XXX
2024/03

Weekly コラム 571 回
【再利用可能なソフトウエアの開発 – SW の資本回転率を考える】

筆者：宗像 尚郎

1. Embedded Computing とは

私たちは「Embedded Compute の領域で世界 Top3 となる」ことを目標に掲げていますが、今回は「Embedded Compute」とは何か、わざわざ「Embedded」と前置きしている意味は何か、そして私たちが追求すべき「Embedded」の課題と勝ち筋について考えてみたいといます。

1.1 Embedded = 埋め込み（組み込み）ではない

日本語では通常「Embedded」は「組み込み」と翻訳され、この言葉は「組み込み機器業界」など広く一般的に使われているのですが、「組み込み」という言葉からマイクロコンピュータで制御された「比較的規模の小さな電子機器」が想起されるのではないでしょうか。しかし、この類推が必ずしも正しくないことを最初に指摘します。「Embedded（組み込み）」の本質は「専用の目的を持った」であって、その対義にあるのは「汎用的に利用できる」コンピュータです。すなわちパーソナルコンピュータのようにユーザーがインストールしたアプリ次第で用途は様々に変化できるコンピュータをさしています。図 1 に「Embedded」と「PC」の違いをまとめました。この分類に従えば、例えばスマートフォンは「Embedded」には分類されないことになります。

	汎用コンピュータ	専用コンピュータ
代表例	PC、スマフォ	家電、産業機器
用途	汎用	特定用途
アプリ	ユーザーがインストール	プリインストール（変更不可）
操作	マウス、キーボード、タッチ	専用物理スイッチ
拡張性	あり	なし

図 1：PC（＝汎用） と Embedded（＝専用） の比較

では特定用途向けに使われる「Embedded Compute」にはどのような特徴があるでしょうか？ 最初に気付くのは HMI（=Human Machine Interraction）、つまり機器の操作方法に違いです。キーボードやタッチパネルを使って機器を操作する PC やスマートフォンとは異なり「Embedded」機器には機能ごとに専用のスイッチがあります。そう考えると大型のタッチパネルで様々な機能を呼び出すようになった最近の自動車は、もはや「Embedded」の範疇ではないのかもしれません。もちろん「Embedded」に求められるのは HMI だけではなく、以下のような特性を備えることも必要でしょう。

1）信頼性／可用性（長時間連続して動作する、途中でリセット／リブートは許されない。また故障の原因となりやすい冷却ファンなどの機械部品もできれば使いたくない）

2）省電力性（原則連続運転なので、電力消費を抑制したい）

3）ネットワークへの接続（LAN、公衆網）、不揮発性のデータストレージ対応

4）保守性（遠隔故障診断、部品交換の容易性、最近では SW の遠隔更新など）

5）プライバシー／セキュリティ（個人が特定できるデータが流出しない、金銭が盗まれない、等）

1.2 大規模な Embedded System もある

「マイクロプロセッサーが使われている機器が Embedded である」とは言えない実例を示します。ここでは銀行の現金預け入れ引き出し機（ATM）と商店で売り上げの管理集計を行う POS システムに注目します。これらが「専用コンピュータ」なのは明らかですが、ATM や POS にどのようなハードウエア、ソフトウエアが使われているかご存じでしょうか？ 実は大部分の ATM／POS の基本ソフトウエアには Microsoft Windows が使われています。ということから、実質 PC そのものが組み込まれていて当然制御用プロセッサーは Intel/AMD の汎用プロセッサーが使われているのです。ATM／POS を制御する Windows は PC で我々が使っているのと実質同じものです。このため 2020年1月に Windows7 のサポートが終了した時に稼働停止になった ATM もありました。Microsoft の組み込み機器向けの Windows の変遷を参考に、ルネサスが標榜する「Embedded Compute」と PC 向けの汎用プロセッサーにどのような境界があるか考察してみましょう。

1.3 Windows Embedded の類型

図２は Windows の組み込み向けバージョン Windows Embedded の変遷を示したものです。これを見ると過去には Windows Embedded には２つの系統があったことがわかります。「あった」と過去形で語っているのは Microsoft は既に WindowsCE を起点とする「Embedded」向けの製品群の提供を止めてしまったからです。現在では PC 互換ハードウエアを採用する ATM／POS 向けの高信頼バージョン（Windows 11 IoT Enterprise）だけを提供していますが、既に紹介したようにこれらは PC 用の Windows と実質的に同じものです。以前あった「Embedded」シリーズには用途特化バージョンが展開され、カーナビなどで一定の成功を収めた時期もあったのですが、これらは既に Linux や Android に置き換えられています。何故 Microsoft は「Embedded」ビジネスから撤退してしまったのか、何故成功できなかったのか、そして今後ルネサスが主戦場とする「Embedded」で成功していくには何が必要なのか考察してみましょう。

図２：Windows Embedded 製品の変遷

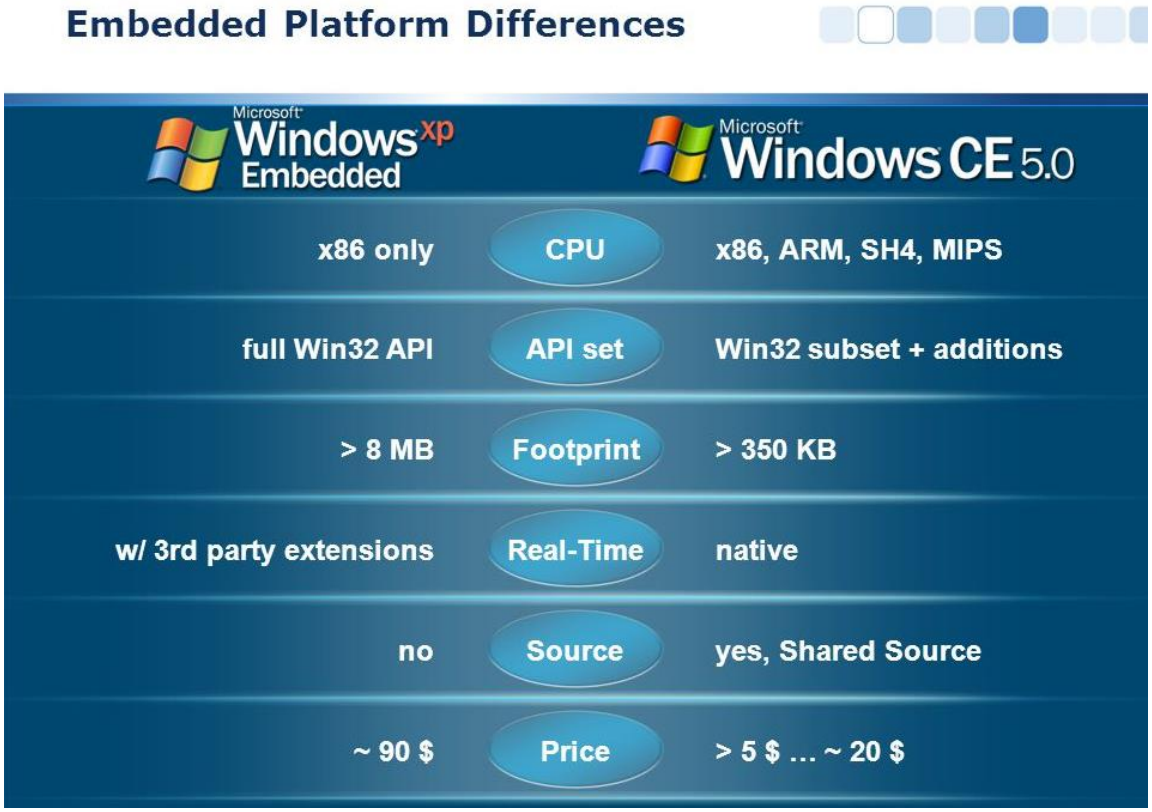
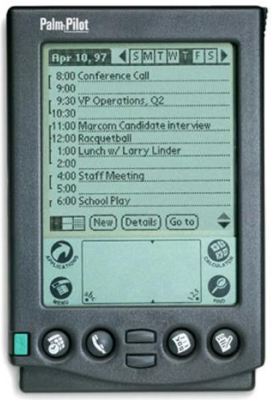


図３：Windows Xp Embedded（フル機能版）と Windows CE（軽量版）の違い

1.4 PC 陣営による「NetBook」ムーブメントを振り返る

Microsoft の「Embedded」ラインの起点となった WindowsCE (CE は Consumer Electronics の意味）は1996年に発表され、カシオのカシオペア（CPUは日立 SH3）、HP Jornada（同 SH3）、NEC のモバイルギア（NEC VR4121）などの「ハンドヘルドPC」に採用されました。「ハンドヘルドPC」は、当時流行していた PDA（＝Personal Digital Assistant）と呼ばれた小型情報端末より高機能な「簡易 PC」 を目指した新しいカテゴリーの製品でした。そのためキーボードとタッチパネルが搭載され、簡易版の Excel や Word もプリインストールされていました。Intel は Microsoft が提案した「簡易 PC」を狙って Atom シリーズと呼ばれる低価格プロセッサーを発表し WindowsCE 系列の OS を組み合わせてデジタルテレビなどの家電（Windows CE)、スマートフォン（Windows Mobile）、自動車のインフォテイメント領域（Windows Automotive）への参入を試みています。



PDA

Handheld PC (Windows CE)

NetBook (Windows Xp)

しかし、これらの Intel の試みは成功しませんでした。同時期に Microsoft は Atom を使った UMPC (=Ultra-Mobile PC) 規格に準拠した「ネットブック」を発表しました。「ネットブック」は NotePC のサブセットという位置づけで、Microsoft はフル機能のブラウザーや標準版 Excel などのビジネスアプリケーションも動かせるとアピールしました。その根拠は「ネットブック」には PC 用の Windows XP がそのまま搭載されていたからです。しかし Atom と PC用のメインストリームプロセッサーとの性能差あまりにも大きく「ネットブック」でビジネスアプリケーションを動かしても絶望的な性能しかだせず、ユーザーを満足させることはできませんでした。結果として Intel や Microsoft などの PC 陣営が「Embedded」領域に参入する試みはことごとく失敗に終わったのですが、ではどうして当時の家電製品やゲーム機ではこのような性能面の不満がでなかったのでしょうか？ それはソフトウエアが当時のマイクロプロセッサーでも十分な性能を発揮できるように作られていたから（WindowsCE のような冗長な SWになっていなかったから）なのですが、何故 Microsoft ができなかったことが可能だったのでしょうか？

2 Embedded の特性と課題

2.1 暗黙の仮定に依存した最適化

PC と比較すると「Embedded」には以下のような制約があります。

1）マイクロプロセッサーの処理能力は相対的に低く、利用可能なメモリー容量も限られる
2）専用機なのでコスト競争も厳しく、性能余裕や拡張性を持たせることができない
3）通常プログラムは ROM 化されているので、出荷後にプログラムを書き換えるのは困難

ほぼ PC そのものが組み込まれている ATM などを例外とすると「Embedded」で利用可能な計算資源は極めて限られているので、そのような窮屈な環境でも動作可能なソフトウエアを開発しなければならないという宿命があります。このようなプログラミングの前提には「Embedded = 専用機」なのでユースケースを特定（限定）することが可能で、その要件に対してのみ動作を保証するプログラムを開発すれば良いという考え方がありました。「データーフォーマット」、「イベント発生頻度」、「通信速度」などが一定範囲であると仮定できればプログラムを簡略化にできるし、「入出力チャンネル数」、「画面サイズ」、「メモリー容量」などのハードウエア要件を決め打ちにすれば機能削減も可能だからです。
「Embedded」ではこのような「要件ファースト」が当たり前だったのですが、ここは一つのプログラムで色々な用途やハードウエアをサポートする前提で設計されていた WindowsCE とは設計思想が根本的に異なることに注目する必要があります。

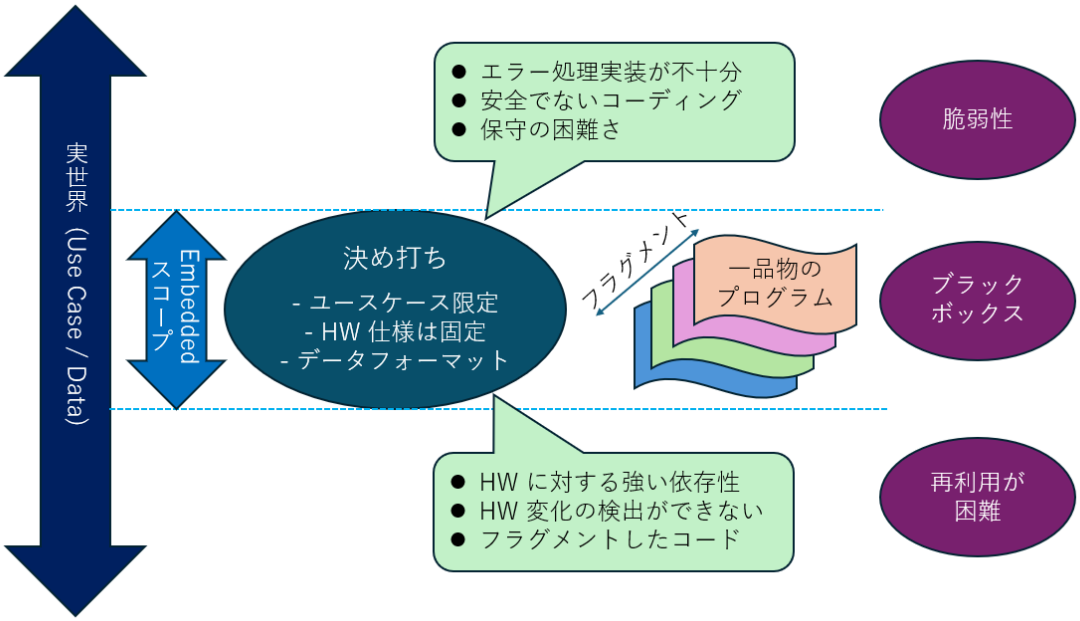
2.2 再利用性

通常「Embedded」ではハードウエアは先に決まっているので、プログラムで「I2C インターフェースが何チャンネル利用可能か」とか「メモリーはどのくらい搭載されているか」とかを検出（プローブ）する必要はありません。ここは、後から周辺機器を追加した時でもそれを自動検出する Plug&Play 機能に対応しているパソコンとは考え方が違う部分です。「ハードウエアの決め打ち」をすればプログラムはシンプルにできるのですが、半面ハードウエア構成が変わってしまうとソフトウエアが動作しなくなるというネガもあり、結果として大部分の「Embedded」のソフトウエアは再利用ができない一品物となっているのです。プログラムの規模が小さかった時代にはこれでも良かったのですが、状況は大きく変わっています。 近年は「ソフトウエア規模の拡大 ⇒ 毎回ゼロからのスクラッチ開発では時間と費用がかさみ、品質もあがらない ⇒ 製品の競争力が無い ⇒ 開発リソース確保が制約条件となりビジネスが拡大しない」という負のスパイラルに陥っているのです。この状況を打開していくために「Embedded」領域でも「再利用可能なソフトウエア」の開発が喫緊の課題 になっているのです。

2.3 脆弱性

「プログラム（＝論理設計）に暗黙の前提への依存性が含まれる」のは「Embedded」制御用ソフトウエアだけの問題ではありません。「閉じたシステム」（＝外部入力が無い自己完結したシステム）では問題

は顕在化しませんが、機器が「ネットワーク」に接続されたり「メモリーカードなどの記録バイスがサポート」されたりするようになると様々な問題が露呈しました。ひとつは単純にプログラムがハングアップして動かなくなる「誤動作」です。10年ほど前までは、パソコンで作業中に突然プログラムがフリーズしてそれまでの作業が無駄になることも珍しくありませんでした。 多くの場合、想定外のデータに対する「エラーハンドル処理」が十分考慮されていなかった結果プログラムがハングアップしたのです。そしてこのような単純な「誤動作」以上に深刻なのが故意に想定外の入力を与えてシステムをわざと停止させたり、ユーザーデータを改ざんしたり盗み出そうとするハッキングです。前者が一定時間内に大量の接続要求を送りつけてシステムが動けないようにする「DoS（＝Deny of Service）攻撃」、後者はプログラムが用意したデータ受信用領域より大きなデータを送りつけて Web サイトを改ざんしたり任意のプログラムを動かしたりする「バッファー オーバーフロー」攻撃などです。近年このような「論理の脆弱性」を狙った攻撃（サイバーテロ）は、莫大な経済損失が発生させる大きな社会問題になっています。この対策としてプログラム検証時に「わざと規格外（＝想定外）の入力を与えて、システムが誤動作しないか検証する」ファズテスト（Fuzz Test）が広く推奨されています。これは、例えば「動画」が入力されると想定しているプログラムにわざと「静止画像」を入力してみるといったテストです。以前なら「規格外のデータを入力したユーザーが悪い、そのようなケースでは誤動作しても仕方が無い」と言えたかもしれませんが、今ではそのような言い訳は通用しないのです。



2.4 Embedded の再定義

これまで常識とされてきた「ハードは先に決まっていて、その特定のユースケース（だけ）に適用した最小限のプログラムを開発する」という「Embedded」ソフトウエア開発の基本思想には本質的に「再利用

性」「脆弱性」の課題があること。そして近年のプログラム規模拡大のトレンドの中で、これらのネガが大きな問題になっていることを説明しました。ここでは更に「Embedded」の定義自体も変化していることに注目します。以下に最近の「Embedded」で必須となっているシステム要件を列挙します。

１）各種ネットワークへの接続
　１－１）LAN/WiFi/BT/5G Modem などの通信プロトコルへの対応
　１－２）各種ログインに対応する認証方式への対応
　１－３）コンピュータウィルス対策
　１－４）SW Update への対応
２）他の機器とのデータ交換（ファイルシステム対応）
　２－１）SDカード／USBメモリーなどの記録メディア対応
　２－２）Windows PC とのネットワークを介したファイル交換
３）リッチなユーザーインターフェースのサポート
　３－１）グラフィカルユーザーインターフェース（メニュー画面設計）
　３－２）フリック、スワイプなどスマフォ由来のタッチ入力への対応

ここに列挙した機能は従来の「Embedded」システムでは「対応されていない、または限定的にしか対応していなかった」ものですが、最近ではこれらの機能はユーザーに機器が選ぶ時の重要なシステム要件（KBF = Key Buying Factor）となっている重要な機能です。そのためルネサスの「Embedded」ソリューションでも最先端の「コネクティビティ」や「ユーザーインタラクション」への対応が重要なアピールポイントしているわけです。ここで気付いて頂きたいのは、最新の「Embedded」のシステム要件は15年ほど前に Microsoft や Intel などの PC 陣営の企業が「NetBook」で実現しようとして失敗した製品のコンセプトに極めて近いという事実です。2007年に登場した「NetBook」には 1.6GHz 駆動の 32bit プロセッサー Atom N270 プロセッサーが搭載されていた事を考えれば、現代の「Embedded」の方がむしろプロセッサーの能力は高くなっているのです。

近年の「Embedded」のハードウエア性能は約15年前の「NetBook」を追い越していると言いましたが、一方でソフトウエアの作り方は旧態依然なのではないでしょうか？ 今回私は皆さまに以下の３点について考えてみて頂きたいのです。

1. Embedded ソフトウエア開発スキームを根本的に革新する必要があるのではないか？
2. ルネサスがその変革をリードして「顧客に新たな価値を提供」するチャンスがあるのではないか？
3. それはルネサスが Embedded 業界でリーディングポジションを確立するための礎ではないか？

Microsoft は「NetBook」で PC 用の WindowsXP をそこまま動かそうとして、絶望的に HW 性能が足りずに頓挫してしたわけですが、SW の作り方自体は間違っていなかった部分も多かったとも考えられ

ます。最後に我々が「Embedded」の SW 開発を革新していくには何を指標（ベンチマーク）にしていけば良いのか考えてみましょう。

3. 再利用を前提としたSWを開発する

3.1　ソフトウエアを資本と考える

「Embedded」ではソフトウエアを製品毎にゼロから開発するのが一般的で、過去世代の製品に使われていたソフトウエアを流用（再利用）することは殆どありません。結果として「コードを書いたプログラマー本人だけが制御内容を把握している」という状況に陥りやすく、機能拡張やバグ修正のためにソフトウエアの改変が必要になっても本人以外は誰も対応できない事態がおこりがちです。それでも対処が必要になると、元のプログラムはブラックボックス化していて手が出せないので「元のプログラムを迂回する冗長なワークアラウンドを追加する」といった付け焼刃が実施され、更に意味不明なプログラムに変貌していくのです。一般にソフトウエアは「資産」であると考えられています。しかしもしその実体が「誰もメンテナンスが出来ない属人的なコードの積み重なりによって身動きが取れない状況（下図）」だとすれば、それは「不良債権」だと言わざるを得ないでしょう。そして現在「Embedded = 専用機」を開発する企業の多くは肥大化した「不良債権」に苦しんでいるのです。さらに、このように不良債権化したソフトウエアが「重大なバグの温床」になっているという側面も無視できません。最近日本の金融業界で発生している大規模システム障害が不良債権化したソフトウエアに起因しているという話は有名ですが、マイコンを使っている「Embedded」の世界でも状況は殆ど同じです。

このような状況を打開するには

1. 最初から「将来別の用途や開発者によって再利用されること」を想定してソフトウエアを開発する（ハードウエアやユースケースの前提条件を排除する）
2. ゼロからの新規開発に着手する前に、再利用可能な過去資産が無いか調査し、あれば最大限それを活用する
3. 過去資産をベースに拡張を行った場合には、その変更内容を元のコードにフィードバックする（コードの枝分かれを回避する）

といった根本的な戦略変更が必要です。本来「再利用可能なソフトウエア」は、次世代製品を生み出すための種（シード）なのです。「資産」は過去の開発成果の蓄積であり保全の対象ですが、成長の源泉となるソフトウエアは「資本」だと考える必要があります。「Embedded」で今後必要なのは「ソフトウエアを資本と考える」発想ではないでしょうか。

3.2 再利用可能な SW を開発する

「資本として再利用可能な SW」を実現するには、（少なくとも）以下に挙げるような項目について考える必要があります。本稿は技術解説ではないので各項目の詳細には言及しませんが、どのテーマも正しい技術的理解と継続的な運用が必要なトピックであり、一朝一夕に実現できる内容ではありませんが、これらを一つ一つ実践していくことによってのみ「Embedded」ソフトウエアの「資本化」が実現されるのです。

1. 全ての暗黙な前提条件の排除
2. 処理の抽象化（モジュール化、カプセル化、オブジェクト化）
3. コーディング コンベンション（＝規約）の統一
4. 完全な変更履歴の保全（誰が、何時、何のために、何を変更したかを残す）
5. 社内共通コードリポジトリ（＝検索可能な保管場所）の整備
6. フラグメントの排除（変更を 5 の共通リポジトリに集約し SW の亜種を増やさない）
7. ドキュメントの整備（与件知識が無くても理解できる記述とする）
8. SBOM（= SW 部品表）を整備しコードの出処を明確化する

「再利用可能な SW」の開発メソッドを考える上で参考になるのがオープンソースの開発で使われている手法です。なぜなら全てのオープン ソースは最初から「再利用されることを前提にしている」からです。具体的なツールの選択は開発プロジェクトによって異なりますが、上記にリストした「SW 資本化」の必須要件が具現化されているので、「自社内だけ」「自社とお客さまの間」といったクローズな範囲内で「再利用可能な SW」の開発を目指す場合でも参考になる情報が得られます。

3.3 資本回転率で SW を評価する

最後に「再利用可能な SWの開発」が実際にどの程度具現化されているのかを計測する指標について考えてみましょう。 財務分析では「総資本回転率（Capital Turnover Rate）」という指標が使われていますが、基本的な考え方は「売上高 ÷ 総資本」です。これを応用して「SW の資本化」という本稿の趣旨に当てはめると「製品コード総行数 ÷ 再利用コード行数」を計算することによって「SW 資本化」の程度が数値化できそうです。実はソースコードや論文の盗用を検出する目的で既に「ソースコードや論文の類似性」を計算するツールが色々と開発されています。ソースコードのコーディングコンベンションが統一されていれば、これらのツールを逆に使って「どの程度有効に SW を再利用できたか」を可視化することが可能でしょう。SW業界では伝統的に「開発したコード行数」とか「開発にかかった人月」をもとに費用（＝価値）を計算してきましたが、「SW 資本」の考え方では「どれだけ多くの人に使ってもらったか」だけが評価の根拠となります。アカデミアでは「論文被引用数」が使われていますが、同様な考え方を「Embedded」 SW 開発に導入するために「SW Capital Turn Over Rate」という指標の計算方法を確立して、業界にパフォーマンスをアピールしていくのが面白いかもしれません。

本稿では「Embedded」領域の覇者となるには SW 開発スキームを根本的に変えて「ハードウエアターゲット毎の一品物のSW開発」から「再利用可能な SW 開発」に舵を切る必要があること。その為には、以前 Microsoft や Intel などの PC 業界が「NetBook」で具現化しようとした SW のアーキテクチャの良かった部分も参考にして、最新の大規模 SW 開発の手法を適用していく必要があること。その時には「最初から再利用されることを前提に開発されているオープンソース」の開発で使われているメソッドが活用できることを紹介しました。これまでは「SWは資産」であり保全するものという考え方でしたが、これからは「SW はビジネスを拡大するための資本」と考え、その回転率を評価することで競争力を高めることができるのではないか、という私論を紹介しました。ハードウエア優位性だけでは「Embedded」業界の覇者となるのは難しいので、ソフトウエア領域でのアドバンテージについて継続して検討していく必要があるのは間違いないでしょう。