特集2

高信頼ソフトウェア技術の研究動向
―ソフトウェア基礎技術の確立に向けて―

情報通信ユニット　亘理　誠夫

1．はじめに

コンピュータが出現して50年たち、その装置の形態は、スーパーコンピュータ、パソコン、機器に組み込まれるマイクロプロセッサなど多様化している。その利用は、企業活動、社会活動、個人生活の広い範囲に深く入り込んでおり、一度システムに障害が発生すると、その影響は大きい。銀行システムの障害、携帯電話のトラブル、などの例は記憶に新しい。銀行システムでは多くのATMからネットワークを介して多種の取引が行われており、システムが巨大で複雑である。一方、携帯電話は、単なる電話機能のみならず、ホームページ閲覧機能、メール機能、着メロ機能、写真機能など多くの機能を持ち、これらを実現するソフトウェアは複雑で巨大になっている。複雑化・巨大化に伴い、信頼性の確保が課題になっている。

システムの信頼性は、ハードウェアの信頼性、ソフトウェアの信頼性、システム運用管理の信頼性どれ一つ欠けても成立しない。ハードウェアの信頼性は技術の進歩とともに着実に向上しているが、ソフトウェアは人手によって作られるため不安定な要素が入りやすく、その信頼性には課題が多い。かつてメインフレーム（大型コンピュータ）時代には、ソフトウェア技術者の職人的能力によって大規模で複雑なソフトウェアを高い品質で開発していた。しかし、最近では、ソフトウェアが巨大化複雑化していると共に、流通ソフトウェア部品を組み合わせて短期間で開発するケースが多くなり、部品間、モジュール間の仕様記述やインターフェースの信頼性確保が課題となっている。一方で、ソフトウェアの信頼性技術は、メインフレーム時代からあまり大きく変化しておらず、新しい信頼性技術の研究開発が望まれている。

本報告では、ソフトウェア信頼性技術の研究動向を紹介し、日本のソフトウェア信頼性技術向上への課題を探る。

2．ソフトウェア製造の変化

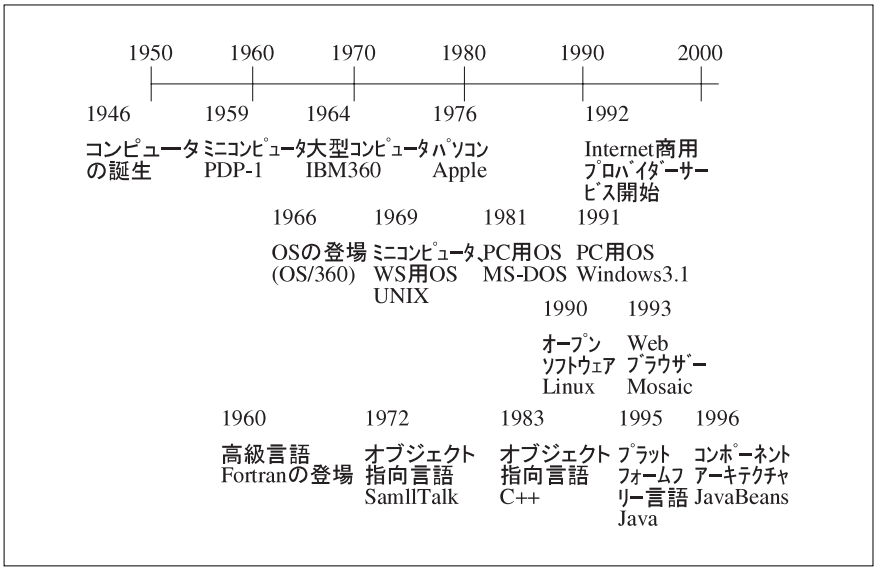
2-1 ソフトウェアを「作る」から「組み合わせる」へ

メインフレーム時代のソフトウェアは、目的ごとに異なったプログラミング言語によって書かれ、そのソフトウェアを動作させるOSはハードウェア系列に依存した企業ごとに異なる独自OSであった。そのため、ソフトウェアの流通がほとんどなく、ソフトウェアは常に一から作り上げていた。その後、ハードウェアのダウンサイジングが進みワークステーション、パソコン時代になると、UNIX、Windowsなどハードウェアに依存しない汎用OSが広く普及するようになった。さらに、Webが普及したインターネット時代になると、OSにも依存しないプラットフォームフリーな（共通の仮想マシン上で動作する）アプリケーションが多く使われるようになってきた。

プログラミングに使われる言語としては、数値計算用のFORTRAN、事務処理用のCOBOLを始めいろいろなプログラム言語が使われてきたが、最近では、C++、Javaなどオブジェクト指向言語が広く普及し標準となっている。このオブジェクト指向言語は、ソフトウェアを独立性の高いモジュールに分離することが可能であり、この普及に伴ってソフトウェアの部品化、流通、再利用が進展した。また、OSとアプリケーションの間にミドルウェアと呼ばれるアプリケーションに共通のライブラリが業種ごとに用意され、アプリケーションプログラム本体はそれを呼び出して使用すればよくなった。このような進展によって、プログラミングは積み木細工のごとくソフトウェア部品を組み合わせてつくることができるようになっ

た。すなわち、ソフトウェアの作り方は「作る」から「組み合わせる」へ変化した。大規模なソフトウェア開発でも短時間で開発することが可能となり、ソフトウェアの生産性は向上したと言える。ソフトウェアの変遷を図表１に示す。

図表１　ソフトウェアの変遷



２－２　ソフトウェアの信頼性向上の必要性

①ソフトウェア開発の変化

最近のソフトウェアには、多くの機能の実現が要求され、複雑化巨大化しており、その信頼性の向上が強く要望されている。例えば、携帯電話のソフトウェアは既に数百万行を超えている。ソフトウェア技術者一人が把握できる範囲は１万行程度と言われ、多人数で開発されている。このように巨大なソフトウェアは、複数企業の複数拠点において開発され、場合によっては開発コストを低減するため海外拠点が利用される。従って、ソフトウェア技術者間、拠点間のコミュニケーションを良好に保つ開発工程（プロセス）管理が重要であり、これが作成されるソフトウェアの品質を左右する。

さらに、IT製品の市場の変化は早く、システムの仕様変更への柔軟性、開発期間の短縮、システムの保守性向上が求められている。このような市場圧力から、短納期のためテストが不十分になったり、仕様変更に伴う整合性チェックをおろそかにしたり、保守のための機能を十分に取らなかったりすることが発生している。開発を優先し信頼性を後回しにする危険性が多く存在している。

一方、ソフトウェア部品の流通が盛んになり、それを使用することによって生産性は向上している。しかし、流通しているソフトウェア部品の信頼性が未確認であったり、インターフェースの不整合やバージョンの違いによる仕様の不整合が発生することが多い。ソフトウェア部品のブラックボックス化における危険性を十分認識する必要がある。

②オープンソースソフトウェアの信頼性

最近、オープンソースソフトウェアの利用が盛んになっている。1990年にPC用Unix互換ＯＳであるLinuxがオープンソースソフトウェア方式によって開発されて注目されるようなり、その後インターネット分散システムの基盤ソフトウェアなど多くのソフトウェアがこの方式で開発されてきている。オープンソースソフトウェアは、ソフトウェアの仕様とプログラムのソースコードが公開され、誰でもそのメカニズムを検証できる。ソフトウェアの開発は多数のボランティアがコミュニティを形成して、同時並行的に進められる。優れた提案がすぐフィードバックされ、よりよいソフトウェアが作られていく。さらにこのコミュニティメンバーが開発したソフトウェアの先進ユーザーとなり、普及が促進される。

また、ハッカーなどによるセキュリティ問題が多発するようになり、その対応のためにソフトウェアのメカニズムを知る必要があることから、ソフトウェアのメカニズムの公開が要求されるようになっている。この点からも、オープンソースソフトウェアは歓迎されている。

図表２　ソフトウェアの信頼性を低下させる要因

- システムが多機能、複雑化
- 仕様の頻繁な変更
- 短納期がテスト軽視を助長
- 海外を含む分散拠点での開発の進展
- ソフトウェア部品のブラックボックス化の進展
- 品質未確認のソフトウェア（オープンソース、ソフトウェア部品）の流通

上述のようなオープンソースソフトウェアの利点がクローズアップされているが、一方、オープンソースソフトウェアは基本的に無償であるため、その品質の保証はない。オープンソフトウェアの開発過程にて十分テストが行われ、多くの使用歴が積み上げられ、信頼性が高くなっていればそのソフトウェアの品質に問題はない。しかし、通常、オープンソースソフトウェアの開発は無償のボランティアによって行われているため、利用する前にその信頼性を確認する必要があり、未確認の場合は検査テスト等に多くの工数がかかることを覚悟しなければならない。以上説明したソフトウェアの信頼性を低下させる要因を図表2にまとめた。

3．ソフトウェア信頼性技術

ソフトウェアの信頼性を確保するための技術としては、図表3に示すように、ソフトウェア開発各工程における開発支援技術、ソフトウェアの開発工程（プロセス）の管理技術、ソフトウェア開発スタイル・手法がある。以下にその技術を紹介する。

図表3　ソフトウェア信頼性技術

設計工程	プログラミング工程	テスト工程
UML 形式手法	オブジェクト指向言語	論理網羅テスト データ限界値分析 状態推移テスト
プロセス管理		
CMM、TQC		
開発スタイル・手法		
スパイラル型開発、XP		

3－1 オブジェクト指向言語

オブジェクト指向言語は、プログラムとそれに付随するデータをひと纏まりとして独立させ（それをオブジェクトと呼ぶ）、オブジェクト間のメッセージ交換によって動作を記述するプログラミング言語である。オブジェクト指向言語では、データをこのオブジェクト内に閉じこめること（隠蔽）によって、データの不用意な他からのアクセスを防ぎ、信頼性を高めることができる。また、オブジェクトのインターフェースを明確に記述させることによっても信頼性を高めることができる。

しかし一方で、プログラミング言語はどのような仕様も記述可能でなければならず、柔軟性が求められる。その柔軟性が、オブジェクト間のインターフェースに不整合を許す場合があり、信頼性は保証されない。すなわち、プログラミング言語のみでは、信頼性と柔軟性を両立させることはできない。

3－2 システム設計記述

ソフトウェアの開発工程においては、システムが見通しよく設計され、その設計が開発メンバーによく伝わっていることが、高信頼なソフトウェア作成に必須である。システム設計に使われる設計図の記述法としてUML（Unified Modeling Language）が1997年にオブジェクト指向言語の標準化団体であるOMG（Object Management Group）によって標準化されている。それまで、多くのシステム記述法が乱立していたが､3つの知名度の高い方法の開発者（注1）がその統一化を図ってUMLを開発した。ソフトウェア設計で使われてきた数多くの図式の集大成ともなっている。これを使用することにより、開発メンバーがシステムの設計を共通の言葉で理解し合えるようになった。

しかし、UMLは設計図の表記法を規定したものであり、設計内容の自由度は制限してない。従ってどのようなシステムでも記述でき、記述された設計の信頼性が保証されているわけではない。

3－3 形式手法

形式手法とは、代数論、集合論やグラフ理論など数学理論をベースとした手法であり、要求されている仕様を数学的に厳密に記述できる。また、作成されたソフトウェアの性質を数学理論によって検査・検証することができる。仕様が形式手法を用いて記述されていれば、仕様に誤りが検出されたり途中で変更された場合、その仕様訂正は、数学的に正しく行われる。仕様変更や部分訂正は、システム全体に思わぬ影響が波及することがしばしばあるため、その仕様変更は信頼性を低下させる大きな原因の一つである。現状では、経験に頼った個別の処理で対応してお

（注1）UMLは、Booch法の提唱者Grady BoochとOMT（Object Modeling Technique）法の提唱者James Rumbaughとユースケースの提唱者Ivar Jacobsonによって、システム表記法の統一を目指し開発された。

り、信頼性確保が課題である。

形式手法の研究は、1970年代初め頃から始められ、これまでに種々のシステム仕様記述法やシステム仕様検証法が開発されてきている[1, 2]。

⑴形式仕様記述言語

形式仕様記述言語とは、数学理論に基づいて仕様を記述する言語であり、記述された仕様は誤りや矛盾がないことが保証される。これまで欧州を中心に種々の記述言語が開発されている。

形式仕様言語は2つに分類され、ソフトウェア仕様を集合論などに基づくモデルによって記述するモデル指向言語とデータの性質を代数論などに基づいて記述する性質指向言語がある。

モデル指向言語としては、1970年代終わりから研究が始まり、英国Oxford大学にて開発されたZ記法(注2)、フランスにて開発されたBメソッド(注3)、IBMウィーン研究所にて開発されたVDM（Vienna Development Method）(注4)などがある。一方、性質指向言語としては、1977年ごろからカリフォルニア大とスタンフォード研究所（SRI）にて開発されたOBJ(注5)、その発展として1995年ごろから北陸先端大にて開発されたCafeOBJ(注6)などがある。

モデル指向言語の代表的なZ記法では、システム仕様を状態機械（状態空間、初期状態、状態空間に作用する操作）として記述し、システムがどのような状態になるか事前に数学的に求めることができる。この手法では単一の状態空間を用いているため逐次処理型のソフトウェア仕様を記述は可能であるが、分散協調型のソフトウェア仕様の記述は困難である。一方、性質指向言語であるCafeOBJでは、データとプロセスを等式論理に基づき統一的に表現し、分散システムの仕様記述や検証を可能としている。しかし、性質指向言語の記述法は関数型言語に近くその表現力が低いのが問題である。すなわち、モデル指向言語は、表現の自由度は比較的高いが解析や検証は難しい。一方、性質指向言語は、表現力は制限されるが解析や検証が容易である。

これらの形式仕様記述言語には、多数の数学モデルに対応した多数の形式仕様記述言語が存在しており統一化されていない。また、記述表現は数学的表記が中心で自然言語から遠いため理解に時間がかかる。このような要因により、高度な信頼性が求められる問題領域のみで適用され、形式仕様記述言語は未だ広く浸透していない。

⑵形式手法による検査・検証ツール

実問題解決に向けた形式手法の研究が米国では盛んであり、記述された仕様やプログラムの動作検査や不正な動作を起こさないことを検証するツールが開発されてきている。このツールは2つに分類される。状態探索モデルに基づく検査ツールとして、1980年に米国ベル研究所にて開発されたSPIN(注7)、1987年に米国Carnegie Mellon大学で提案されたSMV（Symbolic Model Verifier）(注8)などがある。もう一つの分類である定理証明に基づく検証ツールとして、1980年代半ばに米国のスタンフォード研究所（SRI）にて開発されたVS（Prototype Verification System）(注9)

（注2）Z記法は、1970年代終わりにOxford大学Programming Research Groupにより開発された言語であり、集合論に基づきシステム仕様を状態機械（状態空間、初期状態、状態空間に作用する操作）として記述する。

（注3）Bメソッドは、1980年代半ばにフランスのJean-Raymond Abrialが開発したソフトウェア開発ツールで、Z記法をベースとしている。

（注4）VDMは、1974年PL／Iコンパイラーが正しく作られていることを検証したIBMウィーン研究所の研究をルーツとしている。データ集合、リスト、マップから数学的モデルを構成し、その状態変化としてシステム仕様を記述する。

（注5）OBJは、1977年以降にカリフォルニア大のJoseph Goguen（後にスタンフォード研究所(SRI)、オックスフォード大）を中心に開発され、代数論に基づいて抽象データ型を厳密に表現する形式仕様言語である。

（注6）CafeOBJは、OBJの発展として、北陸先端大をリーダーとする国際チームにより、隠蔽代数理論に基づいてデータとプロセスを統一的にモデル化しシステムの動作仕様を記述する言語として開発され、分散システムの動作仕様記述や検証を可能としている。

（注7）SPINは、1980年に米国のベル研究所にて開発が開始された検証ツールであり、線形時相論理（Linear temporal logic）に基づき、システムの動作を網羅的にチェックして、プログラムがダウンせず正しく動作することを検証する。非同期分散システムの動作の検証が可能である。

（注8）SMVは、1987年米国のCMU（Carnegie Mellon Univ.）のK. McMillanが提案した検査ツールであり、検査対象の状態空間を二分決定ダイアグラム（Binary Decision Diagram）よって圧縮して効率的にシステムの動作を検査する。

（注9）PVSは、1980年代半ばに米国のスタンフォード研究所（SRI）にて開発された汎用の定理証明ツールであり、高階論理による証明を用いて、仕様機能の一貫性や完全性を検証する。

図表4　形式手法の適用例

	適用例	形式手法	効果
米国	航空機の衝突回避システムTCAS Ⅱ	RSML	衝突回避の仕様を形式手法にて設計
	土星探査衛星のフォールトプロテクション	PVS	フォールトプロテクション機能を検証
	AT&T交換機プログラム	SPIN	不正動作の網羅的検査
欧州	パリ地下鉄無人運行システム	Bメソッド	安全性を確保
	ロンドン航空管制システム	VDM	品質を10倍以上向上
	欧州IBM顧客情報管理システム	Z記法	9％コスト削減と品質向上
日本	鉄道のポイント制御システム	VDM	安全性を確保、かつ大量運行を可能
	Enterprise Java Beams振る舞い仕様	SPIN	EJB 1.1の仕様記載内容の問題点を指摘

（注）　VDM：Vienna Development Method　　RSML：Requirements State Machine Language
PVS：Prototype Verification System

などがある。

状態探索モデル検査ツールは、自動検査が可能であるが、検査対象が有限状態推移で記述できるシステムに限られる。また、ソフトウェアが大規模になると状態数が増大し、計算量が爆発して実行できないことがある。一方、定理証明による検証ツールには汎用性はあるが、対象の性質に応じて定理を選択する必要があり、自動化はできていない。

日本でも、東工大、東大などにおいて、論理代数に基づいたシステム動作検証システムが研究されており、外部からのあらゆる入力に対してシステム動作が停止しないことの検証法を開発し、これを用いてセキュリティ認証プロトコルの検証が行われた。また、北陸先端大などにおいて、定理証明法を発展させたオブジェクト指向モデル分析の検証技法が開発され、組み込みソフトウェアに使われる状態遷移モデルの検証が研究されている。

上述のようにいろいろな検査・検証ツールが開発されているが、これらのツールにてシステム全体を検査・検証はできない。ツールにて検査・検証できる部分をシステムの全体の中から適切に切り出さねばならない。ツールの利用には、形式手法のメカニズムの基本知識が必要であり、普及を阻んでいる。

⑶形式手法の適用例

形式手法を実問題に適用した例としては、図表4に示すように、コストより安全性を重視しているシステムの設計や検証に使われている。しかし、その利用範囲は比較的小規模なシステムに限定されている。今後は、より大規模で複雑なシステムにも対応できる手法、手軽に使用できる手法、現在主流となっているオブジェクト指向言語の環境と親和性のよい手法の実現が望まれる。

3－4　テスト技術

テストは、システム開発全体の半数以上の工数が必要であるとも言われており、効率的なテストが求められている。プログラム製作中にテストのためのチェックポイントを埋め込むなど事前準備が大切である。このテストを行うためのツールとして、代数やグラフ理論など数学理論をベースとした手法が開発されている。特に、オブジェクト指向プログラムでは、オペレーションが動的に決定（ダイナミックバインディング）されるなどプログラム実行時に決定される要素があり、それを事前にテストする方法として、状態推移図を用いたツールが開発されている。

以下にグラフ理論や代数を利用したテスト技術の例を紹介する。

①パステスト法

プログラムから制御フローを抽出し、テストケースを生成する。制御フローを網羅するようにテストが生成されるので、すべてのソースコードを一度は必ずチェックすることができる。

②論理網羅テスト

論理判定の条件をすべて網羅するように組み合わせ論理によってテストケースを自動的に発生させる。

③状態推移テスト

仕様を状態推移図で記述し、すべての推移をパスするようにテストする。ネットワークプロトコルの検証など、並行性問題を含んでいる場合は、状態空間の爆発を避けるため、ペトリネットのトークンに値を持たせ並行性に関する誤り発見、性能解析を行う方法もある。

④データフローテスト

制御フローグラフを使用し、データに発生する不正を検出する。

⑤データ限界値分析

データの上限値、ゼロ、下限値でシステムが不正動作しないかテストする。

以上、テスト技術について、説明したが、テストの実行に当たっては、バグ出し（プログラムの誤り）に十分な時間をかけ、バグの

数が十分少なくなることを確認する必要がある。しかし、限られた開発期間の中でテストを効率よく行うために、システムの動作の核となる重要な部分を先に時間をかけて行い、システム主要動作に影響を与えない部分は後へ廻すなどテスト手順の戦略も重要である。

3-5 ソフトウェアプロセス管理手法

品質の高いシステムソフトウェアを開発製造するためには、その開発工程（ソフトウェアプロセス）を十分に管理する必要がある。特に、大規模なシステム開発は、多数の開発者が複数の拠点に分かれて実施される。このプロセス管理が不十分であると品質の悪いソフトウェアが作成されたり、開発スケジュールが遅れたりする。

従来、日本のソフトウェア品質向上には、ハードウェアの品質向上と同様にTQC（Total Quality Control）の一環である小集団活動をベースとしたノウハウの蓄積・共有化が成果を上げてきた。

一方、米国では、ソフトウェア技術者の流動性が高く、個人能力に大きく依存する品質管理手法では大きな効果を出せず限界があった。そこで、個人ではなく、組織に注目して品質管理ノウハウ蓄積を行う手法CMM（Capability Maturity Model for Software）が開発された。この手法では、ソフトウェアを開発する組織に、求められる品質保証の要素・機能のレベル（成熟度）を設定し、そのレベルを上げる努力を求めた。この成熟度を測定することによって、その組織・ベンダーの信頼性レベルを評価することができる。

CMMは、米国国防省の依頼によりカーネギーメロン大学ソフトウェア工学研究所が開発し、1987年に公表されている。当初は、主に防衛関連の開発に使われていたが、ソフトウェア品質改善の指標が得られることから世界中に普及し、現在では適用実績は40カ国を上回っている。特にインド、中国では積極的に導入する動きが目立っている。

日本のボトムアップ活動である小集団活動ベースの品質向上ノウハウの蓄積は、CMMや国際的な品質保証規格（ISO9000シリーズ）に一部反映されているが、CMMは組織の能力改善を要求するトップダウン活動が主体である。このようなボトムアップからトップダウンへの発想の転換は、やはり米国から出現している。

3-6 ソフトウェア開発スタイル

最近注目されている開発スタイルに、スパイラル型開発とＸＰ（Extreme Programming）がある。これらは、信頼性を向上させるために考え出されたものではないが、開発スタイルを変えることにより結果的に信頼性が向上している。

⑴スパイラル型開発

従来のソフトウェア開発は、ウォーターホール型開発と呼ばれ、設計、プログラム作成、テストの各工程を順に実行するため、プロジェクト管理は容易であった。しかし、テストは全工程の最終部分にあり、仕様記述やプログラムの誤り（バグ）を発見するのが遅い。全システムの仕様がすべて決定してからプログラミングに入るため、開発が遅れやすかった。また、仕様の途中変更は、プログラミングすべてのやり直しを必要とし、大きな追加工数をもたらしていた。

スパイラル型開発（インクリメンタル型開発とも呼ばれる）では、システムの核となる部分を取り出し、この部分から先に設計、プログラム作成、テストまで行う。核の部分の開発後、周辺部分を順次開発し加えていく。従って、重要な核となる部分は、開発の早い段階からテストが十分行え、そのテストの結果を踏まえて、周辺部分の開発へと進めていくことができる。

⑵XP（Extreme Programming）

1999年に米国のKent Beckが高品質なソフトウェアを早く開発する新しい開発スタイルＸＰを提唱し、注目を集めている。このＸＰの特長は、「ペアプログラミング」、「テストファースト」にある。「ペアプログラミング」とは、仕様からプログラムを作成する人とそれをテストする人がペアになって開発を進める手法であり、工数の半分はテストにつぎ込んでいることになる。また、「テストファースト」とは、プログラム本体をコーディング（作成）する前に、仕様を満足するテストを作成しておく方法である。作成されたプログラムがテストをクリアするまで、コーディングとテストを繰り返していく。ＸＰでは、この2つの特長に加えて、開発メンバーが実践すべき規範を例えば、シンプルな設計、プログラムコードの共有、オープンな作業環境など過去の経験から定めてあり、それを最大限に実践するように求めたことからextremeという名前が付けられた。

ＸＰは、テストを重視した開発スタイルであり、信頼性の高いソフトウェアを開発することができる。しかしながら、この開発スタイルは、システムをペアプログラミングする単位に分割して開発するため、比較的小規模なソフトウェアの開発向きである。

4．日米欧におけるソフトウェア信頼性技術研究の推進動向

⑴米国

米国は、ソフトウェア研究において常にトップを走ってきており、産業競争力も圧倒的に強い。ソフトウェア信頼性技術では、DARPAとNSFを中心に軍事関連分野と基礎研究分野への研究支援が積極的に行われている。また、これらの研究開発において蓄積された技術を産官学連携にて産業界に積極的に技術移転し、新世代の製品を生んできている。さらに、同時多発テロ事件以後、国土の安全保障の観点から、信頼性、セキュリティに対する投資が増加している。

具体的には、国による研究支援として、ミッションクリティカルなシステムの信頼性、セキュリティおよび安全性を目指した「高信頼性ソフトウェア・システム（High-Confidence Software and Systems)」プログラムや科学と工学によるソフトウェアのコスト効率の改善を目指した「ソフトウェア設計と生産性（Software Design and Productivity)」プログラムが積極的に推進されている。

⑵欧州

欧州のソフトウェア産業は米国ほど華々しくないが、数学的な基礎理論ベースとしたアカデミックな手法の研究が盛んである。形式仕様記述言語Z、VDMは欧州の研究機関が提案をしたものである。仕様設計に用いるＵＭＬに形式記述を取り入れる研究も、欧州で盛んである。

ＥＵの研究支援ファンドを見ても、1985年から始まったESPRIT（European Commission's Information Technologies Research Programme）から2002年から始まった第6次フレームワークプログラム（FP 6）まで、一貫してソフトウェア信頼性技術を取り上げている。

⑶日本

日本では、産業界が米国から概念や研究成果を導入して信頼性技術を確保してきた。国のプロジェクトでは技術キャッチアップを目的とするものが多かったが、最近は、独自技術や基礎技術育成のためのプロジェクトが始められている。例えば、科学技術振興事業団（JST）では、若手研究者育成を目的とする「戦略的創造研究推進さきがけ21」の「機能と構成」（2000年から2005年まで）の計画の中で、基礎理論をベースとしたソフトウェア信頼性の課題を取り上げている。情報処理振興事業協会（IPA）では、産学連携により世界市場でのデファクト・スタンダードを確保するソフトウェアの開発を目的とする「次世代ソフトウェア開発事業」の中で、「高信頼・高安全ソフトウェア」（2002年から2007年まで）を取り上げている。さらに、文部科学省では、高度情報通信システム形成のための鍵となるソフトウェアを開発し、安心して参加できるIT社会の構築を目的とする「e-Society基盤ソフトウェア総合開発」（2003年度から2007年度まで）の一部として「高い生産性を持つ高信頼ソフトウェア作成技術の開発」が取り上げられている。

以上述べたように、欧州では理論研究が盛んであり、米国では、実問題への適用を模索したモデル化研究が盛んである。欧米間では研究者の移動も盛んで、欧州にて研究した理論を米国で実証する流れが見られる。一方、日本では、大学を中心に基礎研究が進められているが、その研究は小規模なモデル検証であり実問題には適用されてこなかった。いわゆるtoyモデルの研究に留まり、産業界からは遠く、実問題への挑戦があまりされていない。日本が欧米の輪の中に入り技術発展への貢献を大きくしていくには、単なる理論の提示で終わらず、実問題が解ける理論を提示していく必要があろう。

また、ソフトウェア信頼性技術の研究開発は、直接利益に結びにくく企業では不活発である。欧米ではソフトウェア基礎技術として国が継続的に育成している。日本でも継続的な国の支援が望まれる。

5．ソフトウェア信頼性強化への課題

ソフトウェアの信頼性を強化するためには、信頼性技術そのものを発展させることとソフトウェア開発において高信頼性技術を最大限駆使することが求められる。信頼性技術を導入する面では、我が国は、前述したＸＰやＣＭＭを初め多くの米国発技術を導入してきているが、ソフトウェア開発・管理には文化的側面もあり、日本の企業文化、技術者文化を考慮した導入が必要であろう。

一方、信頼性技術そのものは、少しずつ進展してきているがまだ限定された場面でしか使えず、今後の発展が望まれる。特に、システム設計を行う上流工程における信頼性向上はその後の下流工程に

大きな影響を与える意味でより重要である。例えば、設計工程において仕様の誤りチェックを厳密に行えば、誤りの伝搬が早い段階で止まり、テスト工数の削減に繋がる。また、仕様が形式手法で記述されていれば、仕様変更に対しても、計算機を用いてほぼ自動的に改訂をチェックすることができ、信頼性を確保できる。

しかし、現状では、形式手法の実問題への適用にはまだ手間がかかり、その利用分野は安全性確保が必須なシステムなどに限定されている。その理由としては、形式仕様言語の表現範囲が、通常のプログラム言語の表現範囲より狭いこと、大規模問題では計算量が膨大になることがあり、解ける問題の規模に制限があること、仕様を形式的に記述するには形式手法の技術習得が必要なことなどがある。

一方、日本が強い分野である情報家電、自動車、制御システムなど装置に組み込まれる「組み込みソフトウェア」でも、信頼性向上が課題となってきている。この組み込みソフトウェアには、リアルタイム性が求められ、また限られた資源を最大限利用しなければならないため、職人芸的な要素があり、日本人のきめ細かさが性能が高く質の良いソフトウェアを作っていた。年間50億個以上作られるマイクロプロセッサー（制御用の小型プロセッサーを含めて）の半数には日本発TRONがOSとして使われているなど、組み込みソフトウェア分野では、日本は世界をリードしてきた。

しかし、この組み込みソフトウェアの分野でも、ハードウェアの性能が向上すると共にソフトウェアに多くの機能が要求され、流通しているソフトウェア部品を多用するようになってきている。そのソフトウェア部品は圧倒的に米国の力が強くその影響を受けている。組み込みソフトウェアは一般に多くの数が販売され、不良が発生した場合の影響が大きいため、信頼性は重要なファクターである。従って、組み込みソフトウェア分野で日本がリードを守る一つの方策は、革新的な信頼性技術を確立することにあろう。

そのための一つの方策としては、日本の大学で、継続的に研究されてきた形式手法の基礎理論を米国のように実問題に適用する試みを積極的に進めることである。基礎研究の成果を実問題に適用する中でその限界を見つけ、次の革新的な理論を組み立てていく努力が求められる。企業側には実問題を提供し共同研究していくことが求められる。

また、米国の情報分野の研究に対抗して行くには、日本の情報分野の研究の層の厚さも強化する必要がある。例えば、日米の学生数の統計(注10)によると情報通信分野（電気通信工学と数学／計算機学の合計）の2000年における米国の学生数は、日本に比べ、学部卒1.8倍、修士卒3.0倍、博士取得4.7倍である。学生数の増強が望まれる。一方、中国では大学院の増強が図られ、2000年度すでに大学院学生数では、日本の21万人を上回る30万人(注11)となっており、日本としては量のみならずその質の向上が重要な課題である。従来、情報分野の研究は、欧米の後追い研究が多かったが、キャッチアップの時代は終わり、現在は、独創的で新しい技術を産業界が求めている。従って、研究者の教育では、平均レベルの向上や弱点克服より、一芸に秀でた人材が求められている。強いところを強くし、持っていない部分は他のパートナーと連携する時代を意識する必要があろう。

（注10）日本のデータは、文部科学省学校基本調査報告書（高等教育機関編）のH13年3月における工学部電気通信工学と理学部数学の学部、修士課程、博士課程の卒業生数である。ただし、博士課程では博士を取得せず満期退学した卒業生は除いた。米国のデータは、NSF Division of Science Resources Statisticsから2000年のElectrical EngineeringとMathematical/Computer Scienceの学士、修士、博士の学位取得者数である。

（注11）出典は、文部科学省「教育指標の国際比較」H15年版、H14年版。

6．おわりに

日本のハードウェア産業の中で、強い競争力を持っている製品の特徴は、コストが低いことではなく、その品質・信頼性が高いことにある。ソフトウェア産業においてもさらに発展していくためには、ソフトウェアの信頼性を向上させることが必須の課題の一つである。ソフトウェアは年々複雑化巨大化する一方で、製品のライフサイクルは短くなり、ソフトウェアの仕様改善によるバージョンアップも頻繁である。仕様変更が高い信頼性のもとで、短期間に実行できることが望まれている。そのため、数学的理論をベースとした手法による信頼性技術の進展が望まれる。

従来、日本の大学の信頼性技術研究は基礎理論で終わり実問題には手を触れていなかった。一方、日本の企業はほとんど米国から基礎技術を導入し日本向きに合わせる開発をしてきた。しかし、独自

技術を持たなければソフトウェア産業も新技術で米国に、コスト競争で中国、インドに負けていくであろう。

日本が独創的な信頼性技術を持つためには、独創的な基礎研究から出た成果を産学連携で実問題へ展開する挑戦が必要である。実問題へ展開するには、基礎研究とは桁違いに大きな研究資金が必要となるが、企業は直接コスト削減に結びつかない信頼性技術の研究には消極的であるため、国による支援が望まれる。さらに、基礎から応用への技術展開と、応用から基礎への研究課題提起というよい循環が生まれることが、信頼性技術を真に強くしていくことになる。

信頼性技術研究には、積み重ねによる改良が求められ、地味で継続的な努力が必要である。一般にIT分野では新しいビジネスモデルや新しいシステムを作る統合化技術に注目が集まるが、信頼性技術のようにITの底流にあって広い範囲に影響を与える基礎技術を継続的に進展させていく努力もまた重要である。

謝辞

本稿をまとめるに当たり、北陸先端科学技術大学院大学片山卓也教授、二木厚吉教授、大阪大学井上克郎教授、奈良先端科学技術大学院大学松本健一教授、法政大学中島震教授から、ソフトウェア信頼性技術動向に関して貴重なご意見を頂きました。ここに深く感謝いたします。

参考文献

1）E. M. Clarke, J. M. Wing, “Formal Method: State of the Art and Future Directions”, ACM Computing Survey, Dec.1996）

2）ソフトウェア工学研究財団、「ソフトウェア工学の戦略的推進に関する調査研究」、2002.3