[B6] SW First の実践は一筋縄ではいかない、「よくある阻害要因」の分析と「解決への指針」

Design Solution Forum 2023

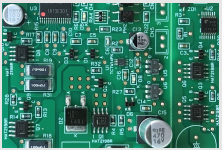
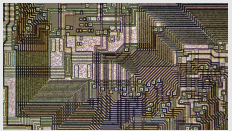
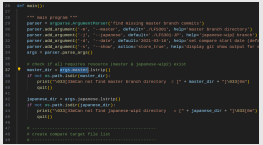
宗像尚郎

ルネサスエレクトロニクス株式会社
ハイパフォーマンスコンピュート ソリューショングループ
シニアダイレクタ

2023-11-22

「SW First」という開発メソドロジー

「SW First」がメガトレンドになった背景 を再考する

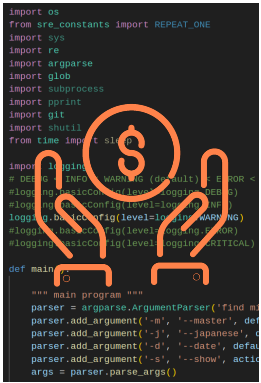
	2010 年以前	最近まで	今後
主な **差異化要因**	PCB基盤上の回路	SoC	SW
差異化技術の **帰属**	最終製品開発メーカー	半導体ベンダー	SW プラットフォーマ
主な **技術要件**	堅牢性	機能の集積	移植性（再利用性）
イメージ			
解決すべき **課題**	機器間ネットワーク 経年劣化対策	処理性能 消費電力（発熱対策）	SW 更新 サイバーセキュリティ

背景には、製品に差異化をもたらす主要因が SW に移行してきた事実がある

「資本としての SW（SaaC=SW as a Capital）」とは？

ソフトウエアによって生み出される「新たな価値」とは？

- 顧客のニーズへのきめ細かい対応（アプリの高頻度更新）
  - SW アップデートによる UX の持続的改善
- 最先端のデバイス（スマートフォンなど）への対応
  - SW 更新による 機能アップデート
- サイバーセキュリティなどの 脅威への対処
  - セキュリティ アップデート の適用
- 利用量に応じたサブスクリプション型の費用回収
  - 高精度ロギング機能（＝会計機能）
  - 柔軟なクラウド連携（＝ App ストア対応）



自動車など多くの製造業は「価値の源泉が SW に移行している」ことに気付いている

現在「SW First」は、２つの異なる意味で使われている？

SW 開発の先行着手 (=Shift Left)

- 実ハードの入手を待たずに
  - SW 開発に 先行着手 できる
  - 精度の高い 性能見積もり ができる
  - システム動作の 結合検証 ができる
- 仮想開発環境 に求められる要件
  - HW エミュレーション 環境
  - クラウド 上の計算資源の活用
  - 実 HW との結合 (HILS)

既存 SW 資産の尊重（＝再利用の推奨）

- 製品毎にスクラッチ開発をしない
- 再利用可能な既存コード を探索する
- 分散開発の仕組み（git 等） の活用
- システムの 検証時間 の短縮
- 検証済コードの利用による
  - 品質の確保
  - SW 生産性の向上

「SW First」の本質は「SW が利益の源泉（＝つまり資本）である」という考え方

同じコードも、捉え方の違いで "資本" にも "資産" にもなる

資産（Asset）··· 静的なもの、結果

- これまでに獲得したもの（開発成果）
- 時間と共に蓄積されていく
- 基本的には 資産の保全 を考える
- 時間経過に伴って 利用価値が逓減 する
- そのため、現在価値の維持は困難
- 属人性の高い資産 の場合、人が抜けると誰も手が出せなくなってしまう
- 結果として 容易に不良資産化 する

資本（Capital）··· 価値を生みだす源泉

- 将来価値（＝利益）の源泉
- 成長戦略のコアとなるもの
- 保全ではなく、回転（再利用）させる
- 現在価値を拡大させる再投資 が重要
- 資本への再投資で、将来価値を高めることが可能である
- 拡大再投資の具体的にシナリオを考えることが 経営戦略 の本質

伝統的な経営観では、経営資源（人、物、金）を保全すべき「資産」と考えていた

「資本回転率」は事業のパフォーマンス評価パラメータ

「資本回転率」を使った SW First 達成度評価 を試みる

- 資本回転率（会計学の専門用語）とは
  - 資本回転率 = 売上げ高 ÷ 総資産
  - 企業が持つ 総資産（人、物、金）がどの程度売上げに寄与したか
  - 社員をコストではなく人的資産（＝人財）と考えるのは最近の考え方
  - 業種によって異なるが 0.3 〜1.7 程度 の値になる

- SW 開発における コードの再利用率を「資本回転率」 と考えてみる
  - SW First とは、最小源の変更で既存のソフトウエアが再利用できる こと
  - これまで 過去開発成果 を「資産」と考えていたが「資本」と見ていなかった
  - 無形のノウハウ（経験値）の集積ではなく、コード自体の再利用を定量評価 する

SW の資本化を実現するには最初から「再利用」を前提とした SW の作りが必要

「組込みシステム」を特徴づけているもの

PC/Cloud 環境

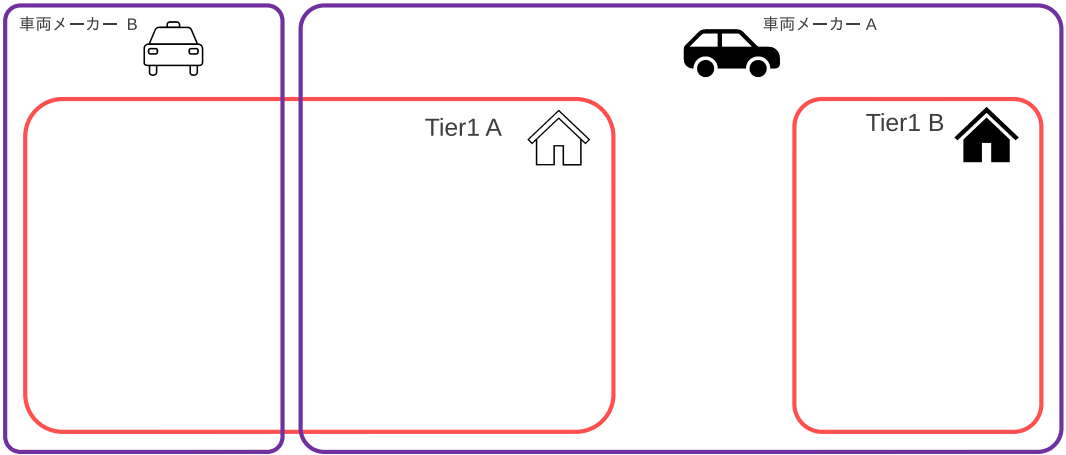
- 標準規格 に準拠した HW の共通性
  - 共通 HW インターフェース
  - 共通 SW インターフェース
  - BIOS が HW 固有部分を隠蔽する
  - 汎用ブートメカニズム
  - 汎用 CPU ＋汎用コンパニオン
- 汎用 OS による HW の抽象化
  - Windows, Linux,..
- HW プロファイリングへの対応
  - 汎用の iso 起動イメージ を利用可能

組み込みシステム

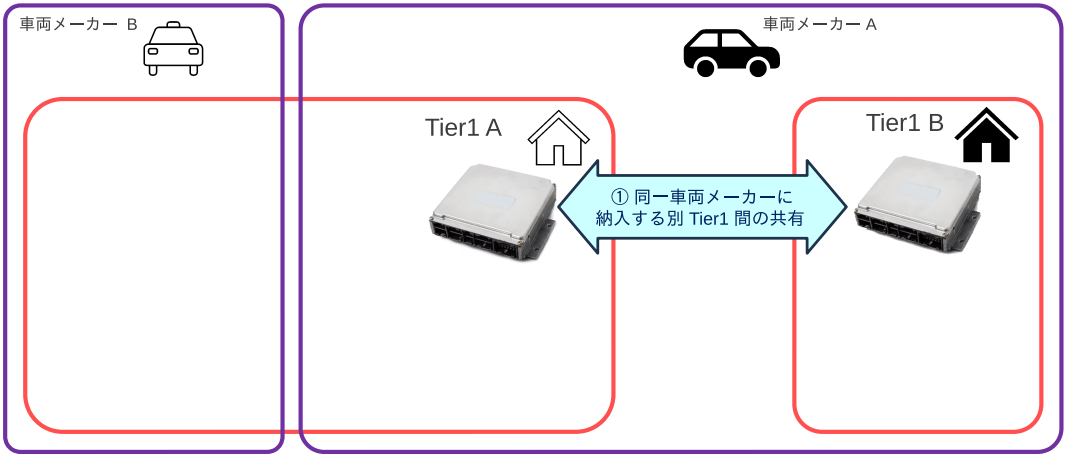
- HW 仕様の 多様性（フラグメント）
  - メモリーマップ
  - ボード毎の 専用起動シーケンス
  - 多種多様な SOC の利用
- 参照可能な 技術情報 が少ない
  - ハードウエア詳細仕様書
  - 汎用 SW 資産
  - 参照可能なブログ記事など
- OS イメージは、各ボード専用
  - Board Support Package (=BSP)

組み込み機器では「SW の資本化」実現へのハードルが、更に一段と高くなっている

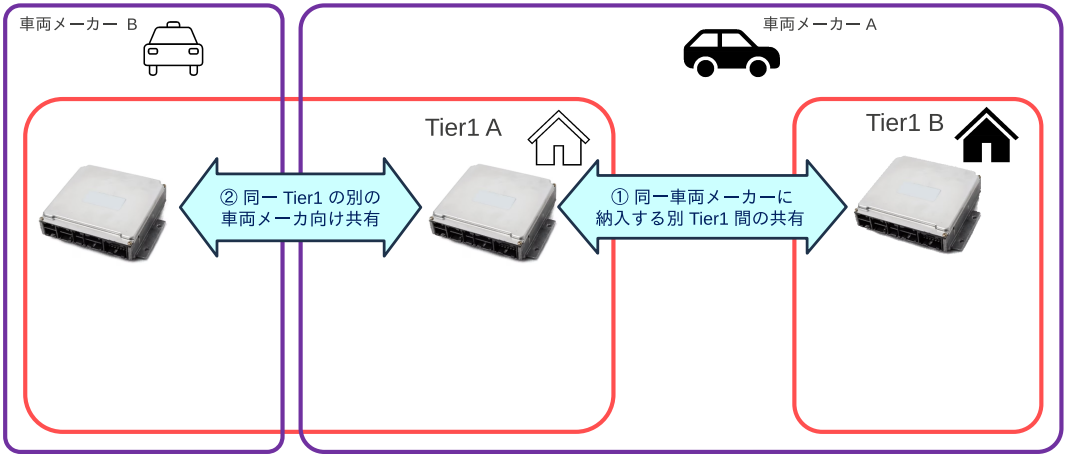
「組込みシステム」における　SW の再利用の類型



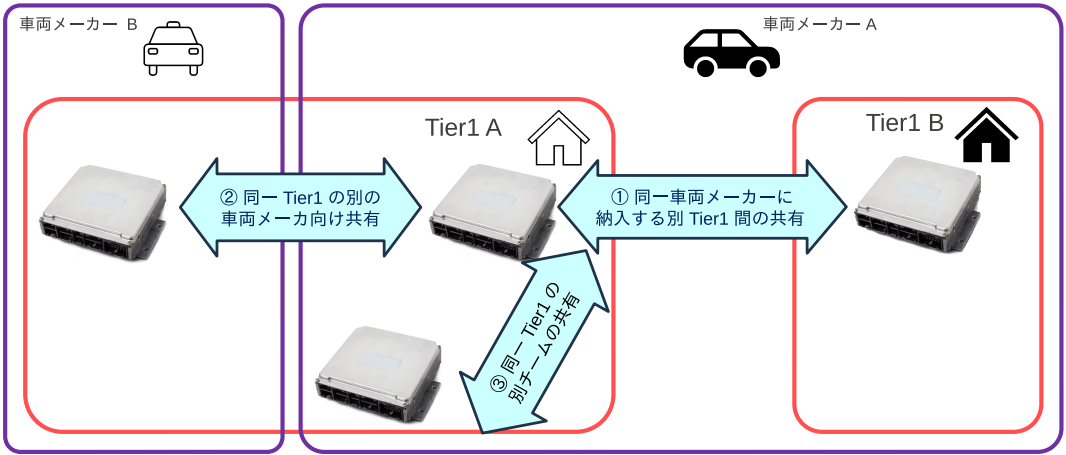
「組込みシステム」における　SW の再利用の類型



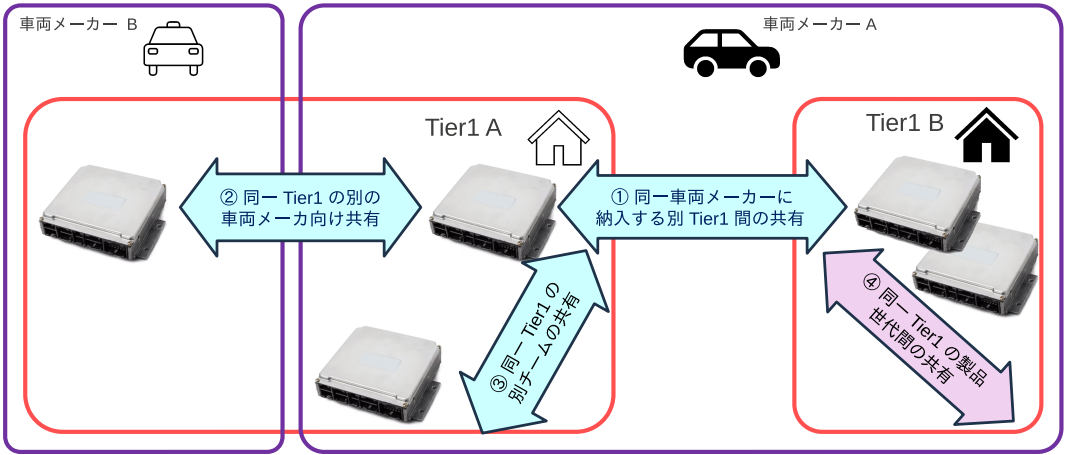
「組込みシステム」における　SW の再利用の類型



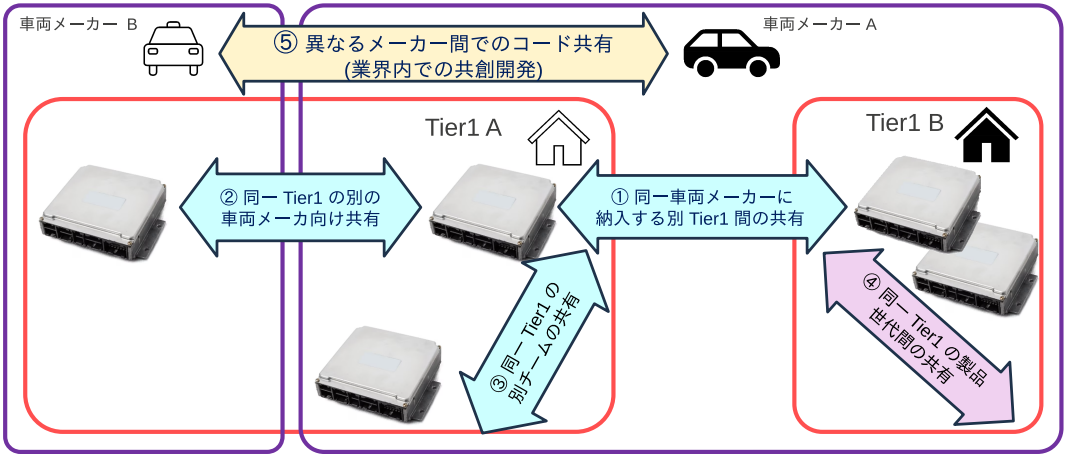
「組込みシステム」における　SW の再利用の類型



「組込みシステム」における　SW の再利用の類型



「組込みシステム」における　SW の再利用の類型



SW First の実践を阻むさまざまな要因

阻害要因（技術）：（1）最適化

これまで是とされた「コードの最適化」が逆に「再利用を阻む仇となっている」

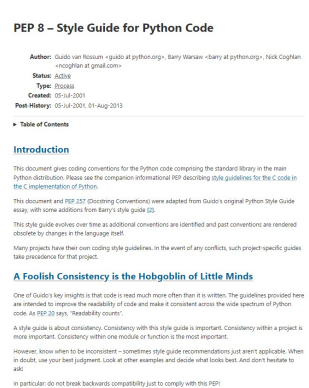
- 特定のターゲットハードウエア仕様を前提とした最適化 を排除する
  - 特定の HW 構成（アドレス、CH 数等）が 暗黙の前提 になってしまう
  - 専用機能（アクセラレータ等）と汎用機能が 分離 されていない
  - 仕向け、グレード別に、類似のボード専用コードが乱立している
- ユースケースの想定 を排除する
  - 最大接続数、データ量（ワークロード）
  - ネットワークの 回線速度（イベントの発生頻度＝割り込み発生数）
  - 故意に 不正なフォーマットのデータ を与えても耐えられるか（= Fuzzing Test）

再利用を前提とするなら、特定の条件を想定した最適化を排除する必要がある

阻害要因（技術）：（2）コードの記述方法

検証済みのソースコードを流用する上での基盤 となる

- コーディングスタイル（インデント、条件文　など）
  - スタイルの統一は ソフトウエア品質に直結 する
  - モジュール分割、共通サブルーチン化ガイド
  - 流用を想定する範囲で スタイルガイドを共有 する
- コメント の書き方
  - 日本語／英語の統一
  - コードとドキュメント記述の対象性
  - Doxygen 対応（コードからドキュメントを自動生成）
- 業界標準の命名法 への準拠
  - 汎用的なデーター名等については標準に準拠する



PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status: Active
Type: Process
Created: 05-Jul-2001
Post-History: 05-Jul-2001, 01-Aug-2013

▸ Table of Contents

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python.

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".

A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

However, know when to be inconsistent – sometimes style guide recommendations just aren't applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask!

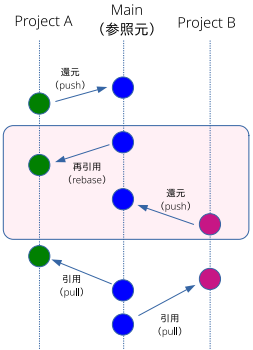
In particular: do not break backwards compatibility just to comply with this PEP!

Linux kernel のフォーマットチェッカー（checkpatch.pl）のような仕組みも有効

阻害要因（技術）：（3）「変更内容の集約」と「トレーサビリティ」

変更内容を参照元コードに戻さなければ フラグメント化 する

- 参照元コードからの ブランチ（引用）とマージ（集約）
  - 参照元となるコード（main）を決める
  - 最新版を 引用（ブランチ）し、独自の変更（拡張）を行う
  - 競合しない事を確認し、参照元にコードを 戻す（マージ）
- 変更履歴の管理（トレーサビリティ）
  - git などの 分散型のバージョン管理システム を導入する
  - 分解可能な最少単位 に変更内容を分解する
  - 変更履歴の 記述ルール を定める（変更理由、変更内容など）
  - ルールに従って変更内容を記録する

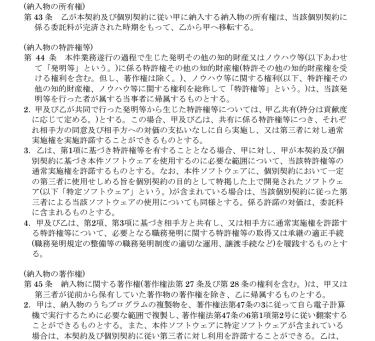


git など、最新のコード履歴管理システムを適切に使いこなせる事が極めて重要

阻害要因（ビジネス）：（4）受託開発における契約制約

既存のビジネススキームが SW 資本化を妨害しているケース

- 成果物の所有権の帰属
  - 原則として 納入物件の所有権は発注者に移転 される
  - ソースコードが納入物件に含まれていない場合もある
  - ソースコードの改変権も契約により異なる
- 著作権
  - 以前から持っていたものを除き、発注者に移管
  - ソースコードの改変 などは著作権の規定による
  - オープンソース が含まれる場合は扱いが異なる



第 6 章　権利帰属

（納入物の所有権）
第 43 条　乙が本契約及び個別契約に従い甲に納入する納入物の所有権は、当該個別契約に係る委託料が完済された時期をもって、乙から甲へ移転する。

（納入物の特許権等）
第 44 条　本件業務遂行の過程で生じた発明その他の知的財産又はノウハウ等（以下あわせて「発明等」という。）に係る特許権その他の知的財産権（特許その他の知的財産権を受ける権利を含む。但し、著作権は除く。）、ノウハウ等に関する権利（以下、特許権その他の知的財産権、ノウハウ等に関する権利を総称して「特許権等」という。）は、当該発明等を行った者が属する当事者に帰属するものとする。
2. 甲及び乙が共同で行った発明等から生じた特許権等については、甲乙共有（持分は貢献度に応じて定める。）とする。この場合、甲及び乙は、共有に係る特許権等につき、それぞれ相手方の同意及び相手方への対価の支払いなしに自ら実施し、又は第三者に対し通常実施権を実施許諾することができるものとする。
3. 乙は、第1項に基づき特許権等を有することとなる場合、甲に対し、甲が本契約及び個別契約に基づき本件ソフトウェアを使用するのに必要な範囲について、当該特許権等の通常実施権を許諾するものとする。なお、本件ソフトウェアに、個別契約において一定の第三者に使用せしめる旨を個別契約の目的として特定した上で開発されたソフトウェア（以下「特定ソフトウェア」という。）が含まれている場合は、当該個別契約に従った第三者による当該ソフトウェアの使用についても同様とする。係る許諾の対価は、委託料に含まれるものとする。
4. 甲及び乙は、第2項、第3項に基づき相手方と共有し、又は相手方に通常実施権を許諾する特許権等について、必要となる職務発明に関する特許権等の取得又は承継の適正手続（職務発明規定の整備等の職務発明制度の適切な運用、譲渡手続など）を履践するものとする。

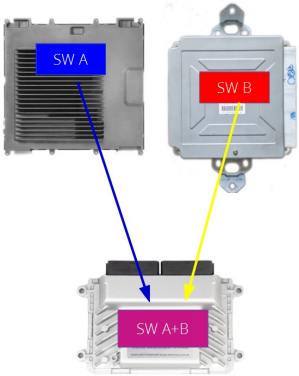
（納入物の著作権）
第 45 条　納入物に関する著作権（著作権法第 27 条及び第 28 条の権利を含む。）は、甲又は第三者が従前から保有していた著作物の著作権を除き、乙に帰属するものとする。
2. 甲は、納入物のうちプログラムの複製物を、著作権法第47条の3に従って自ら電子計算機で実行するために必要な範囲で複製し、著作権法第47条の6第1項第2号に従い翻案することができるものとする。また、本件ソフトウェアに特定ソフトウェアが含まれている場合は、本契約及び個別契約に従い第三者に対し利用を許諾することができる。乙は、係る利用について著作者人格権を行使しないものとする。

受託開発の場合、契約によってコードの再利用に大きな制限がかかる場合が多い

阻害要因（ビジネス）：（5）ビジネス既得権に起因した再利用の制約

組み込みシステムでは SW 境界 ≒ HW 境界 が一般的

- SW 統合 ⇒ 設計開発部門の仕事が減る（社内）
- サプライチェイン（系列関係）が、SW 流通の妨げに
- HW の数が減る方向の SW 再配置 には軋轢が発生する
  - HW 供給者が同じ ⇒ ビジネス規模の減少
  - HW 供給者が異なる ⇒ 一社の ビジネスが喪失
- 既存ビジネスを守りたいバイアス が、SW の流通（再利用）を妨げている場合がある



SW の再利用（統合、再配置）が、既存のビジネススキームと競合する場合がある

阻害要因（ビジネス）：（6）コンソーシアムによる共創活動の成否

OSS 開発コミュニティ = 個人ベース活動

- コミュニティの形成は自然発生的 で、明確な創始者が居ない場合もある
- 活動参加は任意（通常、契約は不要）
- 原則年会費などの 費用負担は無い
- 積極的な貢献が期待される
- 規模が大きくなると、コードリリースサイクル管理、マージルール等の運用ルール の整備が必要になる

産業コンソーシアム = 企業ベース活動

- 創設メンバー (Founding Member)（多くの場合企業）が明確である
- 明文化された メンバーシップ契約 に基づいて活動に参加する
- 参加費や年会費 などの義務がある
- 開発参加には CLA (Contributor License Agreement) が必要な場合も
- 開発リソースコミット (=FTE) の要請

産業コンソーシアム活動は、明確な目的や意思に基づく共創活動であるか次第

目指すべき方向性と、改善への指針

「SW 資本化」の前提となるもの（1）

SW の資本化実現の前提となる 開発の基本戦略

- 再利用を前提とした SW 開発方針 の徹底
  - 汎用性 (=Generality)（特定の前提条件に縛られない）
  - 追従性 (=Trackability)（過去の変更履歴／変更意図がトレースできること）
  - 可読性 (=Readable Code)（適切なモジュール分解、コメント/ログ）
- コードの再利用を支援するための インフラ の導入
  - 全社的にアクセス可能な コードリポジトリ の利用（github など）
  - コード 投稿ルール（構成管理情報、投稿単位など）
  - コード コミットメッセージ 記載ルールの策定と厳格な運用

SW の「資産化」は、徹底したコードの再利用ができることが大前提となる

「SW 資本化」の前提となるもの（2）

SW 開発で生み出される 価値の計算方法 の変更

- これまでは 作業負荷量（＝人月）で価値を計上 していた
  - コード行数
  - 投下時間
  - 自動化、先端 IT 技術活用の程度が低いので 俗人的な管理工数 が発生
- これからは 実際の提供価値 を計算する ← 人月加算より算出方法は複雑
  - コード 再利用率（再利用比率高 ⇒ 検証済み比率高 ⇒ 高品質）
  - コード 被再利用率（＝再利用回数が多いコードほど価値が高い）
  - コードの 適用可能性（＝どれだけの再利用が見込まれる汎用性があるか）

SW 開発が生み出す「資本」としての価値を新たに数値化していく必要がある

最初から「SW 資本化」を目指したオープンソース開発

オープンソース（= OSS）開発の流儀

- Release early, release often（機能ではなく、時間ベースのリリース）
  - 決められた周期（9 週、四半期など）でエンドレスにリリース を繰り返す
  - 結果として 締め切り間際の dirty な修正の混入を回避 している
  - トレース可能な 「小さな変更」を「多数」集積 する
- 完全な 透明性（Transparency）と記録の保全
  - 公開 ML 上での議論（メールアーカイブのデーターベース化）
  - 公開 git 上で変更来歴を管理（コミットメッセージのフォーマットが規定されている）
- プロジェクトの スケーラビリティ（大規模開発に対応）
  - 全世界で数千名の開発者が参加する 多拠点同時分散開発
  - 権限移譲 = Sub-system メンテナー体制

「SW 資本化」は OSS 開発プロジェクトで既に確立されたメソッドを参考にすべき

本日のまとめ

- 近年の電子機器の 魅力の源泉が SW に移行 しており、特に製品購入後の アップデートにより製品の価値を拡大する できることが消費者の期待となっている。
- この世界観を実現するためには、過去製品に採用された検証済みの SW を最大限再利用しながら、更に機能・性能を積み上げていくアプローチ が不可欠である。
- SW の再利用を推進するためには、SW を 結果としての「資産」ではなく、次の価値を生み出す元になる「資本」と捉え、資本回転率を高める ためにはどうしたら良いかを考えてみると良いだろう。
- オープンソースソフトウエア（=OSS）は、最初から複数のターゲット上で動作させる前提で開発 されている。そのため OSS 開発で採用されている 様々な手法や考え方を積極的に活用 していくべきである。