ソフトウェア工学における形式手法
Formal Engineering Methods for Software Process

学生氏名：龍晨陽　LONGCHENYANG
指導教員： 亀田弘之

所属先： 東京工科大学 コンピュータサイエンス専攻　亀田研究室

キーワード：「形式化言語」，「ソフトウェア」，「Structured Object-Oriented Formal Language」

１． はじめに

1.1 背景

コンピューター科学の急速の発展により、ソフトウェアの種類は多くなり、様々なソフトウェアが使われている。そのため、ソフトウェアの品質の確保と、ソフトウェア信頼性を向上させることは、現在の研究の焦点になる。その中で、形式手法は数学に基づいて、関連する技術により、数学の厳密な理論と分析方法をソフトウェアの品質検証へ適用させる手法である。それはソフトウェアの開発に検証の役立つ方法と考えられる。形式手法は長年の発展を経て、技術自体は成熟になりつつあり、多くの研究結果は世の中に展示されていて、応用できるようになった。今まで、多くの形式手法が提案された。例えば、VDM、Z-Method、B-Method、Larch、OBJ など。その中で、よく使われるのは VDM、Z-Method、B-Method である。これらの形式手法をソフトウェア工学に取り組み続き、適用させることはソフトウェア信頼性を向上させる決め手と思われている。

1.2 問題点

開発コストが高い、プロジェクトの進捗がコントロールしにくい、品質保証ができない、メンテナンスを修正するなどの困難がある。

複雑さ：スケールの複雑さ、構造の複雑さ、環境の複雑さ、ドメインの複雑さ、コミュニケーションの複雑さ。

ソフトウェアの設計、開発、保守を指導、実施するための効果的な理論、方法、技術がないことと、ソフトウェアの設計、開発、保守に関する実践的な課題を指導、実施するための効果的な理論、方法、技術がないこと。

２．関連研究

2.1 SOFL 形式化言語の概要

SOFL は、英語では Structured Object-Oriented Formal Language と呼ばれた。

SOFL 形式的な言語では、要件文書の分析マニュアルを作成するための 3 つのステップを含み、迅速で視覚的で正確な要件文書の分析マニュアルが提供される。

SOFL はプロセス指向技術とオブジェクト指向技術を統合したシナリオを提供して、プロセス指向技術はソフトウェア開発の前に要件分析を行う際に強みを発揮し、顧客と開発者のつながりとコミュニケーションを確立し、プロセス指向開発言語とオブジェクト指向開発言語のメリットを活用できるようにしている。

20 年余りの発展を経て、だんだん工業業界のソフトウェア開発の需要に接近し、しかも伝統的な図形言語を関連する形式化の技術の中に溶け込み、独特な優位を持つ形式化の言語を形成している。

2.2 SOFL 形式化言語の基礎理論

(1)SOFL にはデータフローグラフが含まれる。

(2)状態データフローグラフとそれに関連するシステムモデルは階層構造で構成されて、形式的

な言語の安定性を高め、複雑さを低減するために用いられる。

(3)クラスは、複雑なデータストリームまたはデータウェアハウスを表すために使用できる。

SOFL 形式的な言語体系では、クラスとモデルは似た 2 つの概念であり、どちらも似た内部構造を含んでいる。しかし、それらを実際の言語に適用すると、それらは異なる役割を果たす。

３．提案手法

システム類の分類と設計は、Domain Development Driving の原則と設計に従って、全体システムを infrastructure（インフラストラクチャ層）、Domain（ドメインレイヤー）、Application（アプリケーション層）、Interfaces(インタフェース層)の四つの段階に分け、四つの段階の機能コードを詳細に分類したものである。

3.1 SOFL の応用実例の研究

この実験は、構造化されたオブジェクト指向の形式的な言語 SOFL を使用して、システムソフトウェアの機能とセキュリティ要件を実現する実用的な方法を示すことを目的としている。本文で示したケーススタディは、オンラインバンキングのアプリケーションです。登録口座、登録口座、資金照会、詳細照会などの機能を備えている。

４．おわりに

形式的な方法自身はコンピュータシステムの開発に用いられ、システムの性質を記述する数学に基づく技術である。

形式的な方法の利点は一貫性と完全性であり、仕様の実装と正確性が定義されているが、コストが高く、中小規模のプロジェクトでは効率性の向上が期待できないことから、次の二つのアイデアを紹介する。

コストについては、形式的な方法のコストは伝統的な方法に対するそれ自身の難度と一般性から生じるため、この問題を解決する一つの考え方は難度を下げ一般性を上げることである。形式的な方法が特殊な数学言語及び対応している分析、検証ツールを使用するため、着手しにくい、数学言語の厳格さを体現し、自然言語の表現能力と分かりやすさを体現できる言語があるかどうか、実は形式的な方法自身も図形に基づく類型がある。例えば Petri 図、計時 Petri 図、状態図、これらの図形と UML 図は実は相違点があり、複雑な特殊な数学論理言語に対し、これらの図形はよく知らない人に受け入れられ、より直観に見える、では今これらの図形と UML 図或は自然言語を結び付けて、形式的な方法の敷居を少し下げることができるか。特定の分析、検証ツールについては、列挙項目の妥当性を検証することが主な目的であるが、これは伝統的な方法のテストにも存在し、プログラムの自動化テストでも、それまで結合されていた図形言語をプログラム的に表現してコーディングする場合には、個々のテストの注目点を十分に細かく分離すれば、プログラムの自動化テストによって直接それを分析・検証することができるので、馴染みのないツールを使用することによる余分なコストを回避でき、形式的な方法を用いるツールではなく形式的な方法の考え方に従った。

効率性については、コストの上にも成り立っていると思います。もし上記の考えが良い解決策を持っていれば、中規模から小規模のプロジェクトに大きな余分な支出をすることはなく、形式的な方法を使うことは心配の種ではなく、そのナイーブな考えを実行することで効率性が上がる。

５．参考文献

[1] 《Formal Methods in Software Development》10th National Workshop on Recent Trends in Software Testing (RTST17) at NIT Rourkela

[2] 《Formal Methods:Techniques and Languages For Software Development》Article・March 2015

[3]《Formal Methods for System/Software Engineering: NASA & Army Experiences》 Dr. Mike Hinchey/GSFC Caroline Wang/MSFC Josh McNeil/ARMY