記述法として導入する際の技術的解決策

前節までの記述で、開発における仕様の位置付けと形式仕様記述を用いたモデル化の一般

的手順について、ヒアリング結果によって知見として得られた点について説明をした。これらの知見をもとに、冒頭に挙げた「形式手法を上流工程における記述法として導入する際の技術的解決策」について下記のような考察を行った。

(1) 問題領域の用語をそのまま記述したり、自然言語記述と形式言語記述の併記を支援したりできる

本調査での大きなテーマでもある「形式手法を用いた日本語による仕様書作成」へも直結しているのがこの要件である。上流工程での仕様は要求との突合せを行い、妥当性を検証しなければならない。

なお、この仕様と要求の突合せ作業には、形式手法の非専門家が参加する場合があるので、その場合はそもそも形式仕様そのものではなく別の表現を工夫する必要がある。専門家と非専門家の間のコミュニケーションに対する要件の考察はこの後の節で述べることとし、まずここでは「要求と対応付けしやすい形式仕様記述」という観点で考察を行う。

用語を仕様記述言語における型、述語（関数）のレベルに対応付けて管理することができることが必要であるので、そもそも形式仕様記述言語そのものが日本語（問題領域の用語）を扱えなければならない。幸い近年の UTF-8 の普及により、ほとんどの処理系は様々な識別子を扱うことができるようになっている。

よって基本的な扱いには現在ほとんど問題がなくなりつつあるが、要求記述のドキュメントとしてのいわゆる「用語集」との統一管理にはまだ決まった形は提案されていないようである。

しかし、形式仕様記述の型の定義や、関数のシグニチャなどを、通常のドキュメントツールの表の形式に変換したり、データベースに格納してレポートライタの機能で文書として書きだしたりということはスクリプト言語などを用いれば比較的簡単に実現することができる。

Z 記法を使った事例ではいわゆる MS Word のマクロを用いて文書から形式記述の部分を抜き出して形式仕様記述処理系で処理するといったことも行われていた。また適当な大きさ（1 頁程度）の Z 記法の定義とその説明を併記して管理し、説明の中では Z 記法のスキーマを参照しながら索引を作成するという試みも行われていた。

形式仕様記述と自然言語記述の併記は、適度な大きさの記述の塊を併記する方式と、個別の式に対して注釈を添えるものの 2 通りが見られたが、人間がレビューする上では前者の方がまとまった意味をレビューできるので集中しやすい印象も受けた。例えば後者の方式をとっていたのは FeliCa ファームウェア(1)であるが、FeliCa ファームウェア(2)では併記を減らす方向へ工夫し、形式仕様そのものを読みやすい形式で記述することにした。

ある程度まとまった大きさの説明を添えて、形式仕様記述そのものを読みやすくするような工夫（問題領域の用語になるべく限定するなど）を行うと効果的かもしれない。この部分についての定量的な比較結論はまだ出せる段階ではない。

(2) ある観点におけるプロジェクト全体に渡る仕様を記述することができる

今回の事例調査で調べたプロジェクトは皆、形式仕様記述によって、厳密な仕様を表現していた。

要求記述レベルでも、オランダ軍メッセージ分析の事例では要求記述全体がある形式に沿った自然言語で記述してあり、MULTOS や Tokeneer の例ではセキュリティに関する要求を仕様記述の前に形式的に記述していた。

総合分析のところでも挙げた「厳密な記述対象」の表を再掲する。

表 6-1: 各事例に関する厳密な記述対象（再掲）

事例	記述対象
FeliCa(1)	FeliCa チップファームウェア外部仕様(API の仕様)
FeliCa(2)	FeliCa チップファームウェア外部仕様(API の仕様)、API 仕様から派生したデシジョンテーブル（テストケース）
BPS1000	銀行券鑑別仕様（センサデータを抽象化した）
SHOLIS	機能仕様、詳細設計(SPARK)
iFACTS	状態遷移、機能仕様、詳細設計(SPARK)
TradeOne	業務論理（入力伝票、出力伝票、帳簿の状態）仕様
オランダ軍メッセージ分析	要求仕様：IRS（入出力の要求。形式の定まった自然言語） 機能仕様：IRS を満たす入出力の関係 評価仕様：IRS を満たす評価仕様
CAVA	AST の不変条件 AST の変換機能仕様
MULTOS CA	セキュリティ要件 機能仕様 並列性設計(CSP)
Tokeneer	セキュリティ要件 機能仕様
オランダ花市場	システムの振舞要件（UML クラス図、コラボレーション図を要求定義に利用。この内容を Rose を使って VDM++とラウンドトリップ） ソフトウェアモジュール機能仕様（VDM++）

この記述範囲に着目すると、どのプロジェクトも「ある特性」を記述対象にすると決めたなら、プロジェクト内の全ての特性を完全に記述し切っていることがわかる。

例えば FeliCa や TradeOne の場合、基本的に前者はチップの全外部仕様、後者は全業務論

理の仕様を書いている。言い換えると、「誰かの担当分だけ形式的に書いてみた」ということではなく、プロジェクトの正式な成果物として記述対象を定め、それを全て記述するというマネジメントレベルのコミットメントが行われているのである。

先にも挙げたが MULTOS CA や Tokeneer の場合は、仕様記述に先立ちセキュリティ要件を厳密に形式的に記述して、それが仕様の中できちんと保全されるかどうかを証明している。こうしたことは「セキュリティ要件の全て」「セキュリティに関わる機能仕様の全て」を記述しないと無意味になってしまう。

いずれもある特性 - セキュリティ、API、業務論理、銀行券鑑別仕様、機能仕様の全てなどを完全に書いている。そうすることによって、その特性に関しては必ずどこかに起点の記述が存在することになり、形式仕様記述を「辞書」のように使えるようになるのである。

今回の調査対象プロジェクトでは、こうした網羅性の管理自身はみな手作業で行われていた。ソーステキストを TeX などで管理していたプロジェクトは、目次や索引という形で一覧を作成することもできていたが、要求記述との突合せなどには人間の作業が介在していた。この部分を強力に支援すれば、辞書としての形式記述の価値が更に増し、使いやすいものになるだろう。

一番単純なのは形式仕様記述言語のためのブラウザを用意することである。なおここで言うブラウザとは Web ブラウザではなく、Smalltalk 言語のブラウザのように自らの内部構造を参照し提示できるようなブラウザのことを意味している。

(3) ある程度の検証（準備、実施、確認）を自動的あるいは半自動的に行うことができる

モデル規範型の言語では何らかの形で、機械的に証明課題を生成できるが、その証明作業そのものは B などを除きあまり機械的には強力に支援されていない。

よってこうした証明課題を対話的にガイドしながら証明する環境を整えていく必要がある。また仕様変更が行われた際には証明を再度行わなければならないが、過去の証明の知識をなるべく再適用して開発者の労力を削減するツールが必要となる。

証明の対話的支援機構の開発そのものは、それぞれの言語で徐々に進んでいるが、例えツールができても、使いこなすには更にそれなりの訓練とスキルが必要になるためその普及はかなりゆっくりであると思われる。

これに比べて仕様アニメーションを用いる方法は、旧来のテスト設計の知識も応用しやすく、大量の試験を機械的に何度も繰り返し行うことができるために現場での適用が行いやすい。

仕様アニメーションを使って効果的な検証を行うやり方の 1 つとして以下のものが考えられる。

①.事前条件から妥当な入力群を構成する

②.①で作成した入力群を用いて仕様アニメーションを行う。

③.②の結果、事後条件違反、不変条件違反が起きた場合を吟味し、それが陰仕様の間違いなのか（=仕様のミス）、陽仕様の間違いなのか（=想定したアルゴリズムのミス）を決定する。

④.③で条件違反が起きなかったものに関しては、返り値や状態の変化に関する吟味を更に行い正しい計算や操作が行われていることを確認する。この確認は事前に計算を行っておいた期待値との比較で行うことになるが、事後条件の精度が高ければ期待値そのものを事後条件の関数で計算することができるため③の段階で既に結果がわかっている可能性もある。

⑤.①から④を繰り返しながら仕様の検証精度を上げていく。

このような手順を繰り返し、仕様の検証を進めることによって、③の実行結果（計算結果）も蓄積されていく。この計算結果は実装テストのためのテストオラクルとして利用することができる。

こうした作業に必要な、事前条件を満たす入力の生成などは、例えば集合の内包表記などを用いて大量に行うことができる。例えば VDM の場合には

```
{ g(x) | x in set {1,…,100} & f(x) < 150}
```

という表記は x が 1 から 100 の間で f(x)が 150 より小さくなるような x を選びそれに対応する g(x)の値を要素として集めた集合を表現することができる。

こうした集合を GX という名前で呼び、他の集合内包表記で更に使うことにすれば複雑な条件の組合せのデータを比較的簡単に生成することが可能になる。

これを事後条件、不変条件できちんとチェックしつつ出力値も検証すれば、この出力値を実装テストのためのテストオラクルとして利用することができることになる。

以上のような仕様アニメーションの準備（パラメータの管理）と、テストの実施と結果をうまく管理して、実装テストと連携させるための仕掛けを工夫する必要があるが、簡単なスクリプトで実現できるものも多い。

また最近注目されている CI（継続的インテグレーション）の環境もうまく取り込めば、仕様アニメーションのテストを実施した直後にシステムテストへと連動させ、高い信頼性のある検証を行うことができるようになりそうである。

⑷ 仕様そのものや仕様に付随する情報を様々な形（図、表、自然言語）で抽出・加工したり、逆に取り込んだりするための仕掛けをもつ

形式仕様記述言語による表現をそのまま、形式仕様記述に慣れていないステークホルダに見せることはかなり難しい。そもそも形式仕様記述そのものは、どちらかといえば開発者向けに全部の視点を盛り込んで書かれたものであり、一部の情報の組合せだけに興味のあるステークホルダにとっては余計な情報が多すぎることになるかもしれない。

こうしたことから、形式仕様記述言語を現場で使う際には、様々な表現形式への変換を取り込み、必要に応じて図式表現や一覧表などの表現形式、他のツールへの入出力形式に変換して渡したり受け取ったりという仕掛けを考慮する必要がある。

これまではこうした仕掛けは後付けで、仕様記述言語のサブセットパーサーを改めて開発

して一部を変形したりするのが常道であった。しかしながら、このやり方では高度なツールを作成しようとする場合むやみに余計な労力がかかってしまう傾向があった。

この先、形式仕様記述言語処理系を導入する場合には、まずは、構文と型の検査が終わった状態の抽象構文木の構造を作り、それに対してパターンマッチなどで検索をしたり一部の構造を取り出したりするような問合せをかけることができるようにすべきであると思われる（問合せに比べると更新する側はより難しいと思われる）。

処理系によってはこうした外部のツールからのアクセスを許す API を整備しはじめているものもある（例：VDM を支援するオープンな処理系 Overture など）。

(5) 複数の形式手法間の役割分担を支援できる

1 つの開発プロジェクト内で使われる形式手法は 1 つだけとは限らない。例えば MULTOS CA の場合機能仕様は Z 記法で記述され、並列設計は CSP が使用された。

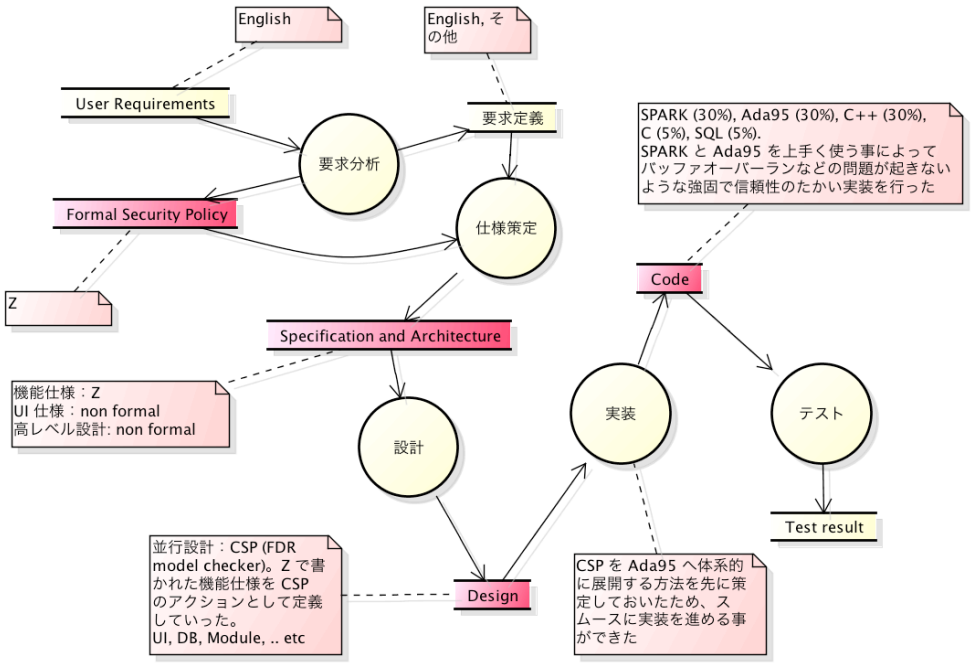


図 6-2: 形式手法間の役割分担（MULTOS CA）

また Tokeneer の場合は Z 記法による詳細化を受けて、詳細設計レベルでは INFORMED という方法論が用いられ SPARK 言語による実装が行われた。

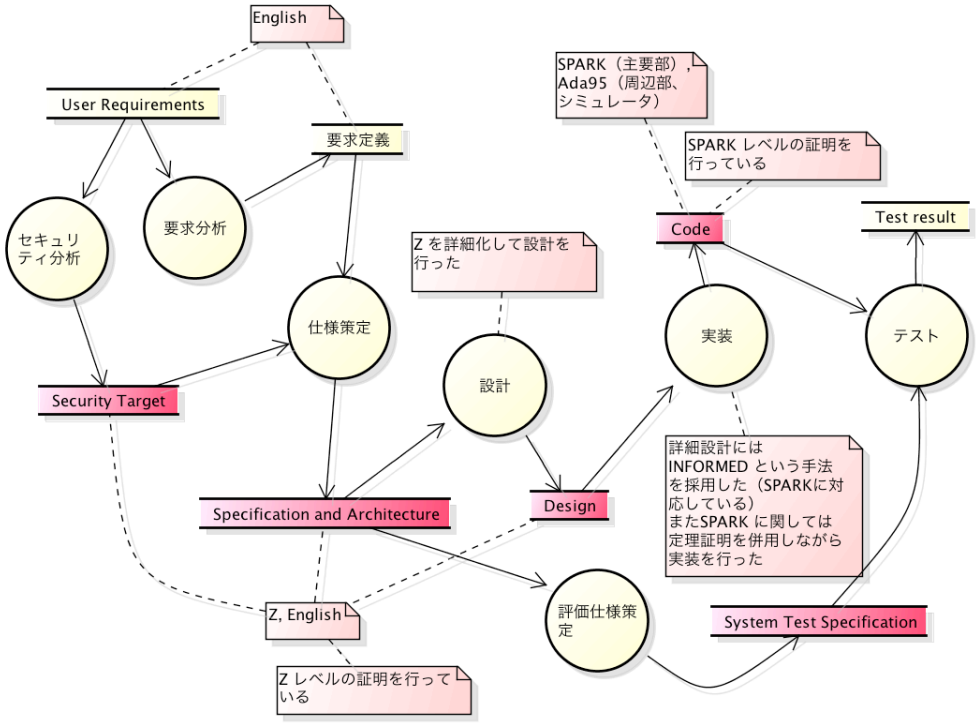


図 6-3: 形式手法間の役割分担（Tokeneer）

こうした連携が行えることによって、仕様や設計などの円滑な移行が可能になるのである。現時点ではこうした連携を自動的にやってくれるツールが整備されているわけではないし、将来的にも作り込まれることはあまりなさそうである。

その代わり、先の項で述べたような形式仕様記述に対する問合せや変換機能が充実すれば、相手の方法論が必要とする情報を生成して渡すことができるようになるだろう。

またこうした状況でも役に立つように、先の項でも述べたが、ある特性に関する記述はプロジェクト内の対象全範囲に渡って行うべきである。

(6) 過去の開発資産を簡単に参照できる

要求記述を分析した際に抽出された様々な用語のうちいくつかは、その問題領域に深く関連していて、他の応用分野でも再利用することで価値を高めることができる。

こうしたことから仕様記述を行う際には将来の参照の利便性を考慮して、ある程度の分類を行って整理しておくべきである。

以下に示す分類は、形式的仕様記述を分類する際のひとつの例である。

表 6-2: 形式的仕様記述の分類例

記述階層名	記述目的	記述例
検証シナリオ階層	仕様アニメーションを行うための検証シナリオ（テストケースなど）を定義した階層	(例えば非接触 IC カードの場合) 個別コマンド検証シナリオなど
機能仕様階層	特定の応用に対する機能仕様を定義する階層	カード操作コマンド仕様など
問題領域データ階層	問題領域に特有の共通データ等を定義した階層	具体的なファイル構造など
問題領域階層	共通問題領域で繰り返し利用される関数や不変条件などを定義する階層	カードファイルシステム、データパケットの変換など
共通領域階層	特定の問題領域に依存しない階層。ネットワーク（数学的な意味で）や、組合せアルゴリズム、文字列や数値に関する共通処理などがここで記述される	階層型データ処理など

この階層は、上の階層が下の階層を利用する構造となっている。記述量は、一番上が多く下に向かって少なくなっていくのが普通である。

また「検証シナリオ階層」「機能仕様階層」は、開発対象のシステム固有の用語が定義され、問題領域の用語とデータを下の「問題領域データ階層」「問題領域階層」から参照することになる。この部分に日本語（自然言語）を多く使い、記述形式を少し工夫し、日本語（自然言語）の解説文書を添えれば日本語（自然言語）の仕様書として読みやすいものになる。

6.2 日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出

6.2.1 サマリ

3 章および 4 章に示したヒアリング結果を分析し、日本語による仕様の記述において誤解の挿入を少なくするための成功事例として分析した結果、記述、追跡、比較、導出という 4 つの項目が重要であることが認められた。さらに、各項目について必要となると考えられる、ヒアリング対象者が言及した基本動作を列挙した。これらの基本動作の多くは、仕

様記述を扱う何らかのツールの上で行われているが、現在のツールではそれらの基本動作への支援が十分とは言い難い。

4 章に詳述した、ヒアリング対象者からの聞き取り内容を踏まえ、基本動作を支援するために必要な要件を整理したところ、仕様記述を支援するツールとして、(i)語彙管理機能、(ii)テキスト分析機能、(iii)記述からの参照機能、(iv)記述間の突合せ機能、(v)構成管理機能という 5 つの機能を実現することが必要であることがわかった。

本節では、日本語と形式仕様を併用した仕様記述の実施を支援するためのツールが提供すべき機能について分析し、技術的解決策として抽出する。ツールが提供すべき機能であり、これらが具体的なツールの仕様の策定とつながるものではない。換言すると、

- ツールは統合環境もしくはその拡張部品であるか、あるいはツールキットとしての小規模プログラム集であるかは想定しない。
- ツールの機能の抽出であり、特定のプロセスや方法論、形式言語を想定しない。
- 各機能を 1 つのプログラムが提供するのか、もしくは複数のプログラムを組み合わせて 1 つの機能を構成するのかは、想定しない。
- ここで抽出するのは抽象的な機能であって、具体的な入出力の形式等は規定しない。

以下に、4 章で説明した、個別分析を行った成功事例から、自然言語と形式仕様の併用に関連した作業で発生した相互作用を列挙し、そこから作業者のより小さな粒度の基本動作を抽出の上、その基本動作を支援するために必要な機能を挙げる。

6.2.2 成功事例で見られた形式仕様と自然言語による記述の相互作用

以下に、成功事例のヒアリングおよび参考資料から抽出された、自然言語と形式仕様の併用の結果、作業により発生した自然言語による記述と形式言語による記述の相互作用を挙げる。「誰が」その作業を行ったのか、あるいは、人間が行ったのかツールが自動的に行ったのかは抽象し、あくまで記述間の相互作用として抽出したものを相互作用の発生元と発生先によって分類し、列挙する。

A. 自然言語による記述から形式的記述への作用

1. 自然言語で書かれた原始要求が形式仕様で整理される。
2. 自然言語で書かれた仕様が形式仕様に書き直される。
3. 自然言語で書かれた「仕様が保証すべき性質」が形式言語で記述される。
4. 自然言語によるフィードバックが形式仕様に反映される。
5. 自然言語仕様から形式テスト仕様が作成される。
6. 実装の誤りから形式仕様の誤りがわかる。
7. 自然言語による説明によって、形式仕様の意味や性質が理解される。

8. 自然言語による説明によって、形式仕様の目的や理由や意義が理解される。

9. 実装でわかった仕様上の誤りから形式仕様の誤りがわかる。

B. 形式的記述から自然言語による記述への作用

10. 形式仕様を書くことで自然言語で書かれた仕様の矛盾や抜けや例外が見つかる。

11. 形式仕様の修正を自然言語で書かれた説明に反映させる。

12. わからない用語の意味が形式仕様を参照することで理解される。

13. 形式仕様の意味や性質が自然言語で説明される。

14. 形式仕様の目的、理由、意義が自然言語で説明される。

15. 証明してみてわかったことが形式仕様の説明に反映される。

16. 形式仕様が設計／実装として詳細化される。

17. 形式仕様から形式テスト仕様が作成される。

C. 形式的記述から人間の理解への作用

18. 形式仕様が形式化された要求を満たしていることが証明される。

19. 設計／実装が形式仕様を満たしていることが証明される。

20. 形式仕様のある部分の動作と設計／実装の動作が一致するかテストされる。

21. 証明の過程で、その仕様への理解が深まる。

22. 形式仕様をプロトタイプとして実行することで、その仕様への理解が深まる。

23. 形式仕様をプロトタイプとして実行することで、顧客が仕様に対してどの部分に同意し、どの部分に変更を要するかが判明する。

24. 記述のある部分のデザインデシジョンがどの記述によって理由付けられるのかを理解する。

上記分類においては、形式仕様と自然言語による記述がそれぞれ 1 ページや 1 段落程度の量で対応関係にある状況における相互作用が記述されている。しかしながら、形式仕様の中にも識別子名やコメント等の内部で自然言語は使われている。例えば、「元本に対する利子」という関数があれば、この「元本に対する利子」はそれ自体が自然言語の記述とも言える。また反対に、自然言語による記述の中に、「ここで、関数『元本に対する利子』は、口座の種別により利率等が異なる。」という記述がある時、「元本に対する利子」という関数名は形式言語における識別子名そのものである。

例えば、下図のような自然言語表現と形式仕様の併記がある。

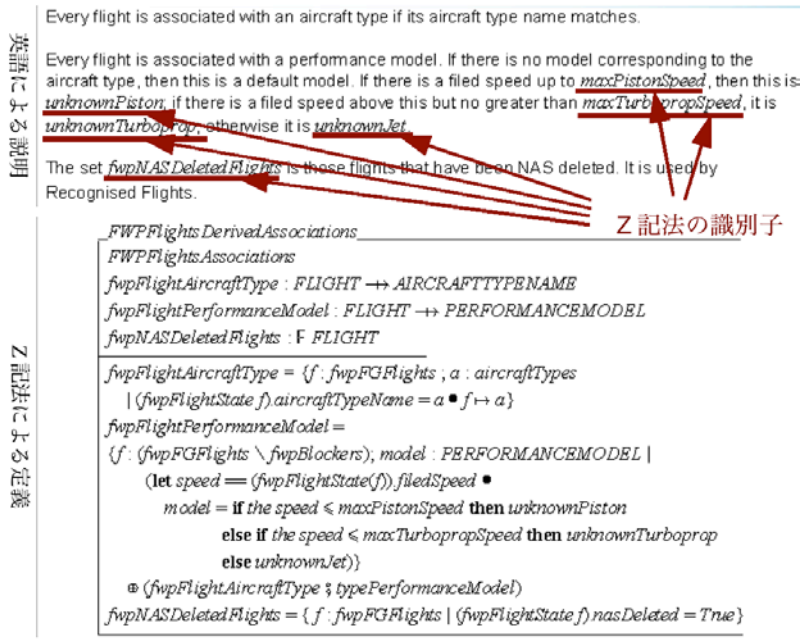


図 6-4: 形式仕様と自然言語表現との併記(参考文献 8 より引用し、説明を補足)

これは iFACTS において記述された仕様の一部である。この併記では、前半が英語による自然言語表現で、後半が Z 記法による形式仕様となっている。英語による自然言語表現のうち、赤い下線を引いた部分が、自然言語に内包された形式言語としての、識別子名である。例えば、3 行目の「If there is a filed speed up to *maxPistonSpeed*, then ...」という表現中の *maxPistonSpeed* とは、後半の Z 記法による定義中で参照されている変数 *maxPistonSpeed* と結びつけて理解されるべきものである。

また、後半の Z 記法による定義では、非常に多くの識別子が用いられている。例えば、*fwpNASDeletedFlights* という変数が定義されている。この変数名は、当然のことながら形式言語である Z 記法の構文要素の 1 つである。しかし、この識別子の字面には自然言語としても意味のある単語が用いられている。すなわち、「Deleted」および「Flights」である。ここで、変数名の字面中の「Flights」とは、自然言語による説明の 1 行目に出現する「flight」と概念として対応している。flight とは、航空管制業務という対象ドメインの用語として定義されている「フライト」であり、後半の Z 記法による定義の実世界における意味を知る上で重要な手掛かりと言える。すなわち、「Flights」は形式仕様中に内包された自然言語要素である。

以上のように、形式言語と自然言語の併用として、まとまった記述のペアの併記のほかに、形式言語による記述に内包されている自然言語としての要素と、自然言語による記述に内包されている形式言語としての要素という関係が存在する。

併記による併用と内包による併用は、上記に掲げた相互作用のいずれにも見られる。次節では、相互作用を作業者のより細粒度の基本動作に分解し、その基本動作の中で併記と内包がどのように利用されているかを分析する。

6.2.3 相互作用を伴う作業者の基本動作

本節では、前節で説明した、作業により発生した自然言語による記述と形式言語による記

述間の相互作用と、記述という作業に関わるより小さな粒度の基本動作との関わりを説明する。作業は、(1)記述、(2)追跡、(3)比較、(4)導出という、4 つの項目に分けて説明する。なお、以下に示す相互作用の番号は、前節で列挙した相互作用の番号である。

(1) 記述作業

- 自然言語表現に対応する形式仕様を記述する。

  関連する相互作用：1,2,3,4,5　(以降、相互作用の番号については 6.2.2 を参照)

  後述の「追跡」および「比較」を伴いながら、形式仕様を記述する。形式仕様の記述は基本動作としては比較的大きな粒度の作業ではあるが、編集という作業の最中の継続的な支援が必要であると考えられる。例えばプログラム用エディタではソースコード中の構文キーワードをハイライト表示したり、常にバックグラウンドで構文解析をして構文エラーや型エラーを発見したりして、ユーザであるプログラマに次にどのような基本動作をすべきか、例えば型エラーを熟読するべき、変数の宣言個所を参照するべき、といった助言的な支援を行っている。形式的仕様記述を支援するツールも同様に、形式仕様を記述している最中にも、次に取るべき基本動作を決めるための参考となる情報提供が行われることが望ましい。

- 形式仕様に対応する自然言語表現を記述する。

  関連する相互作用：11, 13, 14, 15, 16, 17

  後述の「追跡」および「比較」を伴いながら、自然言語で記述する。自然言語での記述は基本動作としては比較的大きな粒度の作業ではあるが、編集という作業の最中の継続的な支援が必要であると考えられる。特に、形式言語の内容の説明や、形式言語での定義の根拠などを示す際には、自然言語中に内包される形式仕様での識別子や、形式仕様に内包されている自然言語要素の語彙との対応関係を把握する必要がある。

- 形式仕様中に識別子やコメントに内包される自然言語要素を記述する。

  関連する相互作用：1, 2, 3, 4, 5

  識別子やコメント中の自然言語は形式仕様が規定する構造を対象ドメインでの事象に対応するための重要な参照を与えることから、語彙の管理が重要である。個別分析対象となったプロジェクトの多くでも、識別子中の語彙が用語辞書などで管理されていた。用語辞書の語彙による索引化などで未定義用語の使用を検出したり、形式仕様と自然言語による記述の追跡性を確保したりするなどの工夫が可能であり、記述作業においてツールでそのような仕組みを提供することが望ましい。

- 自然言語中に内包される識別子などの形式言語要素を記述する。

  関連する相互作用：10, 11, 13, 14, 15, 16, 17

  自然言語中の識別子等の形式仕様要素は、人間が形式仕様で表現されたシステムのモデルを理解する上で重要な参照を与えることから、正しい参照を与えているか、また、参照すべき識別子を網羅しているかを把握することが重要である。一方、形式仕様で記述された全ての識別子が自然言語中に出現すべきということではなく、そこには自然言語記述者が適切な基準を設定して取捨選択をするのが通常である。そして、形式仕様の変更および自然言語による記述の変更を通しても、適切な参照が守られることが重要である。

(2) 追跡作業

- 自然言語表現から派生した形式仕様を参照する。

  関連する相互作用：24

  原始要求から、その要求項目を満たすために記述された形式仕様を得る場合などが相当する。

- 自然言語表現の元となった形式仕様を参照する。

  関連する相互作用：24

  非形式的な設計文書から形式仕様を参照する場合や、外部向けの説明文書の記述から形式仕様を参照する場合などが相当する。

- 形式仕様から派生した自然言語表現を参照する。

  関連する相互作用：24

  形式仕様を読解している時にその自然言語での説明を探す場合などが相当する。

- 形式仕様の元となった派生した自然言語表現を参照する。

  関連する相互作用：24

  形式仕様を読解している時にその形式仕様での定義に関する理由や根拠を探す場合などが相当する。

(3) 比較作業

- 形式言語で定義／参照する識別子について、対応する自然言語表現中の形式言語要素としての出現の有無および箇所。

  関連する相互作用：10, 13, 14, 15, 16, 17

  形式言語で定義あるいは参照する識別子について、対応する自然言語表現中の形式言語要素としての出現の有無や、出現している箇所を把握する。

- 自然言語表現中の形式言語要素の語彙(識別子)について、対応する形式仕様中での定義／参照の有無および箇所。

  関連する相互作用：4, 6, 7, 8, 9, 19, 20, 24

  個別分析を行ったプロジェクトには、自然言語で説明中の識別子にリンクの設定や、索引作成ができるツールを利用していた。

- 用語辞書で定義され形式言語中の自然言語要素中に出現する語彙について、対応する自然言語表現中での出現の有無および箇所を把握する。

  関連する相互作用：10, 11, 13, 14, 15, 18, 21, 22, 23, 24

  識別子やコメント中の自然言語は、形式仕様が規定する構造を対象ドメインでの事象に対応させるための重要な参照を与える。したがって、識別子中およびコメント中の用語が自然言語表現のどの部分で言及されているかを把握するような、語彙の管理が

重要である。

(4) 導出作業

- 形式表現から形式表現を証明する。

  関連する相互作用：18, 19, 21

  証明は形式仕様を導入する上で必須作業ではない。しかし、しばしば証明は安全基準やセキュリティ基準で要求され、また、高信頼性システムを構築する上で強力な武器の 1 つである。証明が必須でない場合にも、証明という作業を通して仕様および記述対象システムに対する理解が深まることが個別分析対象となったプロジェクトでも報告されている。

- 形式表現から形式表現に詳細化する。

  関連する相互作用：16, 17

  形式仕様を他の表現に詳細化することは形式仕様を導入する根幹部分である。すなわち、非実行可能な仕様記述から実行可能な仕様に詳細化すること、仕様記述から設計や実装を作成することや、仕様記述からテスト仕様を作成することが詳細化の典型である。仕様記述はこれらの活動の中心点(ハブ)であり、形式仕様を有効に用いるためには、詳細化作業の支援は必須である。

- 形式表現を実行して結果を得る。

  関連する相互作用：20, 22, 23

  形式仕様のいくつかは実行可能である。仕様の実行はあくまでその仕様記述を満足させる動作の一例にすぎないが、その一例により開発対象への理解が深まる。特に、記述した仕様への最初のフィードバックがインタプリタによる実行であることは少なくない。また、開発文書に慣れない顧客によるレビューにおいても仕様の実行は非常に強力な手段である。

  形式仕様を記述する途中においても、また、その仕様記述を説明する自然言語文書を作成する途中においても、仕様とその実行結果を得ることは基本動作として重要である。

- 形式表現に対してテストを実行し成否を得る。

  関連する相互作用：20

  形式仕様を導入した成功事例では、いずれも効果的なテストが実施されていた。形式仕様からのテストケースの作成と実装に対するテストだけでなく、形式仕様に対するテストも広く行われていた。

形式仕様の意味を理解するために、形式的記述から別の形式的記述を導出し派生として利用することはしばしば有効であり、本調査の個別分析対象となったプロジェクトにおいても形式手法の根幹として多用されていた。以上の導出作業に関わる基本動作では、文書の記述、追跡、比較において、用語の語彙や識別子による文書間の関連性や派生による文書間の関連性を把握し開発者に提示することが重要である。また、形式言語表現間の派生である証明や実行結果、テスト結果なども、形式仕様の理解においても重要な役割を果たしている。

6.2.4 ツールが持つべき基本機能

前述の基本動作はそのほとんどが何らかのツール上で行われており、したがって、それらの基本動作を行う作業者をツールが機能を提供して支援することが有効であると考えられる。具体的には、用語や識別子による関連性や派生関係によって文書間を結びつける作業を、比較的大規模な仕様記述の量に対しても効率よく行うことができるよう支援することが必要だと考えられる。そこでそれらの機能の実現としては、用語や識別子を使って記述を索引化し、また、バージョン管理や構成管理と連携して開発者を支援することが考えられる。

(1) 語彙管理

前述の通り、用語の語彙および識別子は自然言語表現および形式仕様の両方において、その対応関係の把握や記述内容の比較や記述の読解に重要な役割を果たす。そこで、用語および識別子を管理し参照するための一連の機能が必要となる。

- 用語シソーラス定義

  自然言語処理で広く利用されているシソーラスの形式として WordNet がある。例えば WordNet における上位概念-下位概念関係と全体-部分関係を持つシソーラスで用語を管理することが考えられる。その場合、UML クラス図における is-a 関係や has-a 関係と対応するため、クラス図を使った要件の整理にも利用することができる。

- 用語シソーラス検索

  記述中の用語からシソーラス上の関連用語を把握することで、その記述の意味の広がりを把握することができる。

- 用語による索引化

  用語シソーラス上の語彙を使って、自然言語表現および形式仕様中の自然言語要素における用語の出現を索引化する。これによって、自然言語表現間および形式仕様間の用語に関する相互参照だけでなく、自然言語表現と形式仕様中の自然言語要素の間の相互参照が可能になる。

- 識別子による索引化

  形式仕様および自然言語表現中の形式言語要素における識別子の出現を索引化する。これによって、形式仕様間および自然言語表現間の識別子に関する相互参照だけでなく、形式仕様と自然言語表現中の形式言語要素の間の相互参照が可能になる。

- 用語シソーラスと識別子の連携

  それぞれの識別子中の用語を索引化することで、ある用語を含む識別子の一覧およびその逆参照を得ることができる。例えば自然言語表現から形式手法を記述する際には、自然言語表現中のある用語について、その用語を含む識別子の一覧を得ることで記述者を支援することができる。また逆に、形式仕様中の識別子から、その構成要素となっている用語について上位概念や下位概念を通して関連があると考えられる識別子の一覧を得ることができる。

(2) テキスト分析

- 形式言語の構文解析

  形式仕様に関して識別子などを抽出するために、形式言語のパーサが必要である。

- 自然言語中の単語への分解

  自然言語から用語を抽出する必要がある。さらに形態素解析などによって自然言語表現から用語候補を抽出できることが望ましい。

- 記述からの参照

  記述中に出現する用語および識別子の一覧、記述から派生元の記述への追跡および記述から派生先の記述への追跡が行えることが必要である。

- 記述の突合せ

  一方の記述中の用語の他方の記述中での出現位置の表示

  例えば、形式仕様を修正してその修正を自然言語による説明に反映させる時には、その修正箇所で用いられた用語や識別子の自然言語説明での位置を把握することで、修正すべき箇所の目安とすることができる。

- 記述間の用語の差異の表示

  例えば、形式仕様を修正してその修正を自然言語による説明に反映させる時には、その修正で追加した部分で用いられた用語が自然言語表現で説明されているかどうかを知る目安となる。また、その修正で削除した結果、形式仕様では直接関係なくなった概念についての説明が残っていないかを知る手がかりとなる。

- 一方の記述中の識別子の他方の記述中での出現位置の表示

  例えば、形式仕様を修正してその修正を自然言語による説明に反映させる時には、その修正で追加した部分で用いられた識別子が自然言語表現で説明されているかどうかを知る目安となる。また、その修正で削除した結果、形式仕様では直接関係なくなった概念についての説明が残っていないかを知る手がかりとなる。

(3) バージョン管理、構成管理

- バージョン管理

  記述された形式仕様はレビューや設計／実装やテストの結果からのフィードバックを得て改訂していく。そこである程度の規模になったらバージョン管理が必要になる。この時、索引や派生追跡についてもバージョンごとに管理できることが望ましい。

- 構成管理

  ある程度の規模を超えると文書に関する構成管理が必要になる。この時、構成の変更が各索引に反映されることが望ましい。

以上の基本機能を持つツールによって、日本語による記述と日本語識別子を用いた形式仕様に対して、並立や内包を支援し、整合性を持った記述を継続することができ、ひいては誤解の挿入を削減する技術的解決策の１つとなり得ると考える。

7. 考察と結び

7.1 明らかになったこと

開発プロジェクトにおける「仕様書の厳密化」が大切であることは、多くの関係者にとって共通の認識であることは間違いがない。

しかし「厳密な仕様書」が、実際の開発プロジェクトの大きな流れの中でどのような位置付けを占め、それが関連する成果物にどのような影響を与えているのかに関しては、まだ十分な知見や合意が得られているとはいえない状況である。

今回の調査を通し、形式手法を用いて成功したプロジェクトの内部で、

- 仕様書がどのように扱われ、その厳密さはどのように担保されていたのか
- 形式仕様とそれ以外の記法（自然言語による文章、各種の図や表、その他）が、どのような役割分担を果たしていたのか

といった事柄が明らかになった。

どのプロジェクトにも共通していたのが、「厳密な仕様」をプロジェクト全体から参照される辞書あるいは情報のハブとして用いているということである。

そして厳密な仕様の取りまとめに、形式手法を用いることにより、検証済の記述単位と記述範囲をはっきりとさせることが可能となっていた。このため、仕様の一部を形式手法以外の表現手段（日本語による文章、様々な図や表、その他）として抜き出して、議論したりレビューしたりという作業が遥かに容易になっていた。

もし厳密な記述の基礎に、形式手法による単位と範囲の下支えがなかったならば、目の前の記述（日本語による文章や、様々な概念図や表、その他）がどの範囲のものをどの単位で扱っているかが認識する個々人のなかで異なるものとなり、それがいわゆる「仕様の間違い」を引き起こすことになっていたことが予想される。

7.2 厳密な仕様記述を導入するために

最後に「日本語を用いた厳密な仕様記述」を導入するための提言を示す。

（1）厳密な仕様をプロジェクト情報のハブとする

厳密な仕様記述が「部分最適化のための個々の記述の技術の話」として捉えられている間は、大きな広がりは望めないと思われる。

今回の調査結果からも示されたように、形式仕様を核にしながら、連携するその他の記述（日本語の文章や注釈、様々な図や表など）をうまく併用し「厳密な仕様記述」を開発プロジェクトの礎におくことは、十分に機能するやり方である。

特に日本語識別子を使った形式仕様記述言語の採用は、問題領域の用語と対応付けも行いやすく、様々な日本語による文章とのクロスチェックの役に立つので強力な道具となり得る。

形式仕様記述言語による情報のハブの周りに、形式記述ではないものも含む様々な情報が循環する生態系（いわゆるエコシステム）を構築することが厳密な仕様記述を導入し最大に活用するための鍵である。

（２）複数の手法を組み合わせる

厳密な仕様記述を情報のハブとしておくことによって、そこを中心とする様々な情報の出し入れを行うことができるようになるが、単純に自然言語による説明資料を導出するだけに留まるのではなく、他の厳密な手法を組み合わせて、全体の情報の品質と流れを円滑にすることを探るべきである。

例えば設計のために構造を導出し、仕様上の性質と設計上の性質との対応関係を追跡し、更に詳細な文書化を行うといった応用や、設計モデル検査のための条件式を提供するという応用などは、調査事例の中でも見られたものである。

（３）適切なツールを選択／調整／作成する

厳密な仕様記述の形式仕様部分の処理そのものは、対応する言語処理系で支援されているが、情報のハブとして最大に活用しようと考える場合には、既存のツールが提供する機能だけに頼ることはできない。

そもそもプロジェクトの事情は個別に異なるため、どうしても情報の間の連携をとるための個別の工夫が必要となってくる（将来こうした工夫をうまく支援してくれる使い勝手の良い基盤が生み出されることを期待する）。

このため情報を検索、抽出、加工、検証、更新するための適切な道具立てを用意する必要がある。UNIX が提示したツールボックスの概念は今も有効で、情報を様々な道具を組み合わせて相互に活用する環境が大切である。厳密な仕様記述の世界でもこうした組合せを工夫することが有効である。

情報のなお一層の有効活用を行うためには、形式仕様記述のような情報と、非定型の文章や概念を共有する情報をうまく対応づけることのできる検索の仕掛けが必要であるが、現在のところ、そうした道具は、仕様記述テキストに対する正規表現を使った検索や MS Word マクロ機能といったレベルに留まっている。

（４）積極的に事例を開示する

個々の記述をどのように行ったかを議論することも大切だが、それらの記述がプロジェクトの中でどのような位置付けを占めていたのかに関する情報も積極的に外部に開示する。

このことによって、知恵を社会的にオープンなものとし、また自身の工夫に対するフィードバックを得られるようにすることが理想である。

また作成された厳密な仕様に関わる成果物も、個々のアプリケーションの機密に関わる部分以外のものは、可能な限り公開されることが望ましい。

こうすることによって、外部のリソースも巻き込んだ社会的エコシステムができあがることを期待できる。

7.3 結び

本書では、上流工程における仕様書作成に形式手法を用いて成功した国内外の事例をヒアリングにより、仕様書作成に係る諸問題の根本原因が何か、それを形式手法の活用によりどう解決したかを調査した結果を報告した。ヒアリング結果の分析を通して、(1)形式手法を上流工程における記述法として導入する際の技術的解決策の抽出、および(2)日本語による仕様の記述において誤解の挿入を少なくする技術的解決策の抽出を実施することができた。

Appendix

Appendix 1. ヒアリングにあたり準備した質問と回答肢

Appendix 1.1 作業項目 1 に関する質問と回答肢

(a) 当初の問題意識と形式手法導入のきっかけ

Q1-1 **形式手法を導入した問題意識を教えてください**

A. 開発組織の生産性向上 B. 法令/規格からの要請 C. ドメイン特有の要求
D. ステークホルダからの提案 E. その他

Q1-2 従来は上流工程でのバグ発見の工夫として、どのような取り組みをしていましたか？

A. 形式手法を以前から適用していた B. ソフトウェア工学的な方法論
C. 要求工学的な手法 D. 用語辞書やフォームによる半形式化 E. その他

Q1-3 従来は上流工程でのバグ挿入を防ぐ工夫として、どのような取り組みをしていましたか？

A. レビュー B. ウォークスルーによるシミュレーション
C. プロトタイプによる検証 D. 非形式的証明
E. 形式手法(証明) F. 形式手法(モデル検査) G. その他

Q1-4 従来はどのような方法で仕様をレビューしていましたか？

A. 内部レビュー B. 第三者によるレビュー C. その他

Q1-5 形式手法を知った、採用の検討をした、採用を決断した、直接的なきっかけを教えてください。論文や記事、イベント、打ち合わせ等

A. 論文 B. 雑誌記事 C. 学術会議 D. セミナー E. イベント
F. 同僚との打ち合わせ G. 外部の人との打ち合わせ H. 開発プロジェクト
I.その他

(b) 事例の概観、プロジェクト構成

Q1-6 プロジェクトのどの要素のために形式手法を導入しましたか？

A. 対象ドメイン B. 規模(コードサイズ) C. 規模(運用時の構成)
D. 規模(ステークホルダ数) E. 規模(予算) F. 安全性
G. 確実なサービスイン H. 維持 I. 安全性 J. その他

Q1-7 ステークホルダの構成を教えてください。 [新規]

A. 開発組織 B. 顧客 C. 顧客以外のエンドユーザ D. 開発組織外アーキテクト
E. 開発組織外分析者 F. 開発組織外仕様記述者 G. 開発組織外設計者
H. 開発組織外実装者 I. 外部査察者 J. その他

(c) プロセス

Q1-8 開発プロセスの全体像と、そのうち形式手法を適用したフェーズを教えてください。

A. ドメイン分析 B. 業務分析 C. 要求分析 D. 仕様定義 E. 方式設計

F. 概要設計 G. 詳細設計 H. 実装 I. 単体テスト J. 機能テスト
K. 統合テスト L. 受け入れテスト M. 運用 N. 維持 O. その他

Q1-9 形式手法を適用したフェーズそれぞれで、どのような形式手法の作業項目を実践しましたか？

A. 仕様を断片的に形式的記法で記述する
B. 一部仕様を体系的に形式的記法で記述する
C. 仕様全体を体系的に形式的記法で記述する
D. 形式的に記述された仕様に対して構文検査を行う
E. 形式的に記述された仕様に対して実行可能な部分をアドホックに実行して仕様への理解を得る
F. 形式的に記述された仕様に対して単体テストを記述・実施する
G. 形式的に記述された仕様に対して機能テストを記述・実施する
H. 形式的に記述された仕様に対してモデル検査を実施する
I. 形式的に記述された仕様に対してアドホックに証明課題を設定し、証明する
J. 形式的に記述された仕様に対して体系的に証明課題を設定し、証明する
K. その他

Q1-10 実施した形式手法の作業項目について、開発当初から実施していましたか？

A. 開発当初から、利益とコストを考慮した上で実施した
B. 開発当初から、実施可能と判断した上で実施した
C. 開発当初の計画として、利益とコストを考慮した上で開発途中からの実施を計画していた
D. 開発当初の計画として、実施可能と判断した上で開発途中からの実施を計画していた
E. 開発当初は計画していなかったが、開発途中から利益とコストを考慮した上で実施した
F. 開発当初は計画していなかったが、開発途中から実施可能と判断した上で実施した
G. その他

Q1-11 実施した形式手法の作業項目について、開発完了まで実施しましたか？

A. 開発完了まで実施した
B. 保守等で、現在も実施継続中である
C. 開発途中から利益とコストを考慮した上で中止した
D. 開発途中から実施可能と判断した上で中止した
E. その他

Q1-12 形式手法を適用したフェーズそれぞれで、どのような工夫をしましたか？

A. 各ステークホルダから最低一人は形式手法を学習させた
B. 自然言語による注釈を促す具体的な方策を取った
C. 自然言語による仕様記述を併用した
D. ダイアグラム等による形式化された仕様記述を併用した
E. 擬似コード等による記述を併用した
F. 形式手法専用のツールを導入した
G. 既存のツールで形式仕様を扱えるように拡張した
H. その他

Q1-13 仕様記述の生産性に変化はありましたか？

A. 劇的に向上した

B. 全体として向上した
C. 一部に向上がみられた
D. 特に変化はなかった
E. 一部に低下がみられた
F. 全体として低下した
G. 非常に低下した
H. その他

Q1-14 仕様記述の検証性について、仕様の誤りの検出に関して変化はありましたか？特に仕様記述フェーズ内での検証性の変化はありましたか？
A. 劇的に向上した
B. 全体として向上した
C. 一部に向上がみられた
D. 特に変化はなかった
E. 一部に低下がみられた
F. 全体として低下した
G. 非常に低下した
H. その他

Q1-15 仕様記述の伝達性について、仕様に関する誤読・誤解に関して変化はありましたか？
A. 劇的に向上した
B. 全体として向上した
C. 一部に向上がみられた
D. 特に変化はなかった
E. 一部に低下がみられた
F. 全体として低下した
G. 非常に低下した
H. その他

Q1-16 フェーズをまたがった仕様記述の伝達効率と品質はどのように向上しましたか？
A. 劇的に向上した
B. 全体として向上した
C. 一部に向上がみられた
D. 特に変化はなかった
E. 一部に低下がみられた
F. 全体として低下した
G. 非常に低下した
H. その他

Q1-17 形式的仕様記述の伝達効率と品質を上げるためにどのような工夫をしましたか？
A. 各ステークホルダから最低一人は形式手法を学習させた
B. 自然言語による注釈を促す具体的な方策を取った
C. 自然言語による仕様記述を併用した
D. ダイアグラム等による形式化された仕様記述を併用した
E. 擬似コード等による記述を併用した
F. 形式手法専用のツールを導入した
G. 既存のツールで形式仕様を扱えるように拡張した

H. その他

Q1-18 形式手法のトレーニングを受けなかったステークホルダとの間では、どのようにしてコミュニケーションを取りましたか？
A. 形式仕様中のコメントや識別子を参照してのコミュニケーション
B. 自然言語による仕様を参照してのコミュニケーション
C. ダイアグラム等による形式化された図によるコミュニケーション
D. 擬似コード等による記述を使ったコミュニケーション
E. 形式手法を理解した者に通訳的な役割をもたせた
F. その他

Q1-19 形式仕様と自然言語によるコミュニケーションが混在した際に、工夫したことはありますか？
A. 形式仕様中のコメントや識別子を参照してのコミュニケーション
B. 自然言語による仕様を参照してのコミュニケーション
C. 形式手法を理解した者に通訳的な役割をもたせた
D. その他

Q1-20 形式手法を適用しなかったフェーズについて、何か工夫をしましたか？
A. 各ステークホルダから最低一人は形式手法を学習させた
B. 自然言語による注釈を促す具体的な方策を取った
C. 自然言語による仕様記述を併用した
D. 形式手法を理解した者に通訳的な役割をもたせた
E. その他

Q1-21 従来のレビューとの相違点について、まずは個人的な感想をお聞かせください。

Q1-22 従来のレビューと比較して、発生する間違いの種類と量に変化はありましたか？
A. 間違いの種類が変化した。 B. 間違いの種類は変化しなかった
C. 間違いが増えた D. 間違いの量は変化なし E. 間違いが減った
F. その他の変化があった

Q1-23 特に向上した品質があったら教えてください。
A. 仕様の生産性 B. 仕様の検証性 C. 仕様の可読性 D. 仕様の厳密性
E. 仕様の誤り率 F. 仕様の信頼性 G. 仕様の可塑性 H. その他

Q1-24 従来のレビューと比較して、記述量、修正量に違いはありましたか？
A. 記述量が増えた B. 記述量に変化なし C. 記述量が減った
D. 修正量が増えた E. 修正量に変化なし F. 修正量が減った
G. その他変化があった

Q1-25 従来手法と比較して、修正の追跡性、追従性に違いはありましたか？
A. 追跡性が向上した B. 追跡性に変化なし C. 追跡性が低下した
D. 追従性が向上した E. 追従性に変化なし F. 追従性が低下した
G. その他変化があった

Q1-26 形式仕様の導入により、下流工程以降で必要のなくなった工程はありましたか？
A. 方式設計で必要のなくなった工程があった
B. 概要設計で必要のなくなった工程があった

C. 詳細設計で必要のなくなった工程があった
D. 実装で必要のなくなった工程があった
E. 試験で必要のなくなった工程があった
F. 運用で必要のなくなった工程があった
G. 維持で必要のなくなった工程があった
H. その他必要のなくなった工程があっ

Q1-27 形式仕様の導入により、下流工程以降で実施が困難になった工程はありましたか？
A. 方式設計で困難になった工程があった
B. 概要設計で困難になった工程があった
C. 詳細設計で困難になった工程があった
D. 実装で困難になった工程があった
E. 試験で困難になった工程があった
F. 運用で困難になった工程があった
G. 維持で困難になった工程があった
H. その他困難になった工程があった

Q1-28 形式仕様の導入により、下流工程以降で新たに必要になった工程はありましたか？
A. 方式設計で必要になった工程があった
B. 概要設計で必要になった工程があった
C. 詳細設計で必要になった工程があった
D. 実装で必要になった工程があった
E. 試験で必要になった工程があった
F. 運用で必要になった工程があった
G. 維持で必要になった工程があった
H. その他必要になった工程があった

Q1-29 形式仕様の導入により、下流工程以降で実施が容易になった工程はありましたか？
A. 方式設計で容易になった工程があった
B. 概要設計で容易になった工程があった
C. 詳細設計で容易になった工程があった
D. 実装で容易になった工程があった
E. 試験で容易になった工程があった
F. 運用で容易になった工程があった
G. 維持で容易になった工程があった
H. その他容易になった工程があった

(d) ツール/言語

Q1-30 仕様記述言語として何を選びましたか？
A. VDM B. Z 記法 C. その他

Q1-31 ツールとして何を導入しましたか？
A. エディタ
B. 構文チェッカ
C. 証明支援器
D. モデル検査器
E. コード生成器
F. その他

Q1-32 それらの言語やツールを採用した理由を教えてください。
A. 習得の容易さ B. 効果の高さ C. プロジェクトへの適合性
D. 事例を参考にして E. 専門家による推薦
F. その他

Q1-33 ツール/言語について、良かった点と問題になった点を挙げてください。
A. 習得の容易さ B. 効果の高さ C. プロジェクトへの適合性
D. その他

(e) 人材育成および教育

Q1-34 形式手法を適用するにあたり、誰を対象に教育をしましたか？
A. 開発組織 B. 顧客 C. 顧客以外のエンドユーザ D. 開発組織外アーキテクト
E. 開発組織外分析者 F. 開発組織外仕様記述者 G. 開発組織外設計者
H. 開発組織外実装者 I. 外部査察者 J. その他

Q1-35 カリキュラムの内容について教えてください。
A. 言語教育 B. モデリング技法 C. 例題を使った実習
D. 初期集合教育 E. 中途教育 F. フォローアップ教育
G. その他

Q1-36 教育はどのような組織形態で行われましたか？
A. 外部の教育チームに委託 B. 内部の専門チームに委託
C. 開発サイトごとに配置 D. ステークホルダごとの判断
E. その他

Q1-37 教育リソース
A. セミナー B. 書籍 C. 専従メンター D. 掛け持ちメンター E. その他

Q1-38 教育の効果について、
A. ツール習熟 B. 記述 C. 読解 D. 検証 E. 発想力
F. コミュニケーション能力 G. その他

Q1-39 教育に対する動機付け
A. 全員受諾した B. 非協力なステークホルダがいた
C. 拒否の姿勢を示したステークホルダがいた
D. 教育から離脱したステークホルダがいた
E. その他

(f) プロジェクト管理

Q1-40 プロジェクト全体について個人的感想をお聞かせください
A. 形式手法の導入に満足した
B. 形式手法の導入を後悔している
C. 形式手法の効果は全体に大きな影響を与えた
D. 形式手法の効果は部分的だが重大な影響を与えた
E. 形式手法の効果は部分的であり成功の要因は別にあった
F. その他

Q1-41 全体の可視化について
A. 可視化が向上した
B. 可視化に変化なし
C. 可視化が低下した
D. その他

Q1-42 個別プロセスの管理方法
A. プロセスの管理に形式手法独特の工夫をした
B. 従来通りのプロセス管理方法を実施した
C. その他

Q1-43 品質を担保するための工夫
A. 品質面を担保するために形式手法独特の工夫をした
B. 品質面を担保するために形式手法以外の工夫をした
C. その他

Q1-44 メトリクスへの工夫
A. メトリクスに形式手法独特の工夫をした
B. 従来通りのメトリクス計測を実施した
C. その他

Q1-45 形式手法の直接的適用とは別の工夫があったか
A. 形式手法を適用するために、形式手法以外の工夫をした
B. 形式手法の適用とは関係なく、形式手法以外の工夫をした
C. 形式手法以外の工夫は特に実施しなかった
D. その他

Q1-46 形式仕様の導入により、仕様の修正量から QCD に変化がありましたか？
A. Quality の面で効果があった
B. Cost の面で効果があった
C. Delivery の面で効果があった
D. QCD のバランスコントロールに効果があった
E. その他

(g) 意識改革

Q1-47 各ステークホルダにもたらされた意識の変化
A. 形式手法全体に対する評価が変わった
B. 形式手法の実用性に対する評価が変わった
C. 形式手法の効果に対する評価が変わった
D. 形式手法のコストに対する評価が変わった
E. 形式手法の意義に対する意識が変わった
F. 形式手法の実践方法に対する意識が変わった
G. その他

Appendix 1.2 作業項目 2 に関する質問と回答肢

(a) 日本語の工夫

Q2-1 日本語表現に対して、どのようなルール化を行いましたか？

A. 語彙の制限
B. 構文の制限
C. 形式仕様のコメントやアノテーションからツールにより自動抽出
D. 形式仕様との対応関係の管理
E. その他

Q2-2 上記ルールの適用対象とした文書の種類を教えてください。

A. 分析チームが内部で用いる非公式文書
B. 分析チームが他チームに公開した主成果物
C. 分析チームが他チームに公開した補足文書
D. 要求定義チームが内部で用いる非公式文書
E. 要求定義チームが他チームに公開した主成果物
F. 要求定義チームが他チームに公開した補足文書
G. その他

Q2-3 上記ルールについて訓練をしましたか？その場合の訓練対象や訓練の形態、期間について教えてください。

A. スクール形式の初期導入トレーニング
B. スクール形式のフォローアップトレーニング
C. オンジョブトレーニング
D. レビュー
E. その他

Q2-4 上記ルールは上流工程での記述の生産性および上流工程でのステークホルダ間の伝達性にどのような影響を与えましたか？

A. ステークホルダ間の伝達性および生産性が劇的に向上した
B. ステークホルダ間の伝達性および生産性が全体的に向上した
C. ステークホルダ間の伝達性および生産性が部分的に向上した
D. ステークホルダ間の伝達性および生産性に変化はなかった
E. ステークホルダ間の伝達性および生産性が低下した
F. その他

Q2-5 上記ルールは下流工程での記述の伝達性にどのような影響を与えましたか？

A. 可読性が向上して下流工程の生産性が向上した
B. 伝達性が向上して上流チームへの問合せが減少した
C. 伝達性が向上して誤読による誤り混入が減少した
D. 伝達性が向上して下流工程の生産性が向上した
E. その他

Q2-6 上記ルールの適用対象となった仕様記述について、下流工程での理解の誤りに起因する手戻りは発生しましたか？どのように対応しましたか？

A. 仕様の曖昧性による誤読により手戻りが発生した
B. 読み手のミスによる誤読により手戻りが発生した
C. ルールの不備による誤読により手戻りが発生した

D. その他

Q2-7 上記ルールの運用上の問題と問題の発生理由と（日本語表記としての）対応策があれば教えてください。

A. 記述に時間がかかり生産性が低下した
B. 読解に時間がかかり生産性が低下した
C. ルールの適用が不徹底だった
D. ルールへの誤解による誤読により生産性が低下した
E. 教育コストおよび作業量の変化の見積もりを誤った
F. その他

(b) 形式記述の工夫

Q2-8 既存の自然言語記述との連続性を考慮した工夫があれば教えてください。

A. 用語の統一
B. 記述間の関連性の管理
C. コメント、アノテーション、マークアップ等のルール
D. その他

Q2-9 導入時の問題点と対応

A. 教育コストがかかった
B. ルール徹底にコストがかかった
C. 習熟するまで生産性が一時的に低下した
D. その他

(c) 両者の比較（長所、欠点）

Q2-10 形式的要素を取り入れた日本語について、通常の日本語と比べどのような長所や欠点がありましたか？

A. 形式的要素を取り入れたことで曖昧性による誤解が減少した
B. 形式的要素を取り入れたことで習熟コストがかかった
C. その他

Q2-11 形式的要素を取り入れた日本語について、通常の形式仕様と比べどのような長所や短所がありましたか？

A. 日本語と組み合わせることによって可読性が向上した
B. 日本語と組み合わせることによって曖昧性による誤解が生じた
C. その他

Q2-12 形式的要素を取り入れた日本語について、導入時の問題点やその対応策について教えてください。

A. トレーニングのコストに問題があった
B. 習熟までのラーニングカーブに問題があった
C. ルール徹底の方法に問題があった
D. その他

(d) 今後の展開

Q2-13 形式手法を背景に持つ日本語仕様について、今後の見通しについてどうお考えですか？

A. 成功したルールのまま今後も適用したい
B. ルールを改良した上で今後も適用したい

C. 適用範囲を広げて今後も適用したい
D. 適用範囲を再検討して今後も適用したい
E. 効果があったので今後も適用したい
F. 今回は効果はなかったが再度適用したい
G. 今後は適用しない
H. その他

Q2-14 過去の経験を踏まえ、形式手法を背景に持つ日本語仕様について、適用性をどのように考えますか？
特に、ドメイン特化な部分と、一般に言える部分について教えてください。
A. ドメイン特有な利点
B. ドメイン特有な問題点
C. ドメイン独立な利点
D. ドメイン独立な問題点
E. その他

Q2-15 一般的な技術的反省と期待についてお話しください。
A. ルールの見直しまたは拡充
B. 訓練の見直しまたは改善
C. 形式手法を背景に持つツールの開発または改良
D. 日本語仕様を扱うツールの開発または改良
E. その他

Appendix 2. ヒアリング分析時の着目点

Appendix 2.1 作業項目１に対する着目点

作業項目１に関わるヒアリング項目をまとめるにあたって以下の分析項目に着目した。

(a) 当初の問題意識と形式手法導入のきっかけ

形式手法を導入した問題意識

従来での上流工程でのバグ発見の工夫

従来での上流工程でのバグ挿入を防ぐ工夫

従来での仕様レビューの方法

形式手法を知った、採用の検討をした、採用を決断した、直接的なきっかけ

(b) 事例の概観、プロジェクト構成

形式手法を導入した目的

ステークホルダの構成

(c) プロセス

開発プロセスの全体像と、そのうち形式手法を適用したフェーズ

形式手法の作業項目

実施した形式手法の作業項目について、開発当初から実施していたか

実施した形式手法の作業項目について、開発完了まで実施したか

形式手法を適用したフェーズそれぞれで、どのような工夫をしたか

仕様記述の生産性の変化

仕様記述の検証性

仕様記述の伝達性(誤読・誤解など)

フェーズをまたがった仕様記述の伝達効率と品質

形式的仕様記述の伝達効率と品質を上げるための工夫

形式手法のトレーニングを受けなかったステークホルダとのコミュニケーション

形式仕様と自然言語によるコミュニケーションが混在した際の工夫

形式手法を適用しなかったフェーズでの工夫

従来のレビューとの相違点

従来のレビューと比較しての間違いの種類と量の変化

特に向上した品質

従来のレビューと比較しての記述量、修正量の違い

従来手法と比較しての修正の追跡性、追従性の違い

形式仕様の導入により、下流工程以降で必要のなくなった工程

形式仕様の導入により、下流工程以降で実施が困難になった工程

形式仕様の導入により、下流工程以降で新たに必要になった工程

形式仕様の導入により、下流工程以降で実施が容易になった工程

(d) ツール／言語

仕様記述言語

ツール

言語やツールを採用した理由

ツール／言語の良かった点と問題になった点

(e) 人材育成および教育

教育対象

カリキュラム

教育の組織形態

教育リソース

教育の効果

教育の動機付け

(f) プロジェクト管理

プロジェクト全体の個人的感想

全体の可視化

個別プロセスの管理方法

品質を担保するための工夫

メトリクスへの工夫

形式手法の直接的適用とは別の工夫

仕様の修正量からの QCD の変化

(g) 意識改革

各ステークホルダの意識変化

Appendix 2.2 作業項目2に対する着目点

作業項目 2 に関わるヒアリング項目をまとめるにあたって以下の分析項目に着目した。

(a) 日本語の工夫

日本語表現に対するルール

上記ルールの適用対象とした文書の種類

上記ルールについての訓練

上記ルールの効果(上流工程の生産性、伝達性)

下流工程での記述の伝達性への影響

下流工程での理解の誤りに起因する手戻り

上記ルールの運用上の問題と問題の発生理由と対応策

(b) 形式記述の工夫

既存の自然言語記述との連続性を考慮した工夫

導入時の問題点と対応

(c) 両者の比較（長所、欠点）

通常の日本語と比較した長所や欠点

通常の形式仕様と比較した長所や短所

導入時の問題点やその対応策

(d) 今後の展開

形式手法を背景に持つ日本語仕様の今後の見通し

形式手法を背景に持つ日本語仕様の適用性

一般的な技術的反省と期待

Appendix 3. 参考文献

1. Spivey, J.M. : The Z notation: A Reference Manual, Prentice Hall, 1992
2. Hall,A. : Seven Myths of Formal Methods, IEEE Software, Vol. 7, Issue 5, pp. 11-19, Sep 1990
3. Bowen,J.P. and Hinchey,M.G. : Seven More Myths of Formal Methods, IEEE Software, Vol. 12, Issue 4, pp. 34-41, Jul 1995
4. King,S., Hammond,J., Chapman,R. and Pryor,A. : Is Proof More Cost-Effective than Testing?, IEEE Transactions on Software Engineering, Vol. 26, No. 8, Aug 2000
5. Chapman,R. and Dewar.R. : Re-engineering a safety-critical application using SPARK95 and GNORT, Reliable Software Technologies Ada Europ '99. Lectrue Notes in Computer Science, Vol. 1622, pp. 39-51, June 1999
6. Chapman,R. and Hall,A. : Correctness by Construction: Building a Commercial Secure Sytem, IEEE Software, Jan/Feb 2000.
7. Barnes,J., Chapman,R., Johnson,R., Widmaier,J., Cooper,D. and Everett.B.: Engineering the Tokeneer Enclave Protection Software, In. Proceeding of the IEEE International Symposium on Secure Software Engineering (ISSSE) 2006
8. White,N. : Formal Methods in Air Traffic Control, http://slideshare.net/AdaCore/white-open-do
9. Ross,P.E. : The Exterminators, IEEE SPECTRUM, Sept 2005
10. Ian O'Neill : SPARK – a language and tool-set for high-integrity software development, Industrial Use of Formal Methods – Formal Verification (Edited by Jean-Louis Boulanger), Wiley ISTE, 2012
11. Fitsgerald,J., Larsen,P.G., and Sahara,S. : VDMTools: Advances in support for formal modeling in VDM, SIGPLAN Notices 43,2, pp. 3-11, Feb 2008
12. Larsen, P. G. and Fitzgerald, J. : Recent industrial applications of VDM in Japan. In FACS 2007 Christmas Workshop: Formal Methods in Industry, 2007
13. Smith,P.R. and Larsen,P.G. : Applications of VDM in Banknote Processing. In John S. Fitzgerald and Peter Gorm Larsen, editors, VDM in Practice: Proc. First VDM Workshop 1999, Sep1999
14. M. van den Berg and M. Verhoef : Formal Specification of an Auctioning System using VDM++ and UML, In John S. Fitzgerald and Peter Gorm Larsen, editors, VDM in Practice: Proc. First VDM Workshop 1999, Sep1999
15. M. van den Berg, M. Verhoef and M. Wigmans : Formal Specification and Development of a Mission-critical Data Handling Subsystem, In John S. Fitzgerald and Peter Gorm Larsen, editors, VDM in Practice: Proc. First VDM Workshop 1999, Sep1999
16. 栗田太郎 : 形式手法の実践に対してよく尋ねられる質問とその回答--モバイル FeliCa の開発における形式仕様記述を通して, SEC journal, 7(1), pp. 34-39, Mar 2011

17. 栗田太郎 : モバイル FeliCa のソフトウェア開発における品質確保のための構造と実践--抽象度の制御やコミュニケーションの活性化に向けて, 情報処理学会デジタルプラクティス, 1(3), pp. 148-157, July 2010

18. 栗田太郎,荒木啓二郎 : モデル規範型形式手法 VDM と仕様記述言語 VDM++ : 高信頼性システムの開発に向けて, 日本信頼性学会誌 : 信頼性 31(6), pp. 394-403, Sep 2009

19. 栗田太郎 : 仕様書の記述力を鍛える – モバイル FeliCa 開発における形式仕様記述手法の導入事例, 日経エレクトロニクス, 2007 年 2 月号, pp. 133-152, Feb 2007

20. 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