組込ソフトウェアのテスト技術：課題と対策 ～ テスト知識のアップデート ～

松尾谷 徹
有限会社 デバッグ工学研究所
matsuodani@biglobe.jp

要旨

組込ソフトウェアは，ピック（*PIC*）と呼ばれている超小型から高度運転支援（*ADAS*）や *Autoware/ROS2* による実用レベルの自動運転まで，幅広いバリエーションがあります．そのテスト環境も技法も多様ですが，汎用 *OS*（*Windows*，*Linux* など）のテスト技術とは異なり知名度が低く，導入が進んでいません．

組込開発の本質的な特性はハードリアルタイム処理における「確定性」です．物理的エネルギーを持つシステムの制御は，振る舞いとしての高信頼性や安全性が極めて重要であり，その根拠になるのが「確定性」です．組込エンジニアは，確定性の実現など特殊性の高いナレッジが必要な職種であり，新たな人的資源の投入で簡単にスキルを習得することが難しく，現職エンジニアのナレッジアップデートに頼ることになります．そのためには，技術の整理と理解が必要と考え本稿をまとめました．

1. はじめに

組込ソフトウェア（ESoft: Embedded Software）は，現代における「モノ作り」の基盤技術の一つです．「モノ作り」の対象には，家電や IoT などの小型から，自動運転などの大規模で系統的なシステムまで多様な製品種があります [1]．ESoft は，規模に関わらずモノを制御することから，そのテスト（組込テスト）は機能だけでなく「振る舞い」を対象として評価します．

ESoft の振る舞いで重要な特性は「確定性」です [2]．確定性とは，処理の順番や動作の遅延時間も変動しない特性です．一般的な性能要件のテストは，平均応答時間や応答時間の分布を調べ，システムのスループットを高めます．組込システムにおけるリアルタイム要件は，スループットではなく，厳密な処理の再現性で，最大遅延時間がマイクロ秒やミリ秒単位で要求されるハードリアルタイム要件と呼ばれています．

「確定性」を実現する処理タスクは，特殊な記述と制御が必要です．制御には RTOS（Real-Time Operating System）が使われます．使用するハードの性能を引き出し，厳しいハードリアルタイム要件を満たすため，RTOS は周期タスク制御を使用します [2]．

RTOS は確定性が必要でない非確定性の処理も行います．結果，組込テストでは「非確定性タスク」が確定性要件を阻害しないことも評価する必要があります．阻害は，論理的な依存関係だけでなく，ECU リソースの競合やキャッシュ状態などからも生じるので，特別なテストが必要になります．

2 章では，RTOS の機能と実装方式による「確定性」の仕組みを中心に，複数のテスト技法によって実現している組込テスト技術と抱えている課題について説明します．ESoft テストが抱えている課題については 4 種類に分類して示し，対策を 3 章以降で示します．4 種類の課題とは，

(a) 仕様変更の影響課題

(b) タスク間の影響課題

(c) クロス開発の課題

(d) ナレッジアップデートの課題

3 章では，(a) の課題である仕様のちょっとした変更でも挙動に影響を及ぼすことから，制御仕様と呼ばれる挙動に関する仕様を主とし，コード実装などの影響を緩和する対策として，モデルベース開発や形式仕様について説明します．

4 章では，(b) の課題である実装における特別な制約やタスク間の影響に関する課題と，(c) のクロス開発の課題について考えます．(b) の対策としては英国の自動車開発から生まれた MISRA-C の成果や Android OS のサンドボックス機能について，(c) の課題については NASA のアプローチや仮想化技術を説明します．

5 章では，「モノ」を制御する組込システムのナレッジが，「情報」を扱う一般的なソフトウェアのナレッジとは異なることから生じる課題を「技術異文化」と考え，技術のステレオタイプ化による「無知の誤り」対策について説明します．

2. 組込ソフトの特殊性

この章では ESoft の特殊性について説明し，最後に組込ソフトの特殊性に対する課題をまとめます．

2.1. 目的と歴史

コンピュータを使った装置の制御は，マイクロコンピュータが発明される以前からリレーによるシーケンス制御を置き換えた装置や，ミニコンピュータによるデジタル制御として実用化されていました．組込システムのマイクロコンピュータが誕生したのは，1976 年に Microchip Technology 社によって開発された 4bit PIC マイクロコンピュータです．

初期のコードはアセンブラで製品個別に書かれていましたが，1980 年代には製品としての RTOS が開発されました．この時代の PC の OS はマイクロソフトの MS-DOS(1981 年) で，規模的にも同程度 (10KB 程度) です．

組込システムの目的は，リレーなど電気的あるいはアナログ/デジタル変換など電子的な手段を通じて「モノの制御」を行うことです．テストの目的も，正しい制御を確実にするため「振舞い」，すなわち時系列として応答時間まで踏み込んだ検証を行います．

2.2. アーキテクチャと入出力

組込システムで用いるマイクロコンピュータは，制御用のものであり，図 1 に示すようなアーキテクチャを持ちます．入出力を司る周辺回路の部分が，制御対象の「モノ」によって多様であり，例えば携帯電話 3G 用，エンジン制御用などの種類があります．

多様な周辺回路のテストは，IC の回路テストやファームウェアのテストにおいても個別に開発が必要であり，組込システムの課題の一つです．ソフトウェアのテストでも，テストを実行する特殊なハードである HiL(Hardware-in-the-Loop) 評価ボード自身のテストが必要であり，これもクロス開発の課題です．

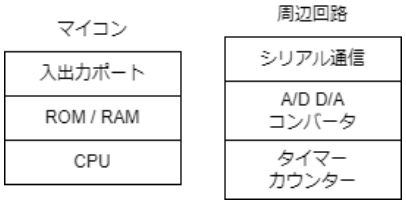

図 1: 制御用マイクロコンピュータのアーキテクチャ

2.3. タスクの確定性と制御

組込システムに求められる特性は，ハードリアルタイム特性であり，その特性は制御を行うアプリケーションタスクの確定性と周期タスク制御で決まります．

周期タスクの制御は，RTOS が行います．RTOS は Linux など固有名ではなく，μ ITRON, VxWorks, QNX, FreeRTOS など様々な製品に対する総称です．

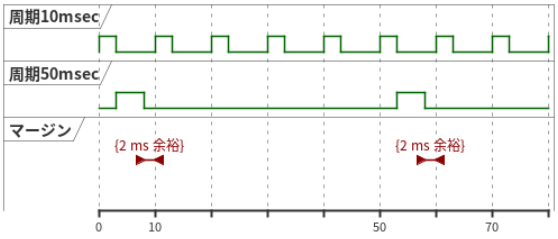

図 2: 周期タスクと余裕時間の例

周期タスク制御は，複数の周期と複数のタスクで構成されます．図 2 は 2 種類の周期タスク 10ms と 50ms による例を示しています．説明を単純化するため，10ms 周期タスクには 3ms で終わる TA_{10} と，50ms 周期タス

クには 5ms で終わる TB_{50} の動作を時間軸で示しています．図のマージンとは，この仕様が成り立つ余裕時間のことで，10ms と 50ms の処理が重なる場合において，次の周期が始まるまでの時間です．この例では 2ms です．

TA_{10} と TB_{50} の処理時間を 3ms と 5ms としましたが，実際には処理内容による変化や割込みによるキャッシュの状態が影響し変動が生じます．この変動はジッタと呼ばれ，変動を含めた確定性をテストとレビューで確保します．

タスクの処理時間を 3ms と 5ms とする責任は，アプリケーションタスクの実装によって決まります．RTOS では制御できない特性です．アプリケーションタスクのプログラミングにおいて，再帰処理や動的メモリを使わないのは，実行時間の変動を抑えるためです．この規則は，ソフトウェア工学におけるオブジェクト指向設計など流用性を高めるための汎化を否定しています．理由は，確定性を確実にするためです．

実際の ESoft では，複数の周期と複数の周期タスクが組み込まれます．設計段階で，最悪余裕時間となるタスクの組み合わせと余裕時間を厳密に定義し，実装段階で各タスクの実行ステップ数を確認して実行時間を積み上げ方式で推測し，ハードリアルタイム性を保証しています．

この方式は，原理的に ECU の稼働率を低くします．汎用 OS の目的は，ハードウェア資源の利用効率を高めスループットを大きくし，開発では部品の汎用性を高め流用率を上げるため階層構造が使われますが，RTOS は資源効率を抑えても確定性を守る方式です．

2.4. クロス開発環境

汎用開発では開発環境とテストベッドは連携していて，テストのために特別な装置や環境の準備はそれほど多くありません．ESoft では，開発環境と実行環境が異なる (クロス環境) ため特別な準備が必要になります．

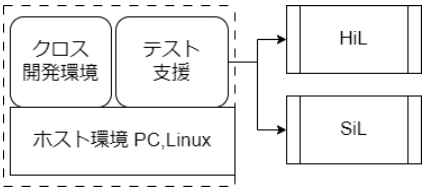

図 3: クロス開発とテストの概念図

図 3 にクロス開発とテストの概念図を示します．コードの開発は，PC 上でクロスコンパイラとリンカーを用いて対象とする ECU で動作する実行コードを生成します．実行コードには，RTOS 本体と I/O のためのレジスタエリアを含めて作成します．

クロスコンパイラの言語仕様が，GCC など標準と一致していれば，関数レベルの単体テストのみホストマシン上でテスト可能です．

機能単体テスト以降は，PC に接続した評価ボードと呼ばれる Hardware-in-the-Loop (HiL) やエミュレータによる Software-in-the-Loop (SiL) を使ってテストを実行します．

HiL は，使用する ECU と周辺回路にデータをセットしたり読み出したりするデバッグ用のハードを加えたもので，対象製品ごとに製品とは別に準備する必要があります．近年，RTOS 製品の中には，汎用的な ECU の命令セットをサポートする標準的な SiL 機能を提供している場合もあります．

2.5. 特殊性と課題

この章では，ESoft の動作を現代のテスト技術では簡単に確認できないことを述べてきました．対策のトレンドとしては shift left 的な発想でテスト環境や技法が必要です．これらの課題をまとめると，

(a) 仕様の確定と変更の課題
ハードリアルタイムの振舞いを保証するには，設計段階のレビューから積み上げて行きます．仕様の変更があると，積み上げたレビューやテスト成果をリセットして再度積み上げが必要になるなど，影響が大きい問題です．コードのバグ修正でも問題が生じますが，制御仕様の変更となると大きな影響が生じます [3].

(b) タスク間の影響課題
周期タスク間は強い相互影響があり，その設計とテストについては厳密な対策を準備する必要があります．一方，非確定性タスクも RTOS では汎用 OS が持つ保護機能を持たないため，隠れた影響を及ぼすリスクがあります．これら隠れた影響を含めた問題です．

(c) クロス開発の課題
クロス開発は，ターゲット環境が 1 つであってもテ

ストの手間がかかります．複数の ECU に対応させるとなると，テストは個別にその数だけ設計を含めリソースが必要となる問題です．近年では開発期間の問題がクローズアップされています．

(d) 人的資源のナレッジ課題
組込システムのテストや設計に要する人的資源は，量的工数ではなく保持するナレッジです．製品固有のナレッジは，チームや組織で歴史的に積み上げたものであり，属人性より属職場性です．新たな技術を導入していくには，チームや職場単位でのナレッジアップデートを実現しないと実践が伴わない課題です．

次の章では，(a) の課題である仕様を確定することにより生産性と信頼性を高めたテスト技術について示します．

3. ESoft の仕様を確実にする技術

1990 年代から家電製品など幅広く ESoft が使われるようになりました．拡大の推進力は書き換え可能な ROM や ECU の価格性能比の進歩など，主にハード技術です．

1990 年代から 2000 年前半にかけて，大規模な組込システムが導入され，自動車のエンジン制御や航空宇宙の分野で多くの開発が行われました．この過程で大きな混乱が生じました．

3.1. 危険なコードと MISRA-C の誕生

1990 年代初頭に英国ローバ社で MISRA-C が開発され，危険な C 言語対策のコーディング規範が作られました．車両システム開発で遭遇した周期タスクに絡むバグの分析から，テストでは検出が困難なコードがまとめられています [4].

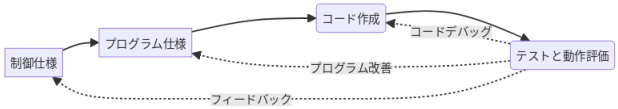

図 4: リアルタイム制御開発の混乱

エンジン制御などリアルタイム制御システムの開発例を図 4 に示します．制御仕様からプログラム仕様を作成し，コード作成を経てテストと評価を行います．このライフサイクルでは，統合段階で振舞いのテストと制御の評価を行います．３種類の異なる不具合，すなわちコードのバグ，プログラム仕様の不具合，制御仕様の不具合が存在します．大量のコードのバグで制御仕様の評価がリセットされ，大混乱が生じました．

ローバ社は，この問題解決のためにソフト工学の専門チームを作り，分析の結果をまとめ，その一部が MISRA-C です．MISRA-C の規範は，一般的な規範ではなく，周期タスクなど特に確定性が必要なタスクが対象です．現在の汎用 OS 配下では型安全やスレッド安全はコンパイラやシステムが対応しますが，組込システムの実行環境にはその種の機能はありません．

3.2. モデルベース開発

エンジン制御は，熱力学や運動方程式を用いた研究分野です．制御の実装方式として E/E アーキテクチャ（電気/電子）を用いますが，制御そのものの仕様は熱力学や流体力学に基づき設計し評価が行われます．

1990 年代の混乱は，コード作成とその不具合が制御モデルを開発し評価するループに入り輻輳を起こし，混乱が拡大しました．制御方程式のような制御要件をソフトウェア実装設計へ変換する原理はサンプリング理論です．サンプリング間隔を RTOS の周期タスク時間間隔とし，制御量を増減することによりデジタル制御が理論的に実現できます．

制御方程式は数値計算を使って制御モデルとして表現できることから，設計を支援する MATLAB 1.0 が 1984 年にリリースされました．その後，モデルから Simulink Coder でプログラム言語を出力する機能が 1994 年に開発され，2001 年には RTOS と連携した自動生成ができる Embedded Coder が開発されました [5].

図 5 はモデルベース開発を概念的に示したもので，モデル設計を中心に設計の評価と改善を行う設計ループです [6], [7]．モデルのアジャイル開発で制御仕様を確定してから自動生成を経て実装の評価を行い，混乱を回避します．

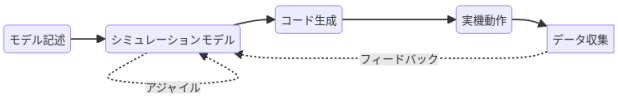

図 5: モデルベース開発の概要

次の課題は，モデルをシミュレーションで評価できて

も，実装されたシステムも評価が必要です．実際に製品をテストする場合，テスト入力として様々なセンサーからの連続的な入力が必要になります．ソースコードだけでなく，テストデータの生成が必要になりました．

モデルが複雑になれば, MiL（Model-in-the-Loop), SiL（Software-in-the-Loop），PiL（Processor-in-the-Loop），HiL（Hardware-in-the-Loop）などの各テスト環境でのテスト入力作成はサンプリング周期も影響し，人手で作成するのは困難です．TPT（Time Partition Testing）方式により，モデルからテストケースを抽出し評価ボードの周期に合わせたテストデータを生成する方式が開発され，Daimler などで実用化されています [8]．図 6 はその概要を示します．

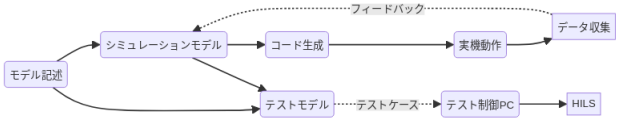

図 6: テストの自動生成の流れ概要

モデルベース開発は，モデルから RTOS への自動生成技術と，HiL や SiL でテストするためのテスト自動生成技術によって完成し，様々な分野で利用されています．

モデルベース開発は，機械工学や熱力学の専門分野における複雑な制御仕様を実装する際に，極力，RTOS や C 言語などソフトウェアの知識への変換を排除することにより，効率的な開発を実現しています．

3.3. 形式仕様による開発

仕様を定義し，その仕様に従って正しく実装されているか．この問いに対する答えの一つがテストです．テストも仕様に従って作成するので「仕様 - 変換 - コード」も「仕様 - 変換 - テスト」も同じ問題を抱えています．

この問題は組込系だけでなく，ソフトウェア共通の問題です．仕様から属人性を除いたコード生成は自動生成と等価です．仕様から自動生成可能な仕様記述言語は，分野別に開発されています．先述のモデルベース開発もその一種です．

変換問題とは別に仕様は正しいか．この問いに対する解決策が必要です．仕様記述言語には仕様の正しさを確かめやすい機能が必要です．モデルベース開発は，制御仕様を制御モデルとしてシミュレーションで評価確認しますが，何をシミュレーションするかはテストケース設計と同様に人手によります．

仕様定義とその証明については，形式仕様の研究として長い歴史があり，多くの成果があります [9,10]．形式仕様は，データの定義，関数の振舞い，それらの集合論的な関係を記述したもので，VDM++，Z 言語，B メソッドなどで実用化されています．

組込システム開発に形式手法を導入した事例として，モバイル FeliCa チップにおける開発が報告されています [11,12]．この事例は，広く利用されているモバイル系製品の開発に形式手法が実際に使われ，現在でも使用されて続けている実用性の高いものです．

一般的なソフト開発では，アジャイル型の開発が多く使われています．速く作って早く確かめる方式で，GUI や事務処理など，利用者が参加して要件の評価が可能な場合に有効な方法です．しかし，組込分野ではこの方式は混乱を生みました．周期タスクによる実行管理は，統合しないとタスクレベルのテストができません．アジャイルで相互影響を限定した，マイクロサービスのような環境で有効な手法ですが，組込システムでは依存関係を分離できないので不向きです．

4. POSIX 系タスクの分離とクロス開発

この章では，第 2 章で示した (b) タスク間の影響課題と (c) クロス開発課題について説明します．

4.1. RTOS の限界と POSIX 系タスク対策

航空宇宙分野や車両分野とは別に，1990 年代半ばから携帯電話端末の開発が始まりました．携帯電話は，アナログ方式から 2G と呼ばれるデジタル方式が採用されてから飛躍的に市場が拡大しました．欧州が先行し，ノキアが GSM 方式 (Global System for Mobile Communications) で世界市場を広げる中，日本では独自の PDC 方式 (Personal Digital Cellular) で開発が行われました．

国内メーカーは RTOS として μ ITRON を用いた開発を進め，市場の拡大と共に多くの機種を送り出しました．市場が拡大すると，電話機能以外に様々なサービス機能が付加され，それが製品競争力となり，熾烈な開発競争が起こりました．

i-mode やメール機能，さらにタッチパネルによる GUI など，RTOS に非確定タスクを大量に詰め込むことにな

り, 混乱が拡大しました．さらに 3G に進むと W-CDMA (Wideband Code Division Multiple Access) への切り替えもあり, Linux 系の RTOS 採用に変わっていきました．

2007 年の iPhone 以降，モバイル系 RTOS は Android に集約されていきました．Symbian や Windows Mobile など競合するモバイル RTOS がありましたが，Android が独占状態です．Android の技術的特徴は，非確定タスクの動作環境を飛躍的に改善し，GUI 系のライブラリなどを充実させています．

ビジネス的には，オープンソース戦略の成功と言われています．2010 年代から，多くの製品が OSS 化しました．

4.2. Android の特徴

Android は，それまでの RTOS とは異なり，メモリ以外に 5GB 以上のストレージを必要とし，Linux と同様のメモリ管理を用い，プロセス単位で独立したアドレス空間を提供します．周期タスク制御など，確定タスク制御の部分と POSIX 系のタスクを完全に分離し，POSIX 系の機能を充実させました [13,14].

統合開発環境 (IDE) として Android Studio が Google から無償提供され，開発からテスト，さらに実機に接続しての評価までシームレスに開発し実行できるようになりました．JDK (Java Development Kit) のサポートや，携帯ハードの機種に依存しない開発環境によって，非確定型の開発が進みました．

公式言語は Java および Kotlin (Google の Java 互換) ですが，JavaScript や C#が一部で使われています．これらの言語は，HW に依存しない VM (仮想計算機) で実行する JIT 型言語 (Just-In-Time Compiler Language) であり，機種依存のないテストができます．つまり，組込システムのテスト技術として特別なものが不要になりました．

携帯電話関連のソフトを含めた現在のビジネス規模は 4 兆ドル (日本の GDP と同規模) です．日本のモバイル産業は 12 兆円程度で，世界比は 2 %ほどです．日本の市場は飽和状態ですが，中国やインドでは年率 8 %ほどの成長が続いています．モバイル系は，ビジネス的にも技術的にも，組込系とは別に扱われるようになっています．

4.3. クロス開発とテスト問題

モバイル/Android は，クロス開発の問題を解決しましたが，対象とする HW の要件は，5GB 以上の外部記憶装置を必要とするなど，組込システムでは例外的な構成です．モバイル系以外の組込システムにおけるクロス開発課題の代表的な対応として，次のアプローチがあります．

(i) Java/Java 仮想マシン（JVM）

Java は 1995 年に公開された組込システム向け言語です．JVM を使って HW や OS などプラットフォーム独立を実現しています．オブジェクト指向プログラミングを強力にサポートします．

NASA が 1998 年から導入を行い，2000 年から公式に取り入れ，大規模開発で成果を上げています．テストの分野では，TDD (Test-Driven Development) の実践や Java Pathfinder が開発され，モデル検査に導入しプログラム検証の自動化を進めています [15,16].

(ii) QEMU/エミュレータ

QEMU は, さまざまなアーキテクチャをエミュレートすることができるオープンソースの仮想化ソフトウェアです [17]．オープンソースとして利用できることから，RTOS 提供側が RTOS の評価環境として提供しています．最近の研究では，異なる ECU アーキテクチャを搭載した SoC (System on a Chip) の検証に QEMU を使った評価が報告されています [18]．実在するほぼすべての CPU をエミュレートできます．

(iii) 統合開発環境/ゲスト側シミュレータ

RTOS の対応可能なプラットフォームの拡大により，開発ホストである x86 を対象としたターゲットを生成できるようになり，テスト用に Windows や Mac 用のロードモジュールを使います．ホスト上の開発支援ツールと連携し，統合開発環境として提供します．高速で実行できるメリットが評価されています．

以上，組込ソフトウェアに非確定タスク，すなわち POSIX 系のタスクを乗せることによる混乱の事例として携帯電話端末の開発を示しました．対策としては Java

や andoroid の仮想計算，テストの方法としては，エミュレータとシミュレータによる仮想化について説明しました．

5. ナレッジアップデートの課題

組込システムが Web 系や Cloud 系とは技術そのものの捉え方に違いがあることを歴史と共に示してきました．この章では，エンジニアのナレッジやスキルの観点から捉えます．組込業界共通の課題として，エンジニアの育成や流動性の問題が認識されていますが，対策が不足しています．次の問題について考えます．

(A) 属職場，属製品ナレッジの問題

(B) 無知の誤り問題

(C) 技術異文化交流の問題

5.1. 属職場，属製品ナレッジの問題

ソフトウェア工学やテスト技法などは，属人性問題（工業製品なのに人によってばらつきが大きい問題）を解決するために，方法論やツールを提供しています．属職場，属製品とは，その集団版であり，組込開発でよく見られます．発生する理由は，個別最適化が必要で製品寿命が長いなどですが，エンジニアのスキルから見ると，当該職場での OJT で形成されたナレッジが占有しているのが原因です．

製品寿命が何十年も続くことはないので，エンジニアの流動性が問題になります．特に，伝統的な企業になるとエンジニアリングのナレッジが旧世代で埋め尽くされ，技術的失速に陥ります．

世界は，OSS など共有された技術（オープンサイエンス）やツールへ移行し解決していますが，それも一つの解決策です．新興企業の方が新しいナレッジを獲得しやすくなっています．

5.2. 無知の誤り問題

物理学では，法則があり，そこから演繹的に様々な現象が説明できます．しかし，ソフトウェア工学において，法則で演繹的にソフトウェアやその開発を正確に制御することは不可能です．知識人の習性として，未経験の状況においては，自身が正しいと思い込んでいる法則から，演繹的に判断することがあります．

例えば，ウォーターフォールよりアジャイル，C より C++，実装より要件定義，統合テストより単体テスト充実やレビューなどです．「しっかり頑張れば，多少の問題が出てもできる」など過去の経験からエンジニアとしてのナレッジを持っています．

ところが，組込システムでは，未経験の技術異文化と遭遇します．例えば，「ADAS でレベル 3 の交差点のシーンで対向車が...」この認証を確実にするのは？など，別の専門分野との接点では，何が正しい進め方なのかが，既存のナレッジから決められません．

リーダー的立場では決定を下さなければならないが，知らない部分で問題が生じます．この種の問題は「無知の誤り」と呼ばれます．原因は，根拠のない自信とナレッジ不足です．世界初の状況に遭遇するわけではないので，先行事例を調べれば予防ができます．

5.3. 技術異文化交流の問題

ソフトウェアエンジニアのナレッジ問題の根底には，均一なスキル集団での経験がナレッジを作り上げている現状があります．他の分野のエンジニアとの技術コミュニケーションが不慣れです．

例えば，モデルベースで流体熱力学のエンジニアと打ち合わせる場合，彼らの仕事の進め方や基本用語を理解すること，そして彼らにソフトウェアでの制御の仕組みや仕事の進め方を理解してもらう必要があります．

この問題は，(A) および (B) と関連しており，結果的に継続的なナレッジのアップデートが必要です．この現象は，ソフトウェアに限らず，ナレッジワーカーにとって共通の問題です．

ナレッジワーカーと対比する肉体労働の英語名はマニュアルワーカーです．「職場のマニュアルや OJT でナレッジワーカーを育成できるのか」考えなければならない課題が多くあります．

テストエンジニア，ソフトウェアエンジニアのスキルやナレッジの問題を挙げました．業界における対策は，自律的努力です．背景に雇用の流動性があります．日本のように長期雇用の強い業界では，技術者の「タダ乗り現象」として自律的努力をしなくても評価に影響しないなどの雇用慣習が影響しています．

6. おわりに

組込ソフトウェアのテスト技術は，ニッチな分野で全体が見えにくく，組込関係者以外から注目されることが少なく，技術的に注目されていません．

本稿では，組込ソフトウェアのテスト技術を整理し，課題と共に紹介しました．第 1 章では，組込システムの目的が「モノの制御」であり，その要件である振る舞いの「確定性」を実現するための仕組みとして周期タスクを中心に説明しました．その上で，次の 4 つの課題を挙げました．

(a) 仕様変更の影響課題

(b) タスク間の影響課題

(c) クロス開発の課題

(d) ナレッジアップデートの課題

第 3 章では, (a) の課題を解決した事例として, MISRA-C 誕生の意義，モデルベース開発，形式仕様について示しました．第 4 章では，大きな混乱が生じた携帯電話の事例と，その解決策となった Android の仕組み，広範囲で活用が始まった仮想化技術について説明しました．第 5 章では，エンジニアのナレッジ側面から問題を整理しました．

組込ソフトウェアのテストは，「モノの制御」をコンピュータで実現する場合の評価やテストであり，技術異文化交流が必要な境界分野に位置します．組込システムは今後も新たな境界分野を拡大し，未経験の問題に遭遇することが予想されます．その問題整理や人材育成に活用されれば幸いです．

参考文献

[1] 久住憲嗣. 特集 「組込みシステム工学」 の編集にあたって. 情報処理学会論文誌, Vol. 65, No. 2, p. 538 – 538, 2024.

[2] Ashif Mohammad, Rimi Das, Md Aminul Islam, and Farhana Mahjabeen. Real-time operating systems (rtos) for embedded systems. *Asian Journal of Mechatronics and Electrical Engineering*, Vol. 2, No. 2, p. 95 – 104, 2023.

[3] 一騎井口, 恒夫中西, 憲嗣久住. 派生開発のためのリバースエンジニアリング管理手法. 研究報告ソフトウェア工学（SE）, Vol. 2021-SE-208, No. 18, p. 1 – 8, July 2021.

[4] Roberto Bagnara, Abramo Bagnara, and Patricia M. Hill. *The MISRA C Coding Standard and its Role in the Development and Analysis of Safety- and Security-Critical Embedded Software*, Vol. 11002, p. 5 – 23. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[5] A brief history of matlab. https://www.mathworks.com/company/technical-articles/a-brief-history-of-matlab.html/. (Accessed on 05/17/2024).

[6] Jeff A. Estefan. Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group*, Vol. 25, No. 8, p. 1 – 12, 2007.

[7] Manfred Broy, Sascha Kirstan, Helmut Krcmar, and Bernhard Schätz. *What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?*, p. 343 – 369. IGI Global, 2012.

[8] Eckard Bringmann and Andreas Krämer. Model-based testing of automotive systems. In *and Validation 2008 1st International Conference on Software Testing, Verification*, p. 485 – 493, April 2008.

[9] 中島震. オブジェクト指向デザインと形式手法. コンピュータ ソフトウェア, Vol. 18, No. 5, p. 499 – 528, 2001. Publisher: 日本ソフトウェア科学会.

[10] 中島震. ソフトウェア工学の道具としての形式手法. ソフトウェアエンジニアリング最前線, 近代科学社, p. 27 – 48, 2007.

[11] 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 (情報システムの信頼性・安全性). 日本信頼性学会誌 信頼性, Vol. 31, No. 6, p. 394 – 403, 2009.

[12] 中津川泰正, 栗田太郎, 荒木啓二郎. 実行可能性と可読性を考慮した形式仕様記述スタイル. コンピュータ ソフトウェア, Vol. 27, No. 2, pp. 130–135, 2010. Publisher: 日本ソフトウェア科学会.

[13] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, and S. Dolev. Google android: A state-of-the-art review of security mechanisms. No. arXiv:0912.5101, December 2009. arXiv:0912.5101 [cs].

[14] Nisarg Gandhewar and Rahila Sheikh. Google android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, Vol. 1, No. 1, p. 12 – 17, 2010.

[15] Klaus Havelund and Thomas Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer (STTT)*, Vol. 2, No. 4, p. 366 – 381, March 2000.

[16] Matteo Ceccarello and Oksana Tkachuk. Automated generation of model classes for java pathfinder. *ACM SIGSOFT Software Engineering Notes*, Vol. 39, No. 1, p. 1 – 5, February 2014.

[17] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX annual technical conference, FREENIX Track*, Vol. 41, p. 10 – 5555. California, USA, 2005. Issue: 46.

[18] Juha-Mikko Aho. Inter-processor communication in virtualized environment, December 2023. Accepted: 2023-12-14T11:12:02Z Journal Abbreviation: Prosessorien välinen kommunikaatio virtuaalisoidussa ympäristössä Publisher: J.-M. Aho.