特集
フォーマルメソッドの新潮流 3

Part II：産業界への応用

携帯電話組込み用モバイル FeliCa IC チップ開発における形式仕様記述手法の適用

栗田太郎 ● フェリカネットワークス（株）開発部 2 課

筆者らは，携帯電話組込み用モバイル FeliCa IC チップファームウェアの開発に，形式仕様記述手法を適用し，手法導入の目的である，（1）厳密な仕様の記述，（2）仕様の段階的な記述と検証を中心とした，開発スキーム，プロセス，フレームワークの検討と導入，（3）記述精度の向上とテストによる，開発の上流工程における品質の確保，（4）仕様を活用した徹底的なテスト，（5）コミュニケーションの活性化，を達成し，開発の成果を上げると同時に，手法適用の効果を確認した．

ソフトウェア開発における課題

ソフトウェア開発の現場では，曖昧な仕様に起因するトラブルが多く，課題となっている．

先行する工程での問題は，後続の工程に先送りされ，より大きな問題に発展していく．後続の工程における修正コストは，先行する工程よりも大きくなり，開発は不確実なものとなる．

開発者は，不注意や思い込みなどにより間違える．最終的に品質を確保するためには，誤りを修正していく必要がある．しかし，修正する間違いなのかどうかを判断するための仕様が明確でないと，設計，実装，テストを担当する技術者は，「何を」拠り所としてよいか分からない．

仕様の記述は，ソフトウェア開発の成否を決定する重要な活動である．「何を」作ろうとしているのか，「何を」作ったのかを，厳密に記述，メンテナンスすることにより，開発，運用，保守が正確に，精密に行える．そして，仕様開発の後続の工程，プロジェクトマネジメント，コミュニケーションを建設的に収束させることが可能となる．

本稿では，ソフトウェア開発プロジェクトに，厳密に仕様を記述することを目的として形式仕様記述手法を適用し，開発の成果を上げると同時に，手法適用の効果を確認した事例を報告する．

プロジェクトの概要

プロジェクトの開発対象は，「おサイフケータイ」として知られる携帯電話組込み用モバイル FeliCa IC チップのファームウェアである．

モバイル FeliCa は，電子マネーや公共交通機関の乗車券・定期券，クレジットカード，ドアの鍵，身分証などのサービスに応用されている，非接触 IC カードと携帯電話の技術が融合したシステムとマルチアプリケーションサービスの総称である．

システムは，図 -1 に示す，モバイル FeliCa IC チップを搭載する携帯電話機と，携帯電話網経由で接続する FeliCa サーバ，携帯電話機を IC カードとしてかざす FeliCa リーダ／ライタから構成される [6]．

モバイル FeliCa IC チップファームウェアは，セキュアファイルシステムを基本機能として，認証機能，データや通信路の暗復号化機能，電子マネーや乗車券などの複数のサービスが 1 つの IC チップ内に共存するためのファイアウォール機能，携帯電話に組み込むために必要な拡張機能を有している．

モバイル FeliCa サービスやシステムは，ユーザが安心して安全に利用できる社会基盤でなければならない．国内の大多数の携帯電話に組み込まれる IC チップの開発には，ユーザの生活を支えるセキュリティ機能を実現する責任が伴う．数多くのステークホルダとの仕様や運用に関する調整と，出荷時のバグをゼロにしていくためのデバッグや，セキュリティ強化を確実に行わなくてはならない．

プロジェクトの期間は 3 年 3 カ月で，所属するメンバは 50 ～ 60 名，メンバの平均年齢は約 30 歳であった．プロジェクトの開始当初は，形式仕様記述手法に関する知識や経験のあるメンバはいなかった．

ファームウェアが動作するモバイル FeliCa IC チップの製造と販売は，複数の半導体製造事業者が行うため，セキュア CPU とファームウェアの開発環境は複数種類

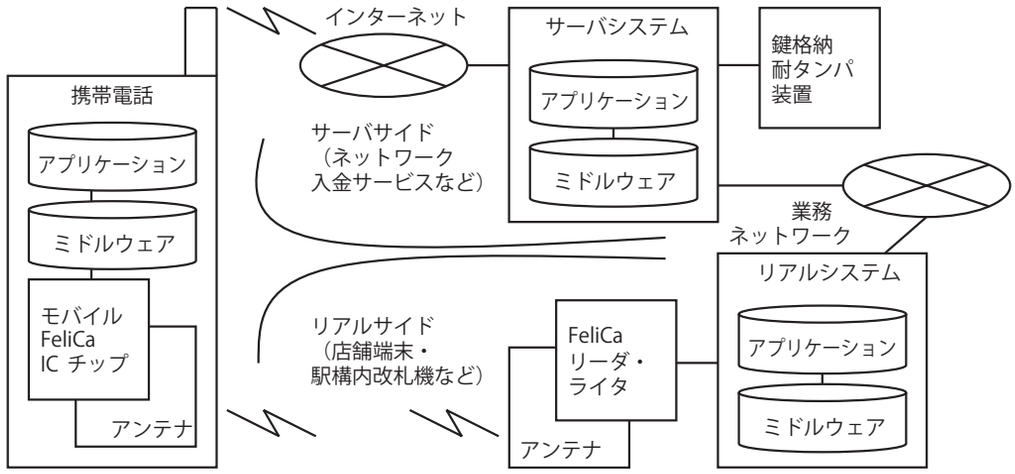

図 -1　モバイル FeliCa システムの構成

ある．これにより，携帯電話に搭載される IC チップの製造と販売にかかわるリスクを低減する．ファームウェアの開発にあたっては，それぞれの CPU 上で完全に同一の動作をするよう，互換性を確保しなければならない．

また，ファームウェアの実装に用いる言語は，C/C++，アセンブラ言語である．

目的

上記の責任を果たすべく，ソフトウェア開発における課題に対する対応策の 1 つとして，開発の上流工程における成果物の品質やコミュニケーションに着目し，厳密な仕様を形式的[1]に記述，検証することにした[5]．

そして，専門家の助言を受けながら，形式仕様記述手法導入の目的を，以下の通りとした．

(1) 厳密に仕様を記述し，機能を定義する
(2) 仕様の段階的な記述と検証を中心とした，開発スキーム，プロセス，フレームワークを検討，導入する
(3) 仕様を多方面からテスト，精査することにより，記述の精度を高め，開発の上流工程における品質を確保する
(4) 動作する仕様と，複数種類の，開発ボード上のファームウェア実装と，IC チップを，1 つのテスト環境と接続する．これにより，テスト仕様のテストと，仕様とファームウェア実装と IC チップが同等に動作することの確認を行う
(5) 形式仕様をコミュニケーションツールとして位置付け，ステークホルダとともにプロジェクトを推進する

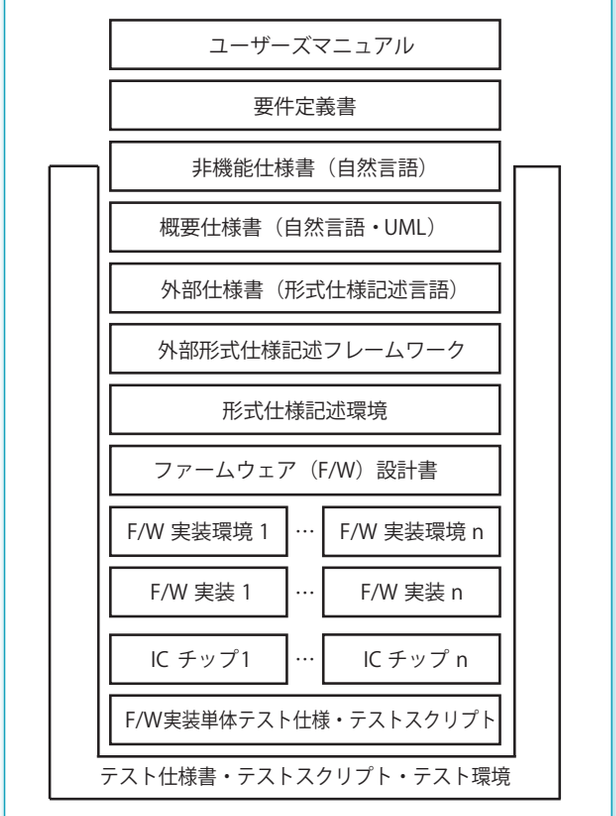

図 -2　開発成果物の構成

方法

●仕様の記述と検証

本プロジェクトにおいて記述，検証した文書，実装したプログラムコード，環境の構成を図 -2 に示す．

本プロジェクトでは，外部仕様書を形式的に記述した．

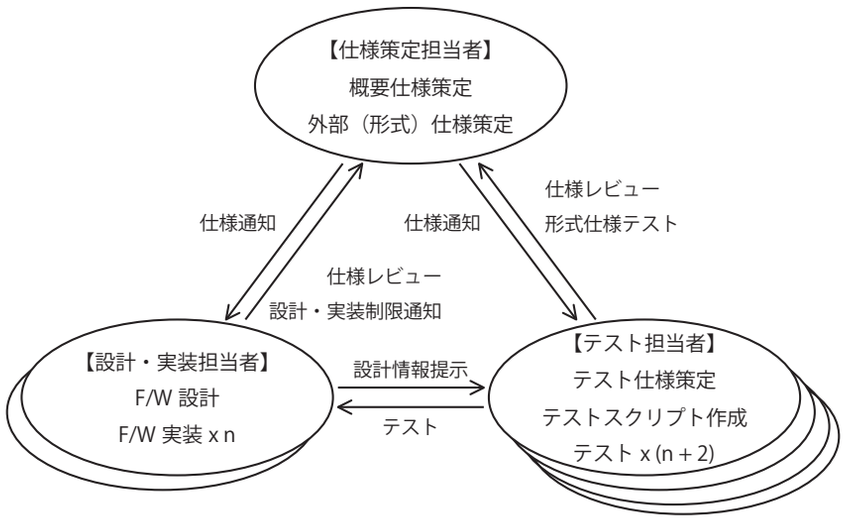

図 -3　チームの構成と分担

外部仕様は，動作する陽仕様として，クラスの単体テストを行いながら記述した．

仕様の記述手順は以下の通りである．

(1) プロジェクトのステークホルダとともに検討した要件と，自然言語や UML を用いて記述した概要仕様を基に，ファイルシステムのモデル化と，仕様を記述，検証するためのフレームワークの設計と実装を行う

(2) 基本的なプロトコルとセキュリティの仕様を記述しながら，ファイルシステムモデルとフレームワークを確定させる

(3) 仕様の段階的な詳細化とレビューを行い，プロトコル仕様とセキュリティ仕様を記述する

(4) 形式仕様の単体テストを行うための仕様を記述し，動作する仕様の検証を行う

外部仕様書の記述対象は，以下の通りである．

- データ構造を定義するファイルシステム仕様
- データ構造を基に，状態の振舞い仕様と，シーケンシャルな動作仕様を記述，検証するためのフレームワーク
- 仕様記述フレームワーク上に記述したプロトコル仕様
- セキュリティ仕様

なお，処理速度性能などの非機能仕様は，形式仕様とは独立して，自然言語を用いて概要仕様とともに記述した．

●言語とツール

形式仕様記述手法の導入に先立ち，専門家と議論を重ねた結果，形式仕様記述言語 VDM++[2] と，モデル分析，仕様記述，構文や型のチェック，単体テストを支援する統合開発環境 VDMTools[4] を導入することにした．

VDM++ は，ISO で標準化されている形式仕様記述言語 VDM-SL[3] に対して，主にオブジェクト指向の拡張を行った言語であり，モデリングから動作する仕様の記述まで広範囲をカバーする．

VDMTools は，多くの機能を備えているが，本プロジェクトでは，(1)仕様の構文チェック，(2)仕様の型チェック，(3)実行可能仕様の逐次実行とデバッグ支援，(4)実行可能仕様のコードカバレッジ計測，(5) 各種 CASE ツールとの連動，機能を用いた．特に，(1) ～ (4)の機能により，自然言語で記述した仕様では困難な，仕様のチェックやテスト，デバッグと，テストの網羅性確認を行うことができる．

なお，VDMTools は，実行可能な仕様を C++ 言語に変換する機能も有しているが，生成されるコードが組込み用途には適していなかったため，利用しなかった．

●チーム構成と開発プロセス

開発は，仕様策定担当チーム，設計・実装担当チーム，テスト担当チームに分かれて行った．各担当の分担を図 -3 に示す．

開発プロセスを図 -4 に示す．仕様策定，設計・実装，テストを行うサイクルを，イテレーティブに回した．1 回のサイクルを 1 ～ 2 週間としてプロジェクトを進めた．

●テストスキーム

開発全体のテストスキームを図 -5 に示す．

テスト担当者は，形式仕様を基にテスト仕様の策定(テスト項目の列挙)と，テストスクリプトの作成を行う．そして，テスト環境とテストスクリプトを用いて，動作

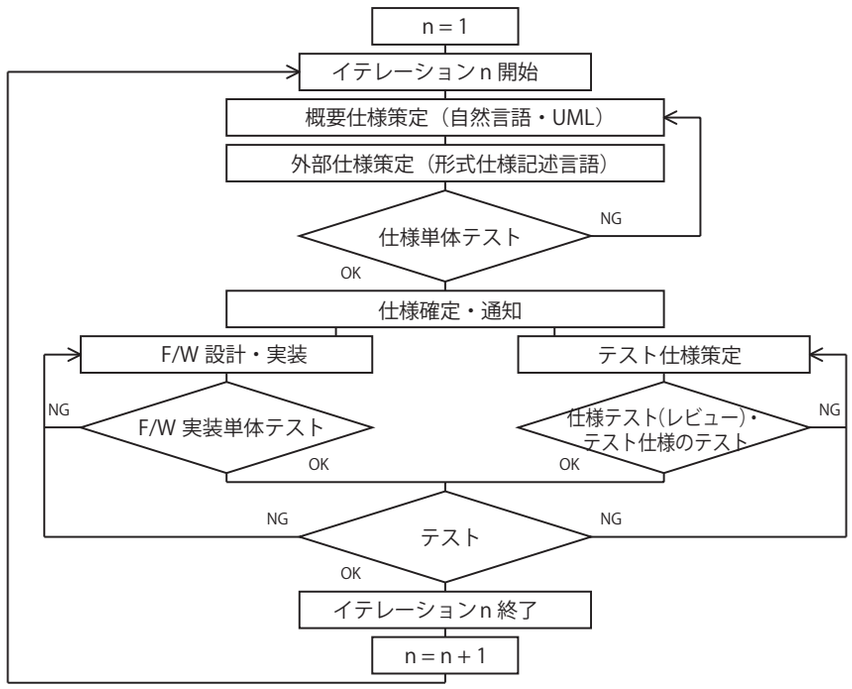

図 -4　開発のプロセス

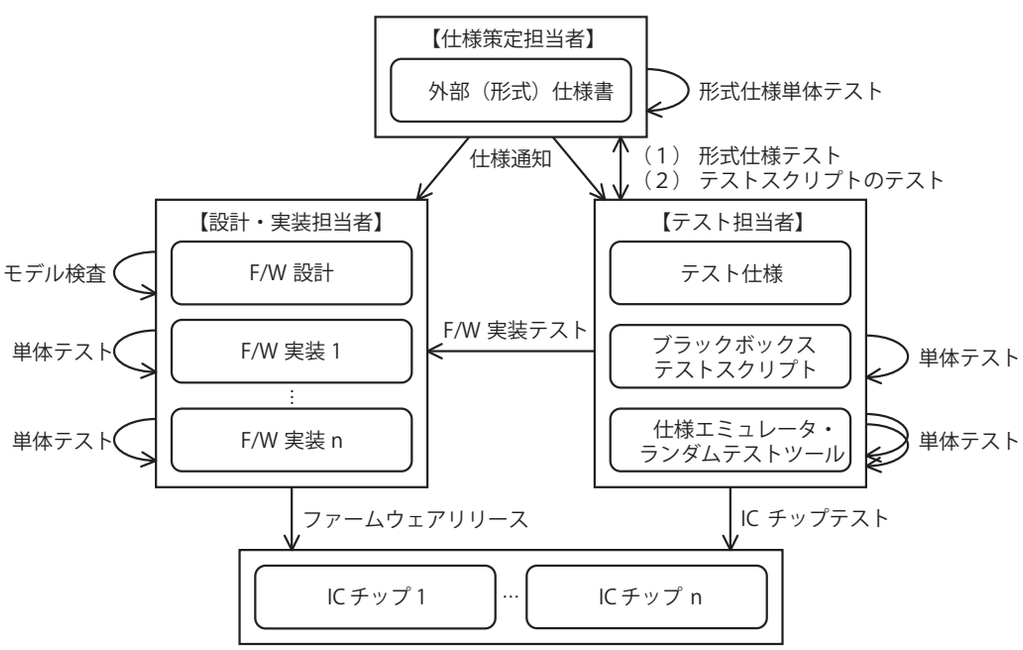

図 -5　テストのスキーム

する仕様と，複数種類の，開発ボード上のファームウェア実装と，IC チップのテストを行う．

テストの結果から，形式仕様通りのテスト仕様，スクリプトであることを確認するとともに，仕様の実行カバレッジを計測して，仕様の全範囲を網羅するテスト項目が列挙できていることを確認する．また，形式仕様と，複数種類のファームウェア実装と IC チップが，テスト環境からの視点において同等であることを確認する．

結果と考察

●仕様記述とファームウェア実装の成果

仕様に関連する主な成果物は，自然言語による 383 ページのプロトコル仕様書と，形式仕様記述言語による

不具合原因	割合
仕様記述もれ	0.2%
仕様記述誤り	0%
仕様不明確	1.8%
仕様見落とし	5.6%
仕様理解不足	10.7%
仕様確認不足	0%
仕様変更通知不徹底	0.2%
その他仕様関連外	81.5%

表 -1　不具合原因の割合

変更元	変更数
仕様策定担当者自身による変更依頼	84 件 （46%）
設計・実装担当者からの変更依頼・指摘・提案	25 件 （14%）
テスト担当者からの変更依頼・指摘・提案	23 件 （13%）
その他	9 件 （4%）
プロジェクト外からの変更依頼・提案	41 件 （23%）
合計	182 件

表 -2　仕様追加・変更・変更数

677 ページの外部仕様書である．プロトコル仕様書は，社内他部署や顧客に向けたマニュアルである．

形式仕様は，単体テストのためのコードと，自然言語によるコメントを含め，約 10 万行の規模であり，この仕様を基に，一種類の IC チップ上のファームウェアとして，C/C++ 言語でコメントを含め約 11 万ステップのコードを実装した．

●不具合原因の割合

形式仕様を基に設計，実装したファームウェアのデバッグにおいて，仕様に関係する不具合の割合は表 -1 の通りである．

仕様は誤りなく正確に記述できているといえる．形式仕様記述手法は，開発の初期段階における誤りの発見に有効である．構文や型のチェックが行える上，動作する仕様を策定することにより，仕様の単体テスト，テスト環境との接続を行うことができる．

一方で，「仕様見落とし」と「仕様理解不足」が合わせて 16.3% あった．これは，「何を」作るのかを表す仕様と，仕様を記述，動作，検証するためのフレームワークの分離が不十分であったことと，動作する仕様の「プログラミング」に気を取られ，「読ませる仕様」の記述ができなかったためである．

また，実装における不具合を機能別に分類すると，機能の重要度や複雑度によらず，分布が一様となった．要件の優先順位や仕様記述の難易度によらず，仕様記述の密度や品質が均一になったといえる．

●仕様の開発効率

仕様記述フレームワークの完成以降，1 人の仕様策定技術者が 1 カ月間（約 160 時間）に仕様策定した仕様の行数は，平均約 1,900 行であった．仕様策定業務は仕様記述だけではないため，単純に比較することはできないが，この数値は，本プロジェクトの実装やテストの工程における開発効率と同等である．形式仕様記述言語を用いることによる特段の混乱はないといえる．

記述密度を高くすることにより，記述の効率を上げることもできる．しかし，記述密度の高い仕様書が，プロジェクトメンバ全員にとって分かりやすい仕様であるとは限らない．また，記述密度を高くすると，開発の効率が下がる可能性もある．メンバの平均的な熟練度に合わせて，適切なレベルの記法やテクニックを用いた仕様を記述すべきであろう．

仕様策定技術者のスキルは，基礎的な情報処理技術の教育は受けているが，ソフトウェア開発業務経験年数には大きな開きがあり，中にはオブジェクト指向設計や，業務そのものの経験がないメンバもいた．しかし，ツールベンダが開催する講習会へ参加した効果もあり，導入への障害はなかった．また，途中から参加したメンバは講習会へ参加していないが，プロジェクト独自の研修や自習によりスムーズにチームに加わることができた．

●仕様の変更量

仕様書の版数が 1.0 になってから，仕様の追加や変更を 182 回行った．その内訳は表 -2 の通りである．

仕様開発は，仕様策定担当者による仕様記述や，ツールによる構文や型のチェック，単体テストだけでは完結できなかった．設計・実装担当者やテスト担当者，ステークホルダのレビューが不可欠である．

一方で，変更数が少なくない理由，特に仕様策定担当者自身による変更依頼が多い理由は，プログラムコードのみが検証対象ではなく，仕様も実行，検証が可能であるため，仕様もメンテナンス対象となり，仕様の改善に対する意識が高くなったためではないかと考えられる．

●仕様の単体テスト

仕様を形式的に記述することにより，ツールの助けを借りて，構文や型のチェックが行える．これにより，仕様の記述段階における誤りを修正できる．さらに，動作する仕様を策定することにより，仕様の単体テストを行うことができる．

形式仕様単体テストのラインカバレッジ率は 82% で

あった．仕様の不変条件，事前条件，事後条件において，ロジックに分岐があるものは，カバレッジ分析からテストケースの網羅率を高めることが可能となる．テストの結果，たとえば事後条件の不正なパターンを発見することができ，品質確保につながった．これは，レビューで発見することは困難な不整合であった．

●ブラックボックステスト

動作する形式仕様と，開発ボード上のファームウェア実装や IC チップのテストを行うテスト環境を接続した．

テスト担当者は，ブラックボックステストのために，約 7,000 のテスト項目とテストスクリプトを作成した．形式仕様のラインカバレッジ率は，目視による検査を加えて，100% である．

テスト仕様の策定と同時に，テスト担当者による仕様のレビューが行われることになる．形式仕様のレビューとテストの結果，仕様の不具合を 6 件検出した．これは，仕様策定段階における記述，レビュー，単体テストでは検出できなかった，仕様策定工程が次工程に先送りした不具合である．この不具合も，仕様の単体テスト結果と同様，仕様策定段階におけるレビューで発見することは困難な種類のものであった．構文や型のチェックに加えて，動作させる仕様をテストすることで，品質を向上させることができた．

また，ブラックボックステストにより，テスト仕様とテストスクリプトの妥当性と，仕様に対するテストの網羅率を確認することができる．動作する形式仕様のラインカバレッジ率が 100% ではない場合は，テスト項目が欠落している可能性がある．これにより，テスト項目の不足や，テストスクリプトの間違いが検出できる．一方で，テスト仕様が間違っているが，テストスクリプトが正しいケースは検出できない．概要仕様，マニュアルが間違っているが形式仕様は正しいケース，仕様，設計・実装，テストのすべてを思い込みにより間違えてしまうレアケースとあわせて，今後の検討課題である．

●ランダムテスト

ブラックボックステストに加えて，「ランダムテスト」を行った．ランダムテストとは，仕様のロジックを完全にエミュレートするテストツールが，動作する仕様や，開発ボード上のファームウェア実装，IC チップに対してランダムにコマンドを送信し，テスト対象から返却されるレスポンスが仕様通りの期待値であることを確認する耐久試験である．約 1 億パターンのテストとデバッグを行ったことで，品質の確保が可能となった．

ランダムテストツールや仕様エミュレータの開発には，厳密な内部状態の把握や精度の高い期待値の生成が必要となる．不変条件，事前条件，事後条件を明確に定義した形式仕様があるからこそ可能となるテストである．

●仕様とコミュニケーション

形式仕様を基にしたコミュニケーションの代表例として，設計・実装担当者，テスト担当者，プロジェクトのステークホルダからの質問とこれに対する回答の履歴を分析した．

仕様に関する質問は，「理解に関するもの」，「意図に関するもの」，「誤りの指摘」，に分類した．

特徴的だったのは，形式仕様は，自然言語で記述された文書と比較して，理解が正しいかどうかを確認する質問が多かった点である．たとえば，「仕様書に記載はあるものの，理解できなかったことにより発生した質問」は，自然言語の仕様書に対しては，理解に関する質問全体の 4% であったのに対し，形式仕様の場合は 21% であった．

形式仕様は中途半端な理解ができないため，仕様の理解に関する質問が多いといえる．さらに，仕様の背景までも理解しようとする開発者は，物事を突き詰めて考えようとするため，形式仕様まで読み込んでから質問をするともいえる．一方で，形式仕様は仕様の背景を理解するためのコメントや，導入教育が必要であるが，これが不足していたという課題もある．

形式仕様記述言語と比較して，自然言語で記述された文書に対しては，記述の明確化依頼が多かった．自然言語による記述は，意味が一意に定まらず曖昧になることがあり，厳密性に欠けやすい．

そのほか，形式仕様と自然言語仕様のそれぞれに対する質問の傾向は似通っていた．

●アンケート・インタビュー結果

本プロジェクトのメンバ全員に対して，形式仕様記述手法導入に関するアンケートを行った．アンケート結果の一部を**表 -3** にまとめる．さらに，仕様策定担当者と，形式仕様を頻繁に参照する設計・実装担当者，テスト担当者 12 人に個別に平均約 40 分のインタビューを行った．

上記のアンケート結果とインタビュー結果を簡単にまとめると，形式仕様記述手法の導入と適用に対して，総じて大きな拒否反応はなく，特にコミュニケーションツールとしての効果は実感しているが，導入当初は新規の学習が必要な上，手本や経験がないため戸惑ったことも多かったという意見が多かった．

仕様策定担当者　形式手法に対しておおむね好感を持っており，プロジェクトに対する適用効果に肯定的である．本プロジェクト内では仕様策定外の業務にも自信

質問項目	仕様策定担当者	設計・実装担当者	テスト担当者	全体
形式仕様を頻繁に参照	87%	13%	27%	24%
形式仕様を参照	13%	27%	46%	35%
仕様は読みやすい	29%	13%	12%	12%
慣れれば何とか読める	71%	53%	67%	59%
導入して大変よかった	43%	20%	8%	14%
よかった	57%	7%	69%	43%
効果はなかった	0%	14%	4%	6%
仕様記述言語は必要	87%	40%	81%	65%
今後も活用したい	0%	13%	19%	18%
案件により活用したい	100%	40%	69%	59%

表-3　アンケート結果

を持って臨むことができる．

設計・実装担当者　仕様は明確だが，どこまでが外部仕様で，どこからが外部仕様を動かすための形式仕様記述フレームワークのプログラムなのかが分かりづらい．また，実装に近い形式仕様の策定と，ファームウェア実装の違いをプロセスも含めて明確にしたい．

テスト担当者　仕様書が存在し，記述が明確であるため，自信を持ってテストを行い，仕様策定担当者，設計・実装担当者に指摘することができる．

結論

形式手法は，仕様策定工程における成果物の品質向上に効果がある．形式的な記述を行うことで曖昧さが取り除かれ，機械処理を始めとするさまざまな可能性が開ける．特に形式仕様記述言語とツールの利用により，仕様の記述が補助され，検証が可能になる．

厳密な仕様は，設計・実装やテストに活用できる．特に，「何を」作るのかを表した仕様は，品質確保のための活動の起点となる．また，形式手法は，プロジェクト内のコミュニケーションの活性化に寄与する．

さらに，動作する仕様がプログラムに近いことにより，プログラム開発において蓄積された技術が応用できる．バイナリ形式の仕様書では困難な変更管理，フィルタやバッチ，VDM++ であれば，オブジェクト指向分析，クラスのユニットテストなどの技術である．

仕様はプロジェクトメンバ全員が参照するものであり，メンバがストレスなく読むことができるシンプルな仕様である必要がある．実装用のプログラムコードよりも，規約に則った，読みやすい，読者が誤読することのない記述を心掛けると同時に，記述，検証とテストに応用できる仕様と，「読ませる」仕様のバランスを考慮しながら，記述を制御していくことが必要である．仕様が記述，管理され，全体が見えるようになることにより，開発者は安心して次の工程に進むことができるようになる．

何よりも重要なことは，「何を」，「どう」作るのかを分けて考え，記述，検証することである．このときにポイントになるのが，仕様を記述，検証するためのフレームワークである．フレームワークの適切な設計により，「何を」が明確になった，読み書きしやすい仕様が記述できる．

導入障壁

本プロジェクトにおける適用のレベルであれば，形式仕様記述技術者に求められる能力は，通常のプログラマに求められる抽象化能力を上回るものではない．

VDM++ 言語を用いた仕様記述は，C++/C#/Java 等の言語や，オブジェクト指向技術の知識，開発経験があれば，1 カ月程度のトレーニングで，仕様を読み書きできるようになる．

しかし，現在普及している一般的なプログラミング言語よりも導入の障壁は高い．言語仕様，ライブラリ，テンプレート，ツール，ノウハウの集約と整理が必要である．また，形式仕様記述手法の導入や仕様を策定するためのフレームワークの検討には，専門家の助言が必要である．

さらに，形式仕様記述手法の導入には，品質の確保と，プロジェクト内のコミュニケーションの活性化の効果があるが，導入以前は後工程で行っていたことを前倒しするため，仕様策定の期間は長くなる．また，形式仕様記述手法は仕様を「決める」助けにはならない．さらに，手法を利用しているからといって「良い」仕様であるとは限らない．

形式手法は，既存の手法に置き換わるものではなく，開発のドメインに応じて，他の手法と組み合わせて開発プロセスに組み込んでいく必要があることを，プロジェクトの関係者全員が理解しなければならない．

課題

今後の検討課題は以下の通りである．

- 仕様が，要求や要件を満たしていることの検証
- 形式仕様を読まないステークホルダとの，要件や仕様の調整
- ユーザ向けのマニュアルの検証
- ソフトウェア開発における記述全体の中で，形式的に記述，検証するスコープの設定
- 組込み系に適した形式仕様の策定
- 記述した仕様の妥当性確認とテスト．たとえば，セキュリティ仕様の脆弱性分析

- 読み書きしやすい仕様と，動作する仕様を両立させる仕様記述フレームワークの検討
- 仕様の記述密度と開発効率，ソフトウェア実装効率，読み手の理解度の，プロジェクト全体での最適化
- 設計・実装担当者やテスト担当者に喜ばれる「読ませる」，「身近な」仕様を，「気軽に」ブラウジング，検索できる環境の構築
- 「良い」仕様の追求

むすび

ソフトウェア開発におけるの課題の１つとして，ロジカルな思考や表現，コミュニケーションの難しさが挙げられる．ソフトウェアはロジックから構成されるが，開発者全員がロジックの追求を得意とするとは限らない．不注意や思い込みなどによりロジックを間違えたり，誤解したりすることもある．特に，歴史的な背景が蓄積された，規模の大きなソフトウェアの論理構造を取り出し，扱うことは容易ではない．

また，日本語などの自然言語を用いて，ロジックを表現，議論するためには，記述やコミュニケーションの高度な技術が必要になる．日本語を用いてロジカルに議論しようとすると，互いを傷つけ合ってしまうことがある．理詰めの言葉が人を傷つけ，時には人格が否定されているように感じさせることさえある．

チームやメンバでロジックだけを追及していこうとしても限界がある．ロジックをコンピュータが解釈できる言語で表現し，単純な確認をコンピュータに任せると同時に，ロジックの記述経緯を自然言語で叙述的に書き添えることが必要である．歴史を記録することにより，読み手は，なぜこのようなロジックになっているのかを理解することができるようになる．背景や意図が，論理に埋もれてしまわないことが重要なのである．

謝辞　形式仕様記述手法の導入にあたり，九州大学教授の荒木啓二郎氏，CSK システムズの佐原伸氏，デザイナーズ・デンの酒匂寛氏にお世話になりました．皆様のお力添えなくして，我々の手法導入は達成し得ませんでした．そして，プロジェクトへの適用は，メンバ全員の努力と協力がなければ実現できませんでした．厚くお礼申し上げます．

参考文献

1) 荒木啓二郎，張　漢明：プログラム仕様記述論，オーム社 (2002).
2) Fitzgerald, J., Larsen, P. G., Mukherjee, P., Plat, N., Verhoef, M. : Validated Designs for Object-oriented Systems, Springer (2005).
3) J・フィッツジェラルド，P・G・ラーセン：ソフトウェア開発のモデル化技法，岩波書店 (2003).
4) http://www.vdmtools.jp/
5) 栗田太郎：仕様書の記述スキルを鍛える，日経エレクトロニクス，2007 年 2 月 12 日号，pp.133-152，日経 BP (2007).
6) 杉山寛和，栗田太郎：携帯電話と FeliCa を融合したモバイル FeliCa 技術，情報処理，Vol.48, No.6, pp.561-566 (2007).

（平成 20 年 3 月 26 日受付）

栗田太郎
Taro.Kurita@FeliCaNetworks.co.jp
1971 年生まれ．1996 年から会社勤務，1999 年からソニー（株）．2004 年よりフェリカネットワークス(株)にてモバイル FeliCa の商用化および第 2 世代の開発に携わる．