S5b 急変しつつあるプログラム言語/モデリングMBDの近況と変遷

-なぜ提案が最近活発になったのか？ その課題背景を現場目線で共有しよう-

アイシン・コムクルーズ㈱ suzumura-nobuyasu@aisin-comcruise.com 鈴村延保

自己紹介

広島大学/工学部電子工学科卒　アイシン精機入社後2014年　アイシン・コムクルーズへ

- **4ビットマイコンの時代から自動変速機やアクテイブ・サスペンション,ABS,ESC等のコンンピュータ、関連するハード、IC設計、実装技術の開発・製品化に従事。 その後開発比重の高くなったソフトウェア課題に取り組む。 機能安全は、その延長で取り組んで現在に至る。 現在はクリチカルシステムの効率的開発やASEAN連携が関心事。**

- **個人的には、パソコンオタク、コンパイラマニア、古代歴史マニア**

- 九州工大　産官学連携講座　非常勤講師　（'08～）
- 新エネルギー・産業技術総合開発機構NEDO　新技術調査委員　:ITベンチャー応援団（'14～）
- IPA高信頼性部会委員（' 14～）
- NEDO　生活支援ロボット実用化プロジェクト第2期　アイシンコンソ　准代表＆研究員
  （' 11～' 13）
- 自動車技術会　電子・電装部会/電子機能対応分科会　ISO26262規格審議委員　('8-'13)
- 経済産業省委託　海外自動車電子化調査WGリーダ等
  - 「平成18年度 Jaspar委託調査　Autosar周辺調査　」
  - 「平成19-24年度（財)JARI　ITS規格化事業/ITS自動車の電子化に係る欧州調査　」
- 地域コンソーシアムプロジェクト活動参加　（経済産業省助成）
  - 自動車統合制御用組込みOSの開発　' 05～
  - 機能安全対応自動車制御用プラットフォームの開発　'06～
  - 形式的仕様記述を用いた高信頼ソフト開発プロセス研究とツール開発　' 10～
  - 故障未然防衛機能を有す高信頼ソフトウェアプラットフォームの開発　' 10～
  - 高度IT 融合社会を支える次世代自動車用セキュリティ・ゲートウェイ・ECU の開発　' 14～
  - 農業機械の次世代電子制御ソフトウェア基盤の開発　' 14～

急変しつつあるプログラム言語/モデリングMBDの近況と変遷

事前案内に　追記

Matlabによるモデルベース設計MBDが車載組み込み開発で本流になっています。
モデルベース設計は機能安全の流れを織り込んで、simulink、UML、sysMLを適材適所で活用しています。 実務で応用が進んだ結果、昨年は不足点、改良点が顕在化された年であった、といえます。 その結果、GSN、D-CASE、SCN(SCDL)、safeMLといった新しいキーワードが広がっています。 従来より、フィーチャー図、DSM図、状態遷移表、マインドマップなどとの併用の必要性も言われ続けています。

一方、モデリング言語の大元であるプログラム言語も、変化の年となりました。
Apple社のSwift言語は発表以来1年で前例のない勢いでデファクト言語に駆け上がろうとしています。ロボット開発でのPython利用、組み込みでmrubyも広がっています。

重要なことは、これらの動きはインターネットの時代、深く潜行して突然浮かび上がります。
以前のように欧米の書籍が翻訳されて広がるという時間の長い動きになりません。
サイレントパラダイムシフトと呼ばれる所以です。 非英語圏では話題が遅れます。
インターネットでは、Google社のGo、Firefox財団のRust、Microsoft社のF#、MITのJulia、C++2017、ES2015、WebAssemblyなど話題は枚挙にいとまがありません。

本セッションでは、**①MBDの根底は何か？、②機能安全が定着してきて、何に困って、今何に取り組まれているのか？、③何が停滞し、何がブレークスルーしているのか？、**について、なぜ提案が最近活発になったのか？　その課題背景を現場目線で共有したいと思います。

急変しつつある
プログラム言語/モデリングMBDの
近況と変遷

事前案内に追記

事前案内に追記

現状認識と課題と背景

①MBDの根底は何か？／プログラム言語の根底は何か？

②機能安全が定着してきて、何に困って、今何に取り組まれているのか？

③何が停滞し、何がブレークスルーしているのか？

急変しつつある
プログラム言語/モデリングMBDの近況と変遷

- 現状認識と課題と背景
  - シリコン革命と影響で10年ごとに必ず起きるパラダイムシフト
- ①MBDの根底は何か？／プログラム言語の根底は何か？
  - 複雑/大規模化対応要件と、そのため開発の上流シフトが進んで変化。
  - MBD/MDDとアジャイル/リーン/摺合せ/プロトタイピング
- ②機能安全(ISO26262)が定着してきて、何に困って、今何に取り組まれているのか？
  - Part2：・ハザード分析の仕方？　・要件/設計のセミフォーマルモデリングの書き方？　・安全コンセプトSCの書き方と粒度？。
  - Part8:セーフテイケースのまとめ方/説明の仕方？
- ③何が停滞し、何がブレークスルーしているのか？
  - モデリング標準化/進化の停滞とその認識遅れ、 コンパイラ技術/CAEの進化、アイデアの積み上げ。

話に入るまえに： 大きな裏情報

-なぜ提案が最近活発になったのか？ その課題背景を現場目線で共有しよう-

- パラダイムシフトは、技術革新から来るのでは無く、せざるを得ないという理由から来る、 さらにサイレントパラダイムシフト。
- 上流シフトはヘテロジニアスでハイブリッド
- 認識遅れ：C系統言語に致命的な欠陥？ (Googleコメント)
- 認識遅れ：オブジェクト指向は、反省期からアーキテクチャ指向へ、さらに次の段階に変化。（ADL周辺を把握しないと見えない）
- モデリング標準化UML、sysMLの進化が10年間の停滞

話に入るまえに:
この提案で討議、共通認識整理したい
背景のキーワード（混乱を承知で事前に）

- システム・プログラミング／フルスタックプログラミング
- プロトタイピング、 リファインメント（段階的詳細化）
- 機能安全でわかる、ISO15504/Spiceは実はかなり下流のみのプロセス定義
- 設計には上流から下流まで**何々分析**がたくさんあります
  - 仕様書や設計書は、それだけでは検証できない。設計書は設計通りのソフトかどうか検証できるが、設計書の不足点は抽出できない、それは何々分析で抽出される。何々分析をしようとすると、製品の環境、 稼働条件、など、前提条件が必要となる。
- MBSE=ヘテロニジアス_プログラミング＋ハイブリッド_コントロールシステム
- オブジェクト指向の欠点／アーキテクチャ指向／Dependency-Injection
- RE／要求工学、SE／システム工学、アーキテクチャ工学、ソフトウェア工学
- IDE ：Interactive Development Environment, Executable Specification
- リーンプログラミング、摺合せ、アジャイル、プロト、REPL、Playgrounds
- 関数言語のエッセンス
- プログラムのコントロールフローとデータフロー側面 、階層、プロダクトライン

進め方:（10分提起＋5分討議）＊5

- **イントロ：　現状認識と課題と背景**
  - シリコン革命とその影響で10年ごとに必ず起きるパラダイムシフト
- **①MBDの根底は何か？／プログラム言語の根底は何か？**
  - 複雑/大規模化対応要件と、そのため開発の上流シフトが進んで変化。
  - MBD/MDDとアジャイル/リーン/摺合せ/プロトタイピング
- **②機能安全が定着してきて、何に困って、今何に取り組まれているのか？**
  - Part2：　・ハザード分析の仕方？　・要件/設計のセミフォーマルモデリングの書き方？　・安全コンセプトSCの書き方と粒度？。
  - Part8:セーフテイケースのまとめ方/説明の仕方？
- **③何が停滞し、何がブレークスルーしているのか？**
  - モデリング標準化/進化の停滞、認識遅れ、 コンパイラ技術/CAEの進化、アイデアの積み上げ。
- **④じゃあ明日からどうする？**

イントロ：　現状認識と課題と背景

シリコン革命とその影響で
10年ごとに
必ず起きるパラダイムシフト

こんな気持ちを持っています

-現状に対して、そう思いませんか？-

2014-8-29

第16回 組込みシステム技術に関する

サマーワークショップSWEST16

- C言語だけでの開発の現状を打破しよう！
- “プロトタイピング－詳細設計－C言語実装の反復設計” が開発の本来あるべき姿。

こんな気持ちを持っています

-現状に対して、そう思いませんか？-

2015-3-10

ESD21/JASA共催

「TPS/Agile組込システム」セミナー

TPS：TOYOTA Product System(**かんばん**)

- 機能安全が示す、摺り合せ反復設計！
- 日本式摺合せ反復設計が国際標準化！

2015-7-23　軽量Ruby普及・実用化促進
ネットワーク設立記念講演会

車載組込みから見たRuby/mruby

アイシン・コムクルーズ㈱　suzumura-nobuyasu@aisin-comcruise.com　鈴村延保

中国経済産業局：
平成26年度 IOT時代を担うM2M領域への
“Ruby(mruby)”の新規参入可能性調査委員会参加

車載組み込み開発から見たRuby/mrubyとC言語開発、Matlab/Simulink開発の比較

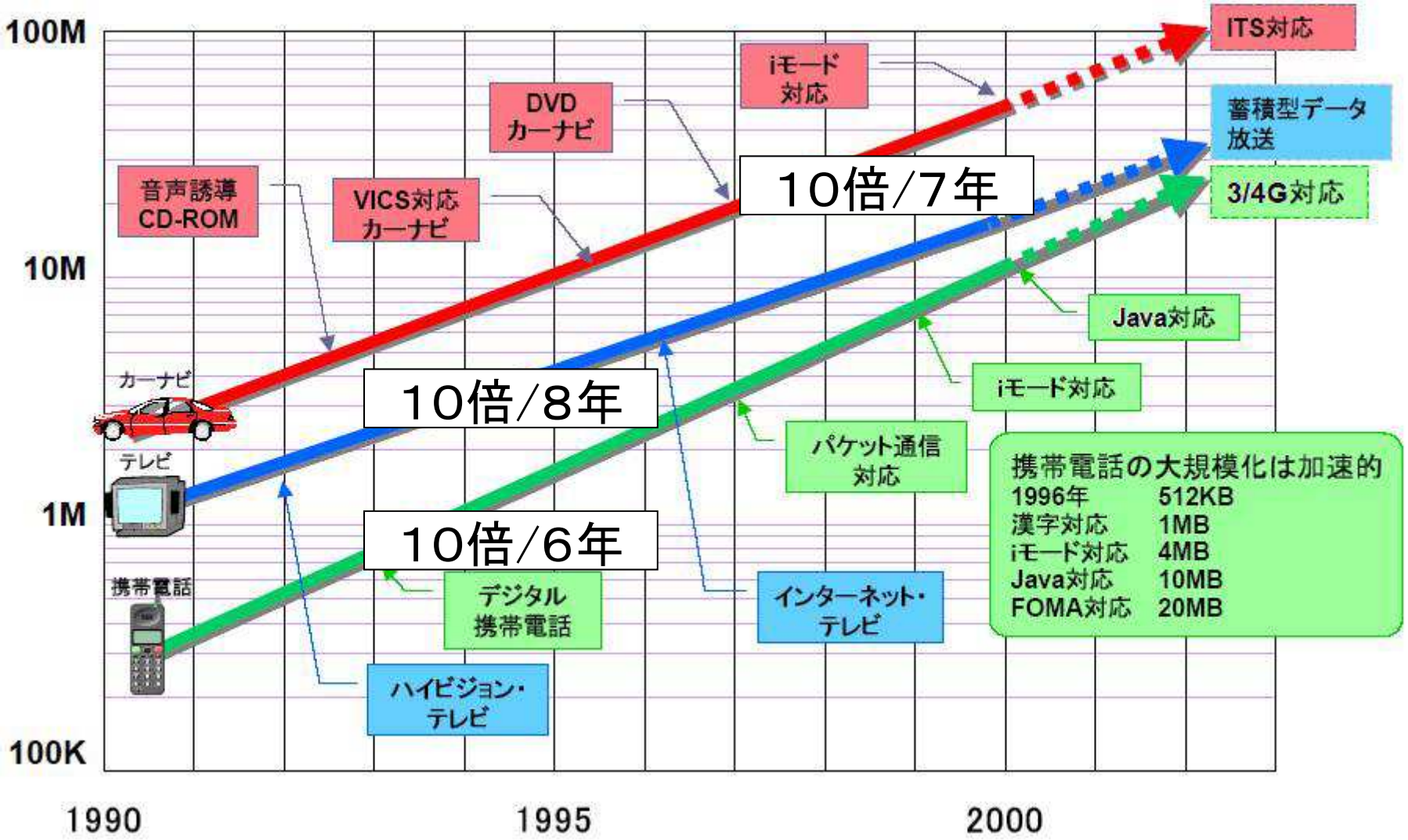
－プロトタイピング言語としてのRubyの可能性－
-知られざる事実：車載開発ではポピュラーなMatlab／M言語そっくりなRuby-

現状認識と課題と背景

開発がどんどん“非会話的”になっている、という認識

- 扱うプログラム、データがどんどん大きくなっている。
- 動かしてみることが、どんどん出来にくくなっている。
- ソフトウェアだけが在っても検証できない。

大規模化する組み込みソフトウェア



資料：日経エレクトロニクス2000 9-11(no.778)をベースに追加、修正。携帯電話の増加率は変更済み

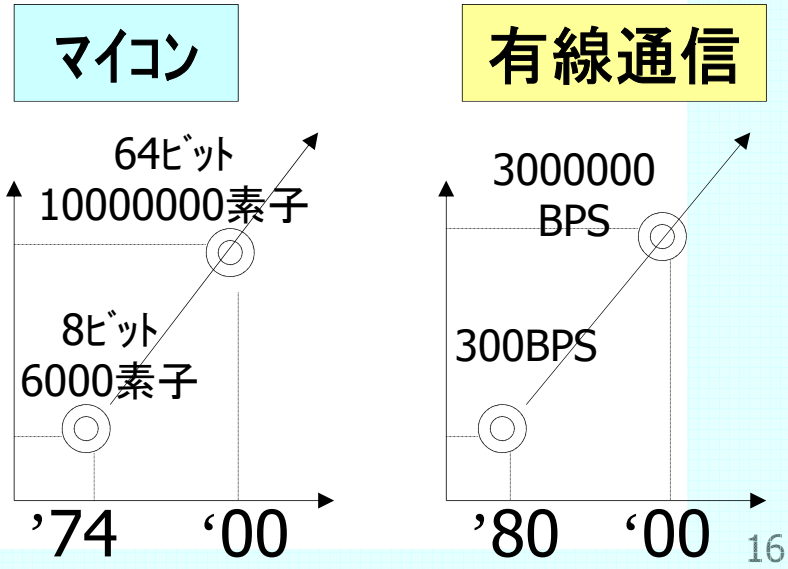
半導体革命(シリコン革命)が車へ

- ICの機能は、継続的に
  高機能、低コスト化していく。
  ＝ムーアの法則 （18ヶ月で集積度2倍を実現）

- マイコン（1971～）

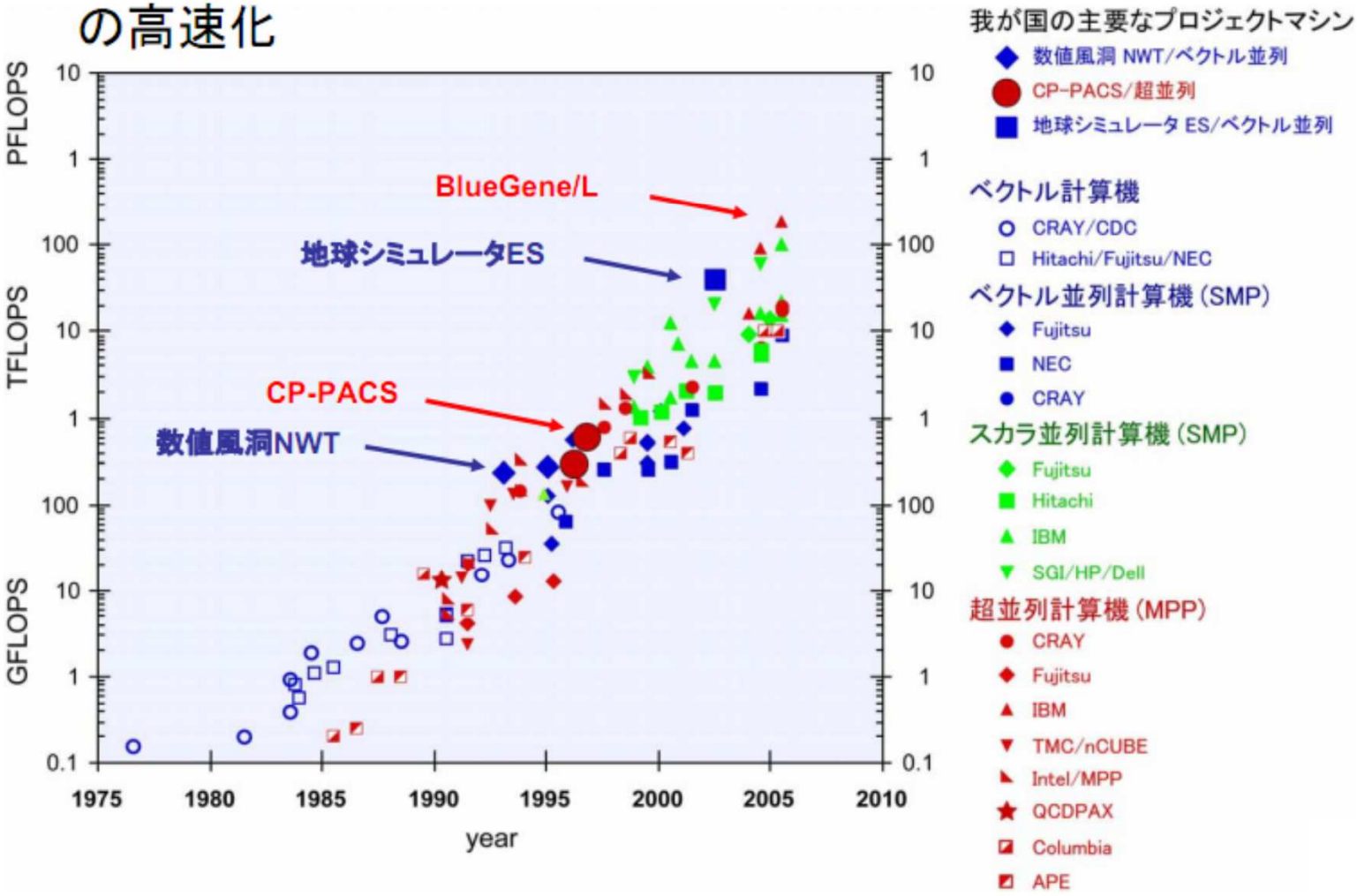
New

- 画像センサ「カメラ」（‘00～）
- 無線通信素子 （‘02～）
- マイクロマシン



スーパーコンピュータの発展

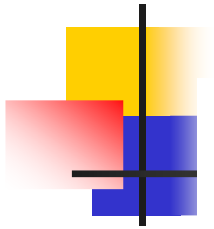
30年間（1976年～2005年）で100万倍の高速化



次世代スーパーコンピュータ開発 プロジェクトへの期待

http://www.rccp.tsukuba.ac.jp/pflops/050926-iwasaki-pflops-symposium.pdf

シリコン革命の、すさまじい速度

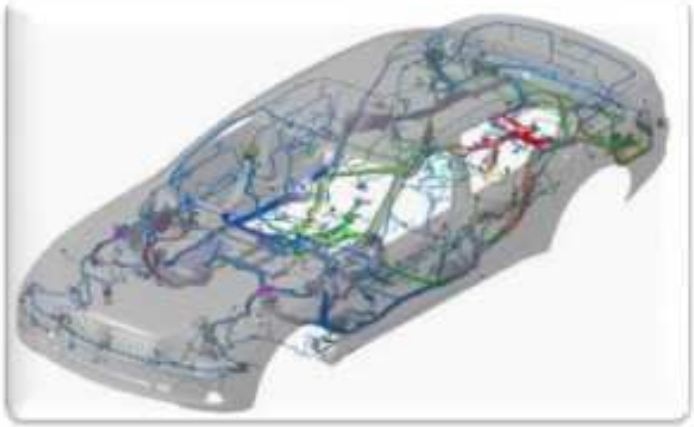


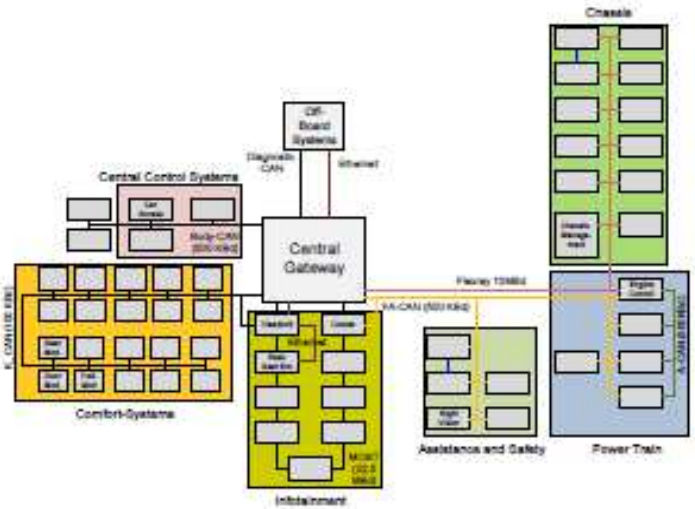
- 過去30年のシリコン革命で起きたこと（事実）
  - マイコンの処理速度は**30年で100万倍化**
    - 8080　8bit　3μSEC　が　64bit　8ｺｱ2ｽﾚｯﾄﾞ　0.3nS
  - マイコンの メモリーは **30年で100万倍化**
    - ｺﾓﾄﾞｰﾙ　PET　4KByte　が　WINDOWS7　4GByte
  - **ネットワーク 速度は　30年で100万倍化**
    - 110-300BPS音響ｶﾌﾟﾗ　が　200MBPS光へ

画像センサ「カメラ」も シリコン革命が始まった

- CCD構造からC-MOS型へ変化することで
  当初10万画素程度(’80年代）が1億画素(2010)へ（1000倍/15年）

AUTOMOTIVE E/E-ARCHITECTURE TODAY





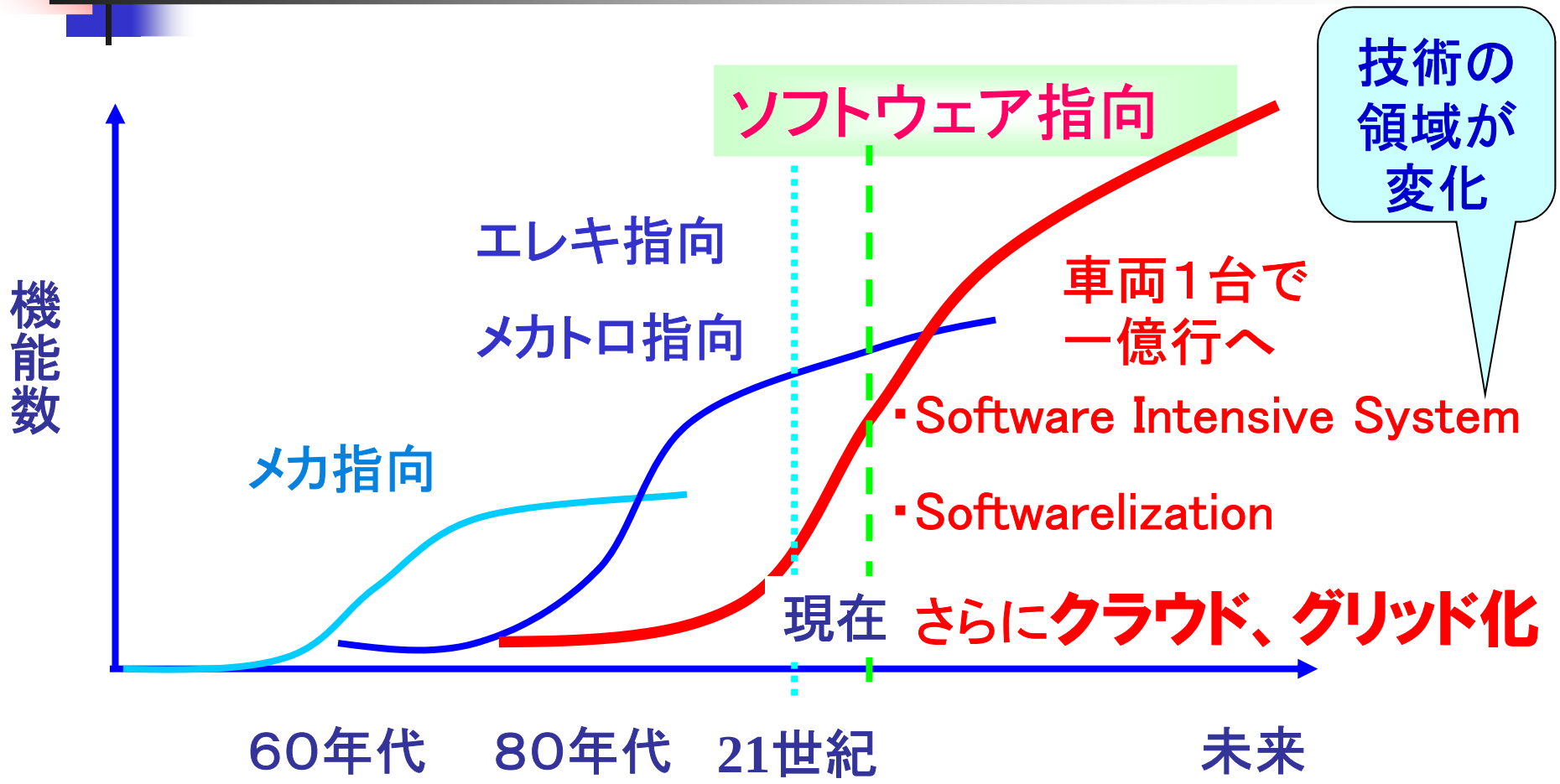
7 series wiring harness	
Length	2751 m
Connecting Plugs	520
Weight	43 kg

Electronic Control Units (ECU)	
No. ECUs	28 … 74
CPUs	ca. 230
GPUs	> 5
Power PCs	3
Busse	CAN, LIN, MOST*, Flex Ray, Ethernet

Other	
Software	3-5 GB Application 15-20 GB Data

http://www.across-project.eu/workshop2013/20130121_BMW_HIPEAC-Conference.pdf

現代の動向：車両,ECU開発がソフトウェア指向へ移行した



動向:“ソフト指向へ変化”の将来はどうなる？

初期の銀行オンライン
システム規模を超えた

・車両全体のソフトウェア量は１０００万行から

１億行へ。（Eu/ARAMISプロジェクトのでは車全体で3-5ギガバイトと表現している）

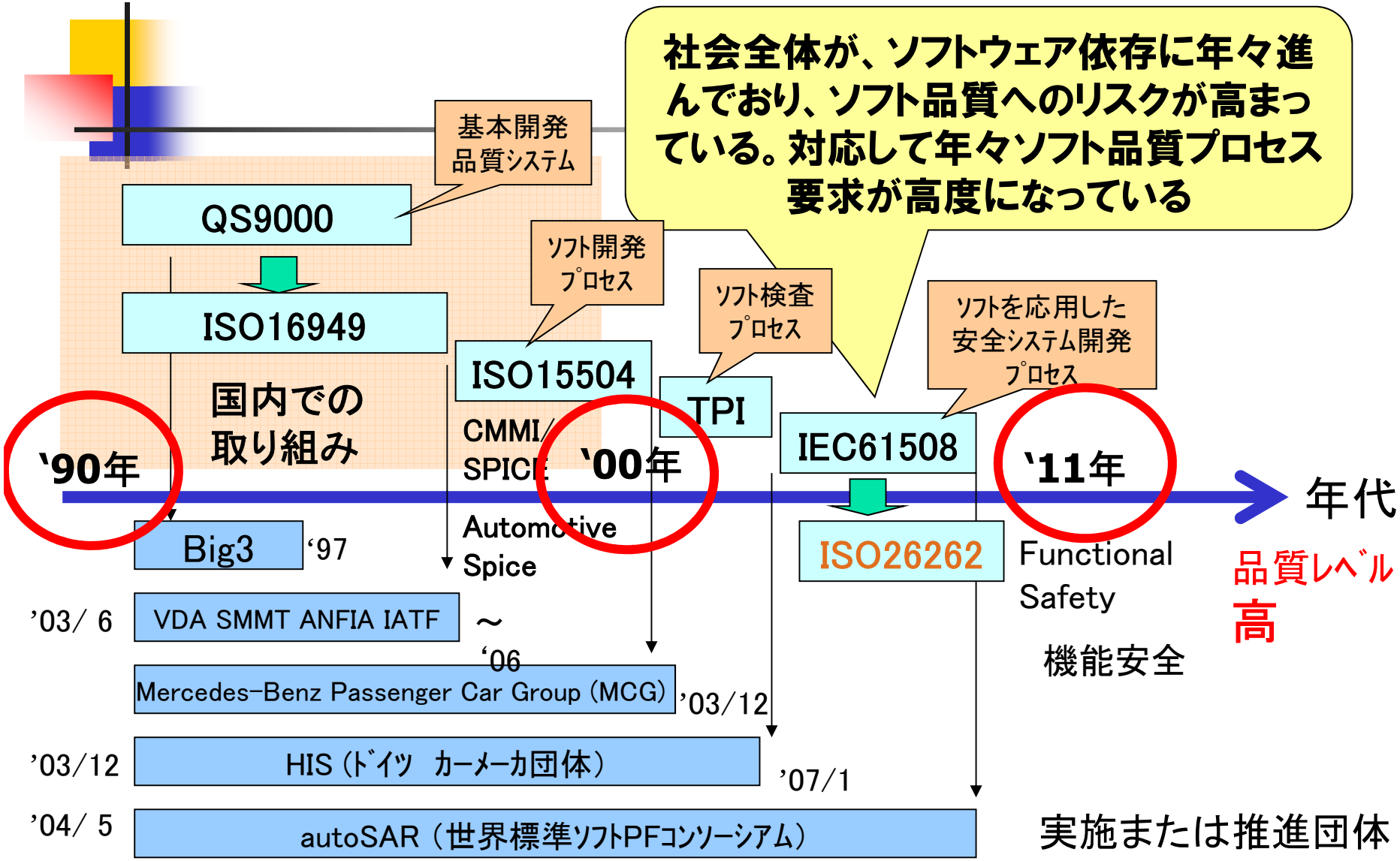
技術の
領域も
変化

効率的で高信頼な開発手法変革が求められている

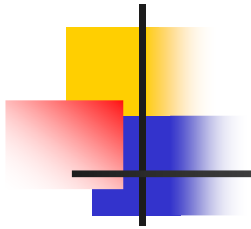
現状認識と課題と背景

シリコン革命とその影響で
10年ごとに
必ず起きるパラダイムシフト

車載組込動向：品質要求基準の変化



パラダイムシフト



- 10年毎のパラダイムシフトが起きている。
- 今後も継続的に起きていく、と想定される。
- 効率的で、高信頼な開発手法変革が求められている。
- パラダイムシフトは、インターネットに潜伏して急拡大するため、見えにくくなっている（サイレントパラダイムシフト）

パラダイムシフト
車載組込動向

- 機能安全
- モデルベース開発（MBD)
- システムズ・エンジニアリング（MBSE)

①MBDの根底は何か？／プログラム言語の根底は何か？

- 複雑/大規模化対応要件と、
  そのため開発の上流シフトが進んで変化。

- MBD/MDDとアジャイル/リーン/摺合せ/プロトタイピング

プロトタイピング－リファインメント
－C言語実装とは？

	開発の始まり	開発途上	最終段階
開発段階	初期試作 構想段階 仕様策定	中期試作 完成度向上	量産試作段階 N増し評価、
必要とされる 開発方策	**プロトタイピング**	**リファインメント** 最終言語への変換	**C言語実装** 評価、最終改良、 適合完了
便利で、キーとなる 技術	早い開発、早い フィードバック 動く、会話型開発、 抽象化、ヴィジュア ライズ、	プロトタイピングで 確認された事項を 誤り無く、注意深く 詳細化していくこと	高品質、高信頼、
C言語とその環境ですべて、まかなえているのか？			

フルスタックという言葉が認識されている フルスタックプログラミング言語とは？

- システム実装構造は階層化されている

階層とも**スタック**とも呼ばれている

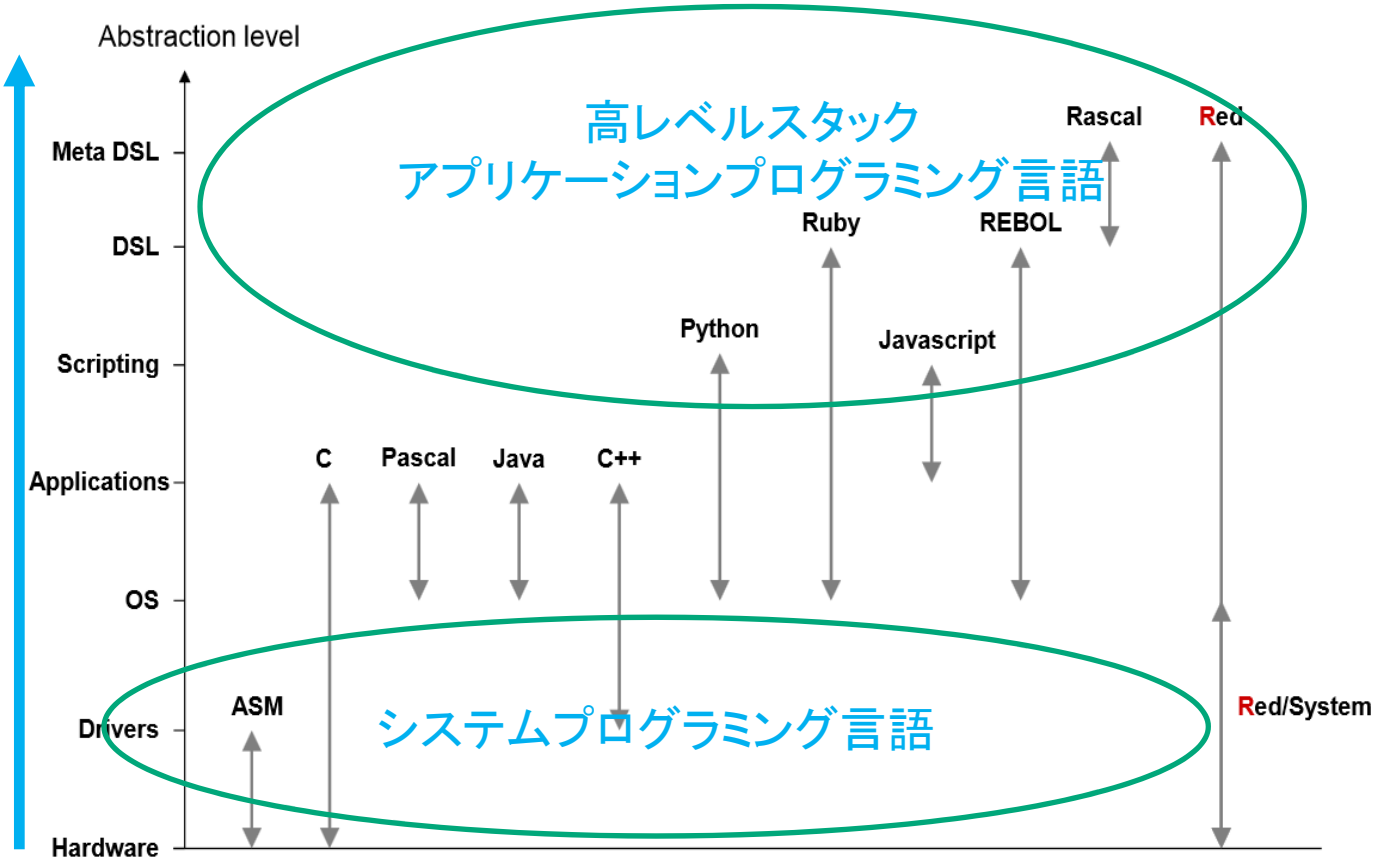
階層	詳細
ｱﾌﾟﾘｹｰｼｮﾝ	統合システム
	ｼｽﾃﾑ、エレメント
	アプリケーション用コンポーネント
PF	PF-API
	ミドルウェア
HAL OS,Driver	HAL OS,ドライバー

フルスタックプログラミング言語とは？

(すべてのソフトウェア階層をカバーできる言語がフルスタックプログラミング言語といわれるが、それはひとつの言語では難しい。)

階層	詳細
アプリケーション	統合システム
	システム、ELEMENT
	APP-COMPONENT

階層とも
スタック
とも呼ばれている



http://static.red-lang.org/Recode2013-Red.pdf

課題：

・ソフト開発の中心ツールであるC言語は危険な言語、

.40年前の真空管時代の言語。

しかし、代替言語が無い状況。

- **C,C++,Java言語には根本的欠陥が残っている（Google視点）**
- 新たな流れ　Swift,　Ruby/Python…

  MBD(モデルベース設計)の発展。

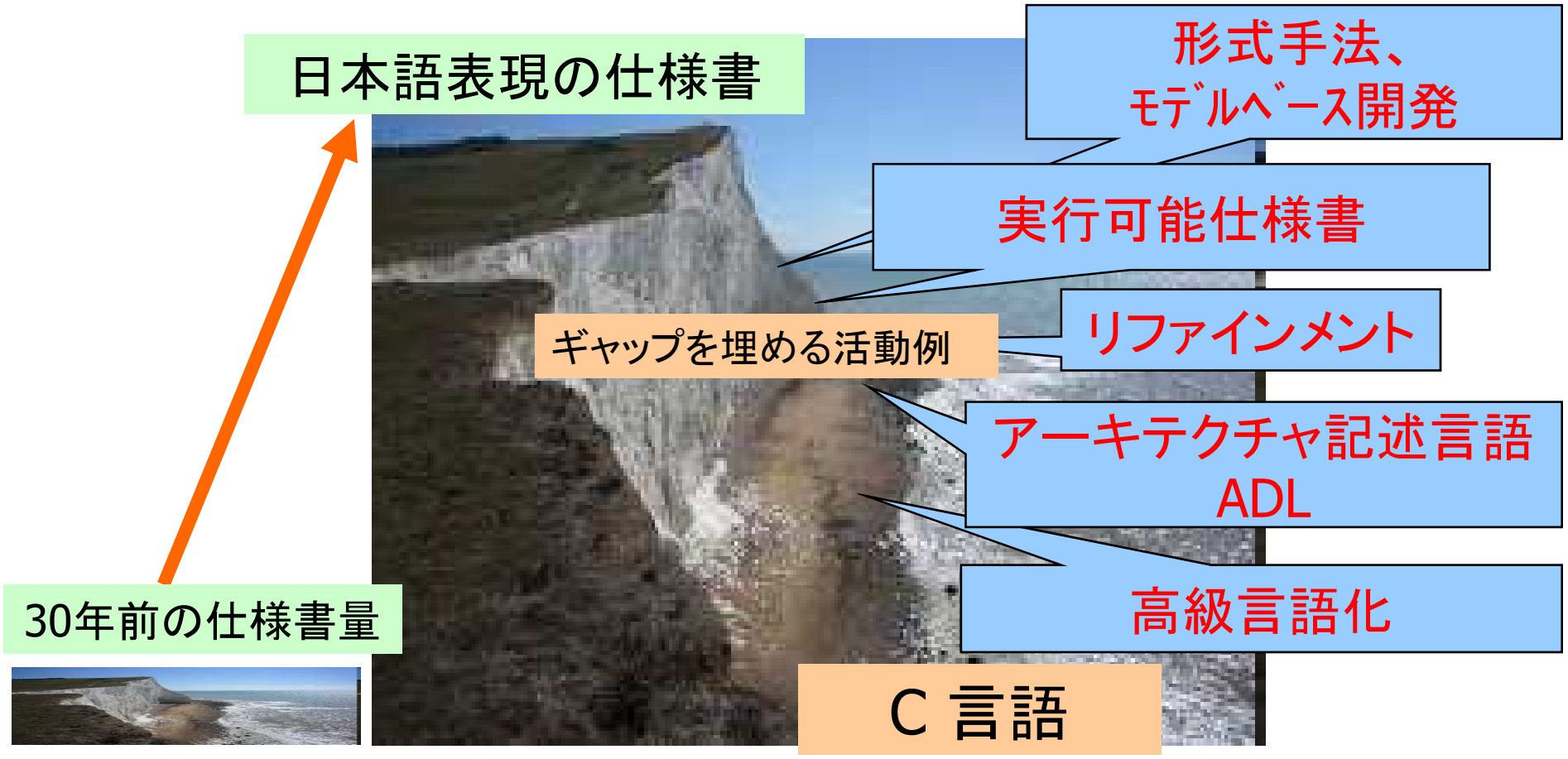
-課題例: 仕様書とプログラミング言語の大きな表現力ギャップが課題になっている-

C言語の長所と欠点。(欠点が多い)

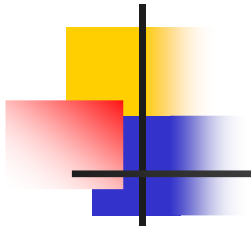
- コントロールフロー表現は得意
- データフロー表現は苦手
- 部品化が苦手
- 階層化が苦手
- アーキテクチャ表現が苦手
- 会話型開発、プロトタイピングが苦手
- 抽象化記述が苦手。
- データの状態属性が苦手(nil概念(Liveness)、mutable、参照透過性)。

C言語は40年前　真空管時代に提案された言語

-課題例: 仕様書とプログラミング言語の大きな表現力ギャップが課題になっている-

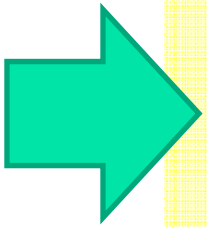


C言語ベースの開発だと



日本語表現の

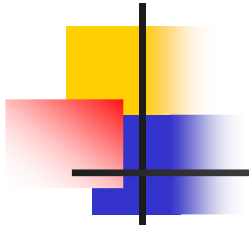
仕様書



C 言語

```
#include "types.h"
#include "core.h"
#include "reader.h"
#include "printer.h"

void throw(MalVal *obj) {
    mal_error = obj;
}
MalVal *get(MalVal *obj, MalVal *key) {
    MalVal *val;
    switch (obj->type) {
    case MAL_VECTOR:
        return _nth(obj, key->val.intnum);
    case MAL_HASH_MAP:
        if (g_hash_table_lookup_extended(obj->val.hash_table,
                                         key->val.string,
                                         NULL, (gpointer*)&val)) {
            return val;
        } else {
            return &mal_nil;
        }
    case MAL_NIL:
        return &mal_nil;
    default:
        abort("get called on unsupported type %d", obj->type);
    }
}
```



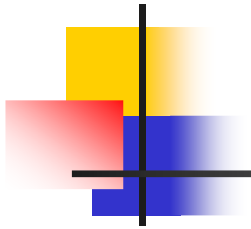
プロトタイピングが重要

Matlab／Simulinkはプロトタイピングがとても得意です。

Rubyもプロトタイピングがとても得意です。

C言語はプロトタイピングが苦手です。

C言語は部品化も苦手？



じゃあプログラミング言語Cの**関数**は、何の目的で存在するの？

```
while ((c = getchar(  )) != EOF)
    putchar(c);
```

C言語には驚くべき事実が隠されている！！

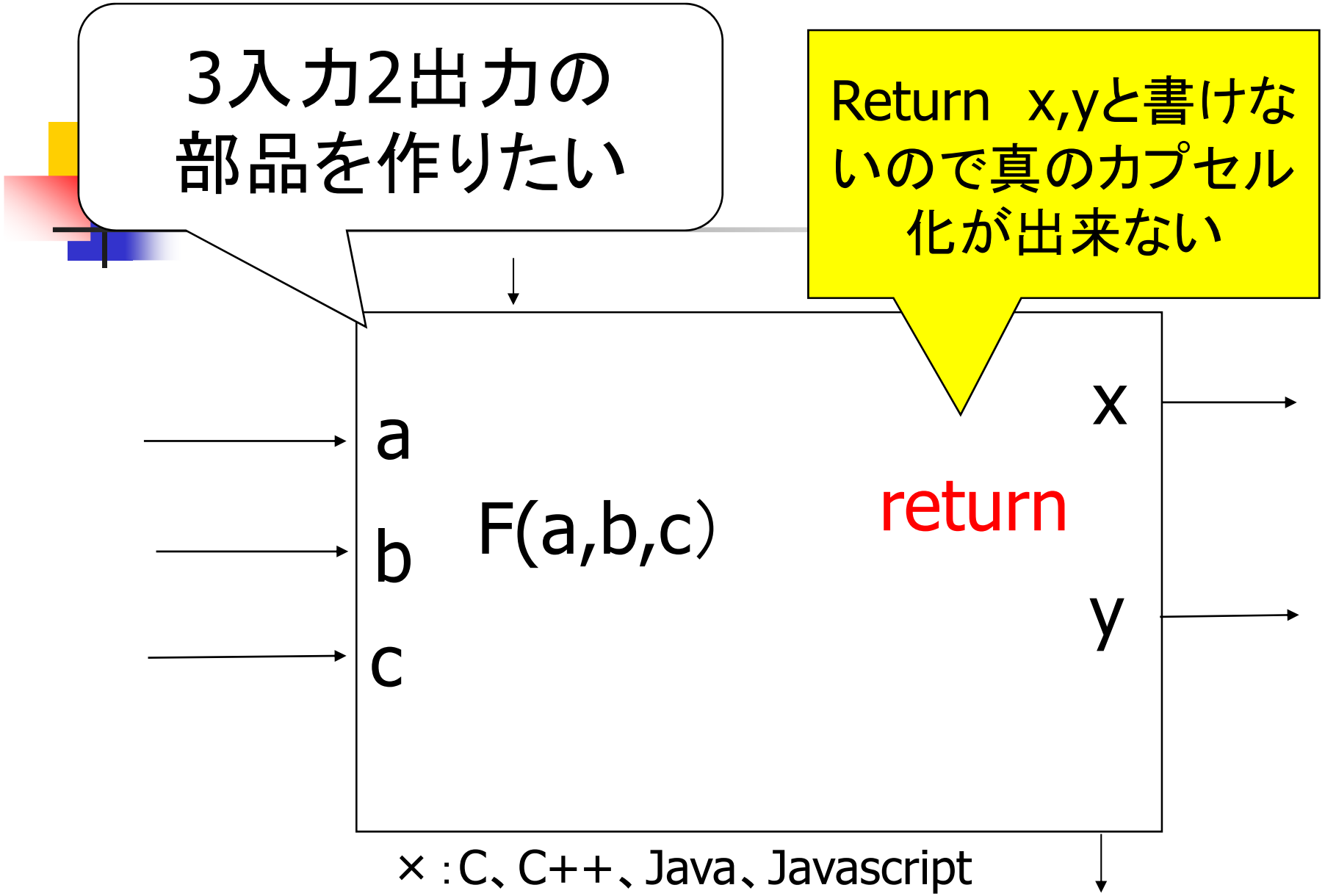
関数は、何の目的で存在するのか？

- **関数は**、結局if文やwhile文など**制御フローの表現を簡潔、高度にするため**の目的が中心。
  （C言語自体　40年前Control Flowが未だ中心の時代、または、　そのような用途領域で生み出され、出来上がった言語）

- **関数は**ﾗｲﾌﾞﾗﾘ構築に使用されるが、制御ﾌﾛｰ記述を効率化するためのﾗｲﾌﾞﾗﾘｲｰ構築のねらいであり、**本来の部品化ﾗｲﾌﾞﾗﾘｰには無理がある！！**

- Control Flow中心の応用時代から、Data　Flowの重要性比重が高まっているのが時代の流れ。

- 例題：部品化プログラムを関数で書いてください
  ３個のデータを使用し、２個の結果データを作成するソフトウェア部品
  (C言語では依存性なく記述することが難しい）



×：C、C++、Java、Javascript

○：Matlab , Swift, Ruby, Go, F#, Haskell

じゃあ、C言語開発の現状を補完するには、明日からまずどうする？

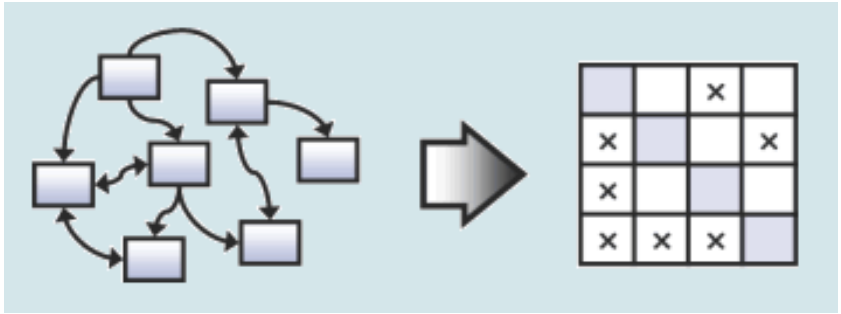
- C言語は機能の低い、危険な言語です。（機能安全の認識）
- スタイルガイドとガイドチエッカーを利用しましょう （機能安全の認識）
- 静的解析ツールを利用しましょう(型整合を含む)
- **設計にはモデリングを併用しましょう。（機能安全の認識）**
- 従来のモデリング(例UML)を併用しても、見える化は、十分ではありません。(UMLは2.0以上またはsysMLを利用しましょう、さらに要件周辺でGSN,D-Case.)
- ツールで補完しましょう
  - 上記　style_guide,　static_analysisを含む
  - 部品化をグローバル変数で支援するツールがあります(Matlab/Simulink,Autosar周辺)
  - アーキテクチャADLを意識した見える化ツールを利用しましょう。
- 広義の階層化、依存関係の見える補完手段など、UMLでは不十分で、さらに改良(イノベーション)が重要です。
  - DSM手法は新しいモデリング提案ともいえます。利用しましょう(matrix、表表現は　欧米ではモデリングと言いませんが)。
  - プロダクトライン表現ではフィーチャー図が必要です。(未だUML標準化されていませんが、欧州提案EAST－ADLでは標準提案されています)、

DSM手法とは

- DSMとは、Dependency Structure Matrixの略であり、繰り返しやフィードバックが多い開発プロセスをわかりやすく表現するために開発された、プロセス改善のための分析技法です。DSMは、1968年にDonald Steward氏によって開発され、以後、マサチューセッツ工科大学、ハーバード大学、イリノイ大学などの数多くの大学で研究が続けられ、ここ10年の 間、Boeing、Lockheed Martin、Intelなどの企業において、多様な分野で幅広く用いられてきました。

  DSMを利用することで、これまでは分析が困難だったシステムの依存関係をより直感的に可視化・分析することが可能になります。

- Lattixは、DSMをソフトウェアアーキテクチャの分析に適用した初めてのツールです。ソフトウェアアーキテクチャの構造分析を行い、サブシステム間の依存関係を簡潔な表形式(マトリクス)で表現します。



System		1	2	3	4	5
Module A	1	.		X		
Module B	2	X	.	X	X	X
Module C	3	X		.		X
Module D	4	X			.	X
Module E	5					.

https://www.techmatrix.co.jp/quality/lattix/dsm.html

じゃあ、C言語開発の現状を補完するには、明日からまずどうする？
-設計にはモデリングを併用しましょう-
でも注意点はまだまだあります

- 状態遷移図で完全であっても、状態遷移表で確認すると、ユーザ目線では漏れがあります。　状態遷移表で確認をしましょう。
- 状態遷移表で完全であっても、安全分析をすると、システム目線では漏れがあります。　多種の安全分析をしましょう。
- 派生開発、車種展開ではプロダクトライン開発を意識し、フィーチャー図で確認をしましょう。
- モデリング**sysMLは**上流要件を記載する場ですが、結果を記載する場であって、上記を推進する道具ではありません。**良いテンプレートが必要です**。さらに要件や設計が妥当である説明が求められてきて、わかりやすく、**説明のためのモデル表記を併用**する方向になってきています。
- これらはC言語開発の課題ではありませんが、カスタマー目線の基本課題です。

MBDの根底は何か？／プログラム言語の根底は何か？ Appleの新提案Swift言語はどうなっているの？ そのほかの言語の状況は？(そのキーワードは？)

- 複数のプログラム言語を比較すると、その長所短所が肌でわかります。
  - Ruby,Rust(Firefox財団)を意識している、その本質は？
  - **REPLとPlaygrounds**
  - **RuPyデイレクション**がサイレントパラダイムシフトの方向(後述)

Swift言語

Designed by Chris Lattner

（LLVMの中心的開発者、ということは、すごい人）

From his Blog

- the experiences from **Rust, Haskell, Ruby, Python**, C#, CLU

- **Playgrounds** feature and **REPL** were a personal passion of mine.

http://www.nondot.org/sabre/

Swift言語
Designed by Chris Lattner

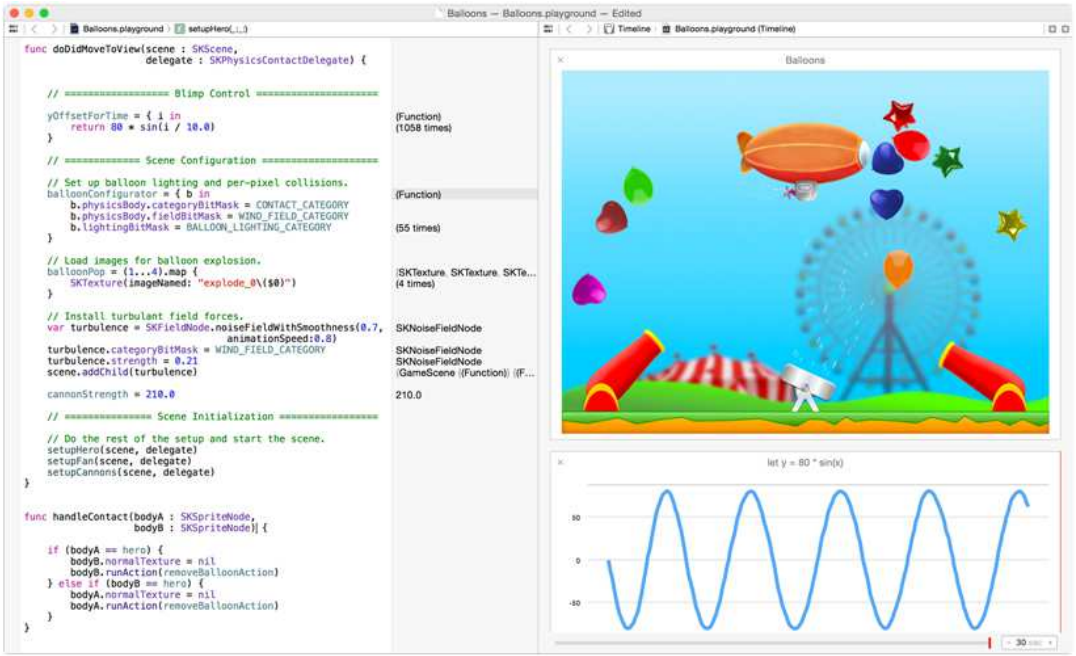
1年前に聞いた方は、今印象がどう変わったか、感じてみてください

・会話型
・プロトタイピング

REPL（Read Eval Print Loop）
REPLとは、コマンドラインからコマンド（プログラミングの命令）を入力して、その場でインタープリターが解釈して実行してくれるもの。
ちょっとしたプログラムを試したい時に重宝する。(LISP由来の環境）

Playgrounds （道場）
会話的に動作を確認する環境。

Control-Flow Debug と
Data-Flow Debug 両方が便利



https://developer.apple.com/swift/

REPL and Playgrounds はプロトタイピングのための環境

	C	C++ /14	Java	Go	Ru st	LISP	Ruby/Python	SCALA	Swift
REPL	×	×	×	×	×	○	○	○	○
Playgrou nds	×	×	×	×	×	○ Plottin g	○SciRuby、 SciPy	○ scalaplot	○

Google 提案

Mozilla 提案

Apple 提案

REPL and Playgrounds はプロトタイピングのための環境

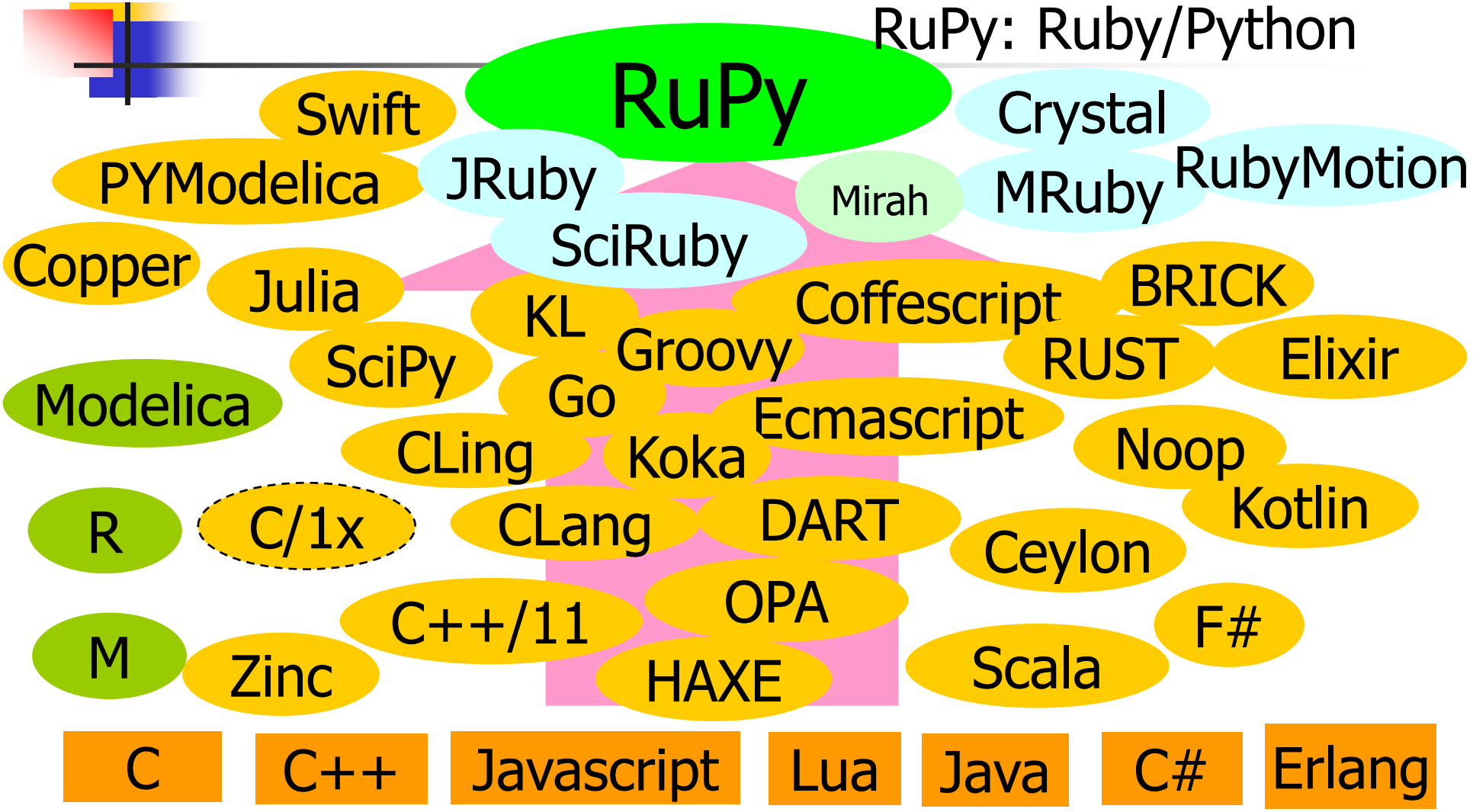
Matlabは実は昔から
・会話型
・プロトタイピング指向だった

	C	C++	Ruby/Python	Swift	Matlab	UML
REPL	×	× ○Cling CERN	○	○	○	×
Playgrounds	×	× ○Draw CERN	○SciRuby、SciPy	○	○	×

Mathworks 提案

総ての言語がRuPy方向へ　　静かなるパラダイムシフト

(自然、簡潔、高級(oo+fo)、会話的=読みやすい=コミュニテイに好まれる⇒コミュニテイ指向のパラダイムシフト方向)



考える背景

時代の流れ

JARI
海外電子化
動向調査より

100年単位の時代変化

エジソン
フォード

ATT　ベル研究所
IBM　ワトソン研
XEROX　パロアルト研

産官学連携
クラスタ
コンソーシアム

天才
一人が
時代を変えた

天才が集まった
会社（研究所）が
時代を変えた

行き詰り

コミュニテイしか、
時代を変えられない

激変

- ﾃﾞｼﾞﾀﾙ技術・ﾈｯﾄﾜｰｸ技術
- ｸﾞﾛｰﾊﾞﾙ企業間取引コストの激減
- オープン国際分業化
- ビジネス・モデル

ヨーロッパにおけるイノベーション政策の発足と経緯

JARI
海外電子化
動向調査より

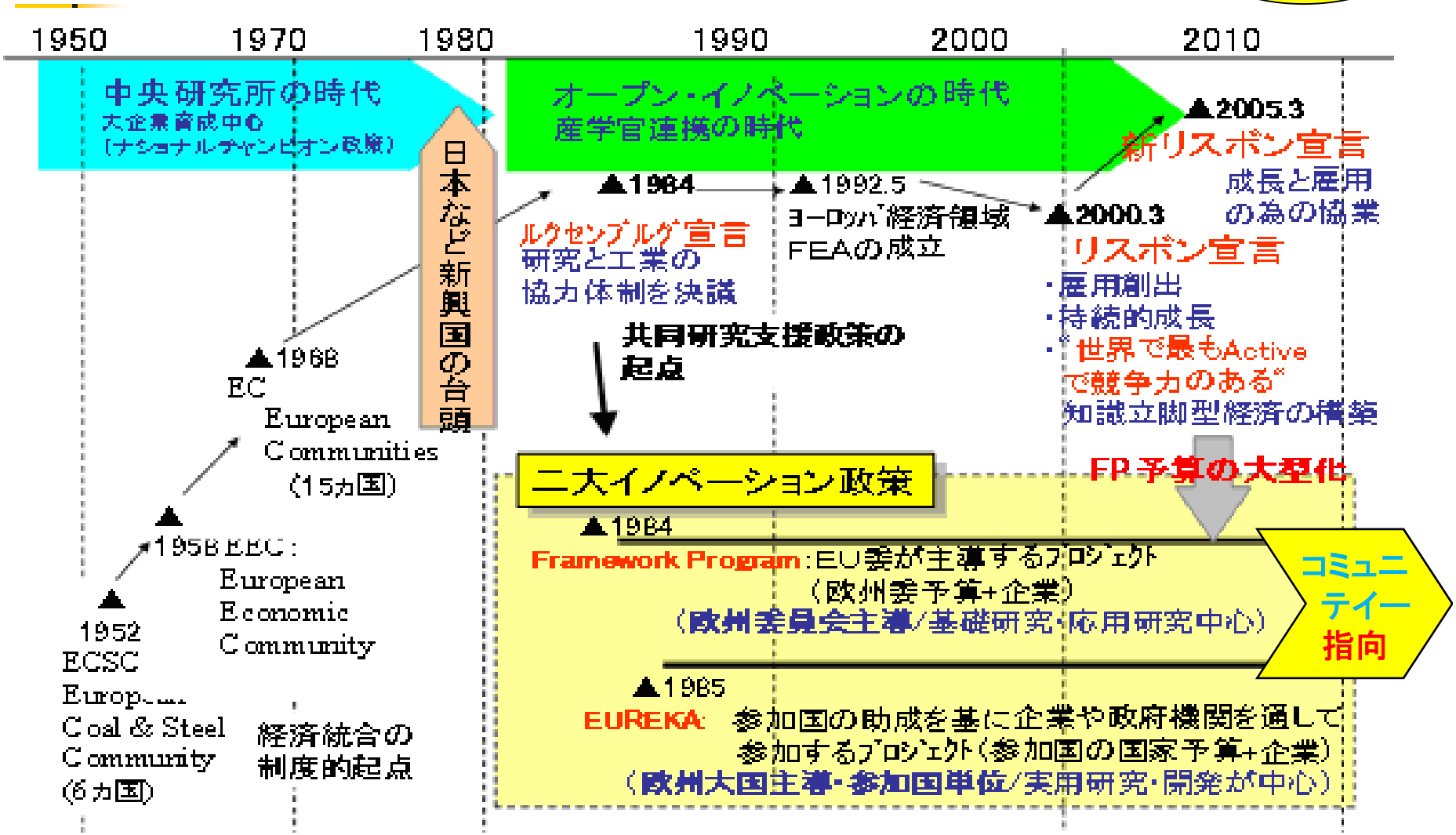


図　ヨーロッパにおけるイノベーション政策の発足と経緯

東京大学　小川紘一先生

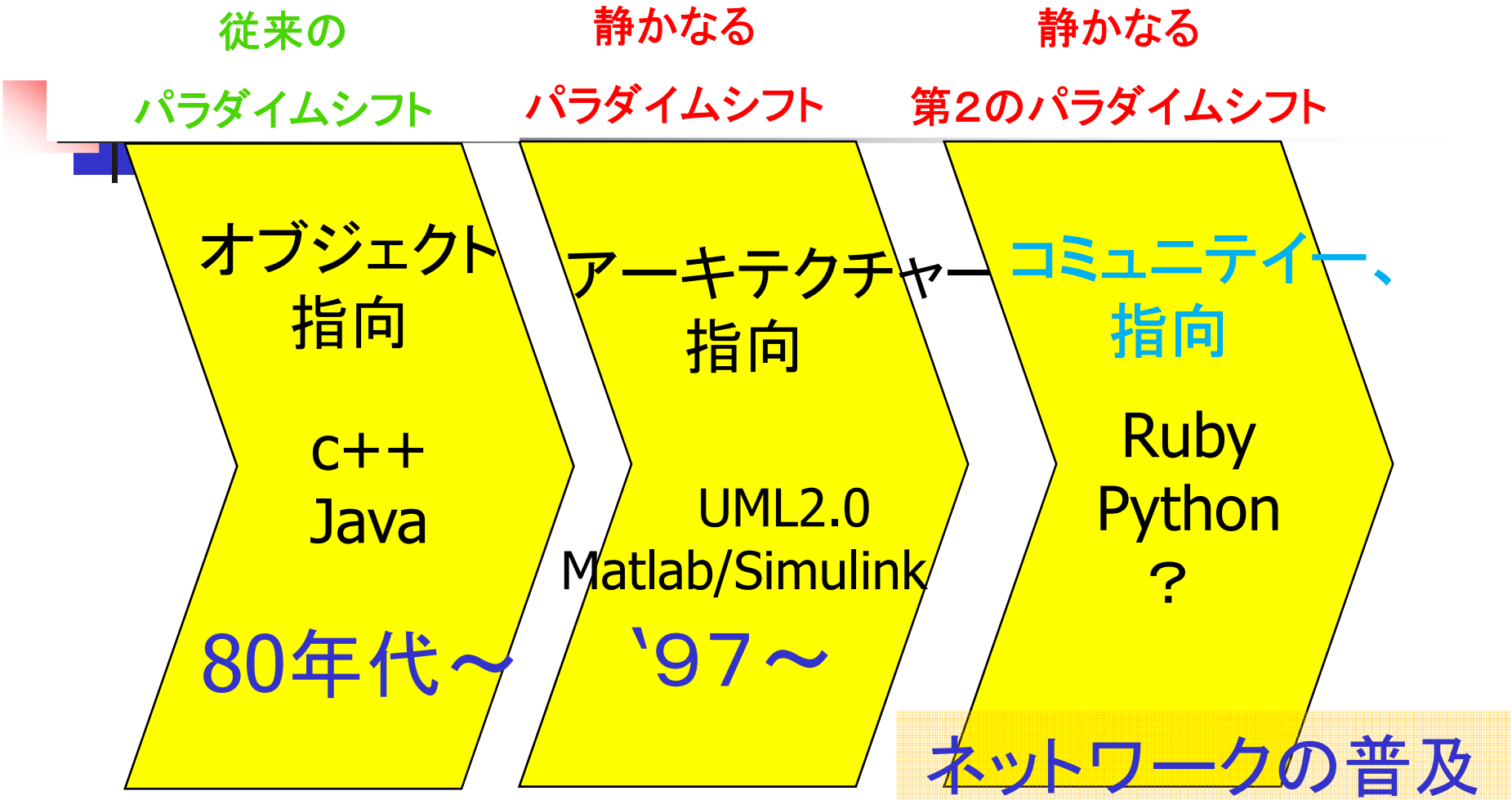
私の時代認識

2012/6時点

静かなるパラダイムシフト

2012/6時点

ソフトウェア開発で例えると



ネットワークの普及と共に、もう始まっている

RuPy Direction RuPy Conference since 2007

people twitting as follows in the Internet

- SCALA is Ruby like Java (More Functional)
- RUST(Mozila) is Ruby like Java,C
- Groovy is Ruby like Javascript
- Elixir is Ruby like Erlang
- Coffeescript is Ruby like Javascript, into Ecmascript
- Julia is Ruby like Matlab, R
- C++/11 is RuPy like C++ (More By Boost)
- JVM7.0 finally Invoke Dynamic

  - Easy to Read, Easy to Learn, Easy to think
  - Less Noise、Less Celemony, Less Magic Spelling
  - Comfortable, like, Love ,so Community love RuPy.
  - Computer Scientists respect Ruby
  - Small Talk &LISP communities respect Ruby

2000

2000年よりオブジェクト指向は反省期へ

資料事例1

Results

Serious UML 1.x weaknesses detected!*

- missing concept hierarchical decomposition
  - divide and conquer (top down)
  - building blocks (bottom up)
  - bidirectional interface
- missing concept for logical components
- missing concept for mapping to physical components
- missing concept for encapsulation
  - black-box approach for just the right level of detail
- …

UMLにおけるADL概念の弱さを指摘

*M. Broy, M. von der Beeck, P. Braun and M. Rappl: *A fundamental critique of the UML for the specification of embedded systems, unpublished, 2000*

Ames Laboratory
US Department of Energy

Fachhochschule Bielefeld
University of Applied Sciences

参考 : ADL Workshop2001 http://www.forsoft.de/zen/publications/2001/AdlWs2001.pdf

ヘテロジニアスとハイブリッド

- **UML**はESLレベルのツール。エレクトロニクス技術者には良いが、メカを含むヘテロジニアスな領域の取り扱いは難しい。(ESL: Electronic System Level)
- **UML**はハイブリッド制御システム記述も苦手。
- **Matlab／Simulink**はヘテロジニアス、ハイブリッドな領域の取り扱いが便利

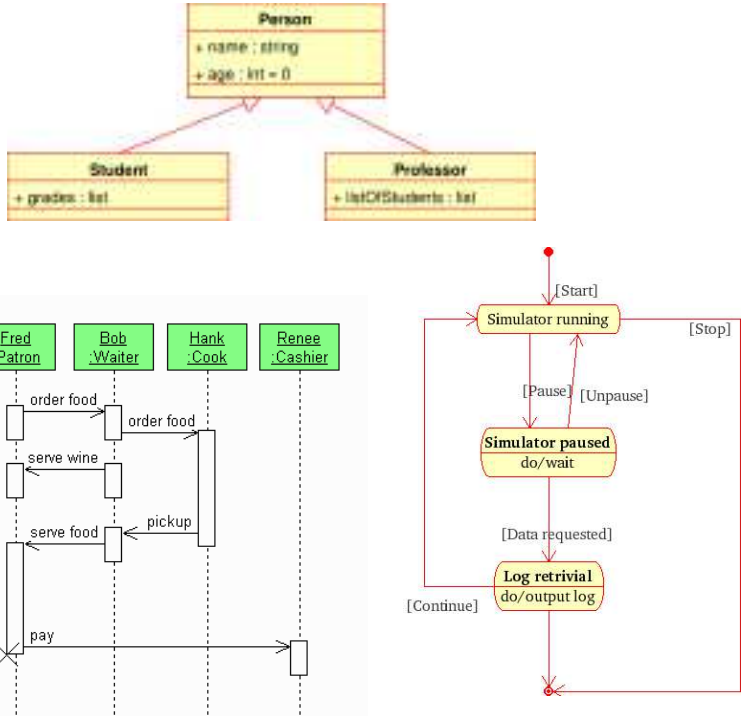
  ＝　メカ設計者まで含んだ　コミュニケーション可能。
  - 実行可能な仕様書が実現できるといわれる。
- **UML**もヘロジニアス／ハイブリッドな領域へ進化中（UML2.0やsysML）

＊Matlab/Simulinkは、IT分野でも最近注目されている関数ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語の実応用例（自動車が概念先行例）

UML と Matlab/Simulink

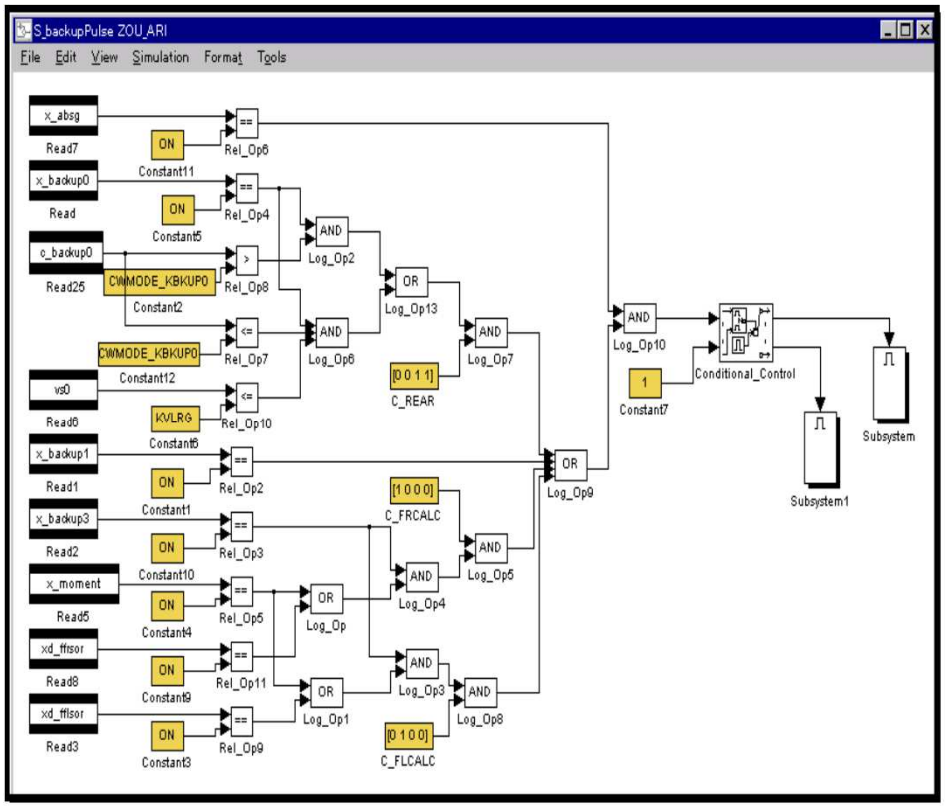
UML

Unified **M**odeling **L**anguage

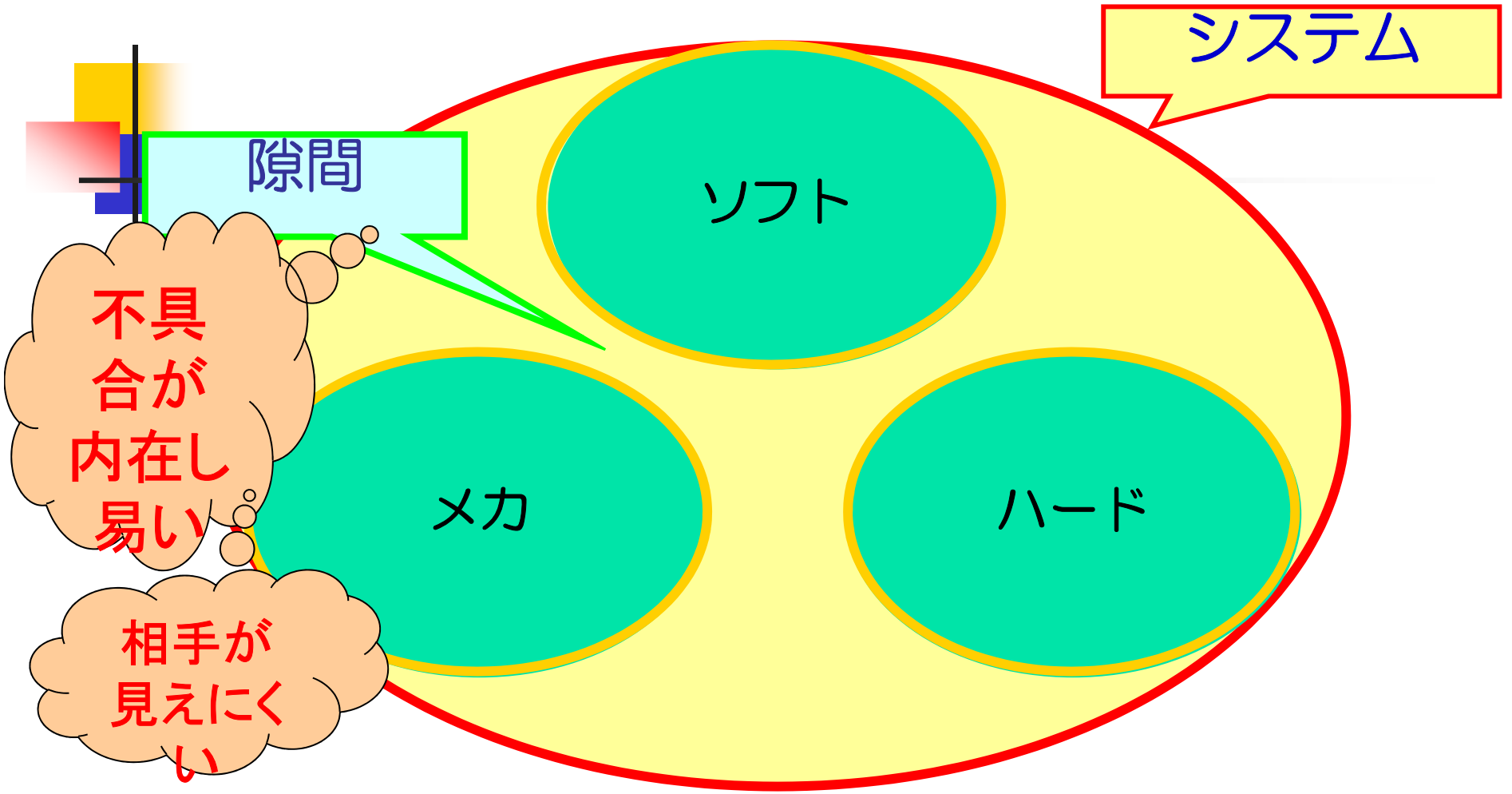


http://ja.wikipedia.org/wiki/

Simulink

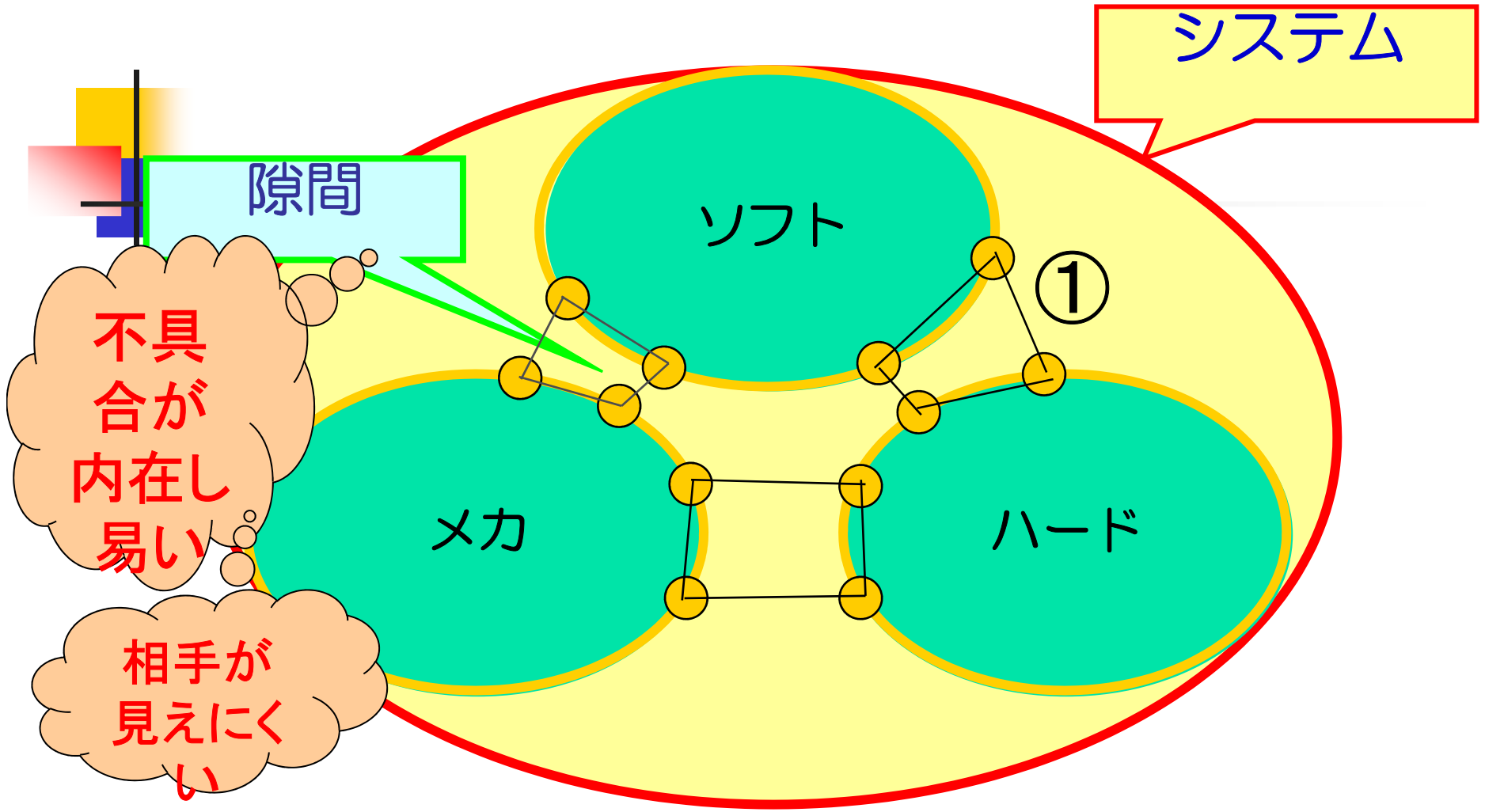


異種混合領域技術の隙間のイメージ図



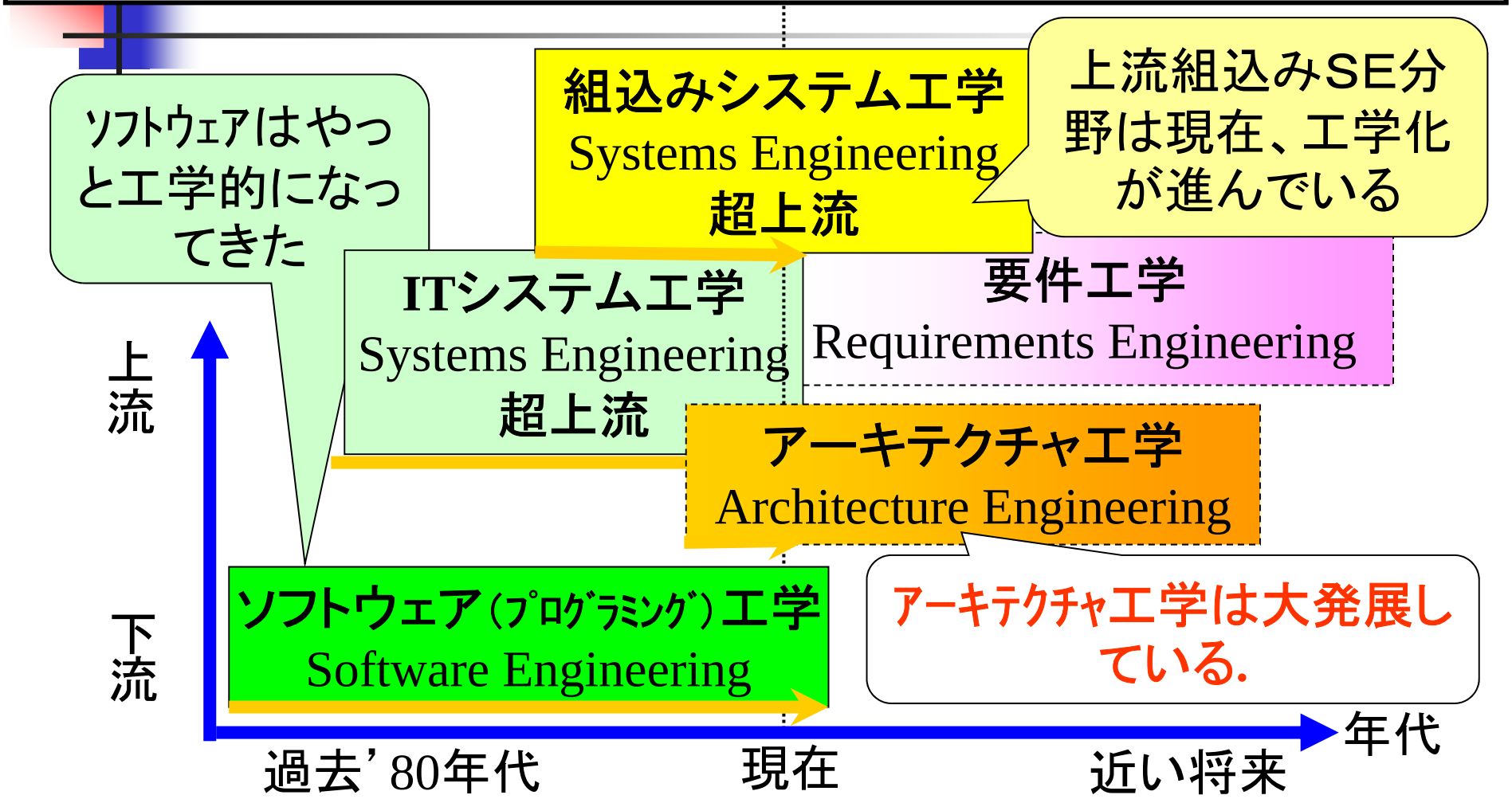
隙間を無くすことが必要
⇒『システム設計』『アーキテクチャ設計』

異種混合領域技術の隙間のイメージ図



隙間を無くすことが必要
⇒『システム設計』『アーキテクチャ設計』

隙間を無くす作業で
⇒『システム設計、記述』『アーキテクチャ設計、記述』が注目されている

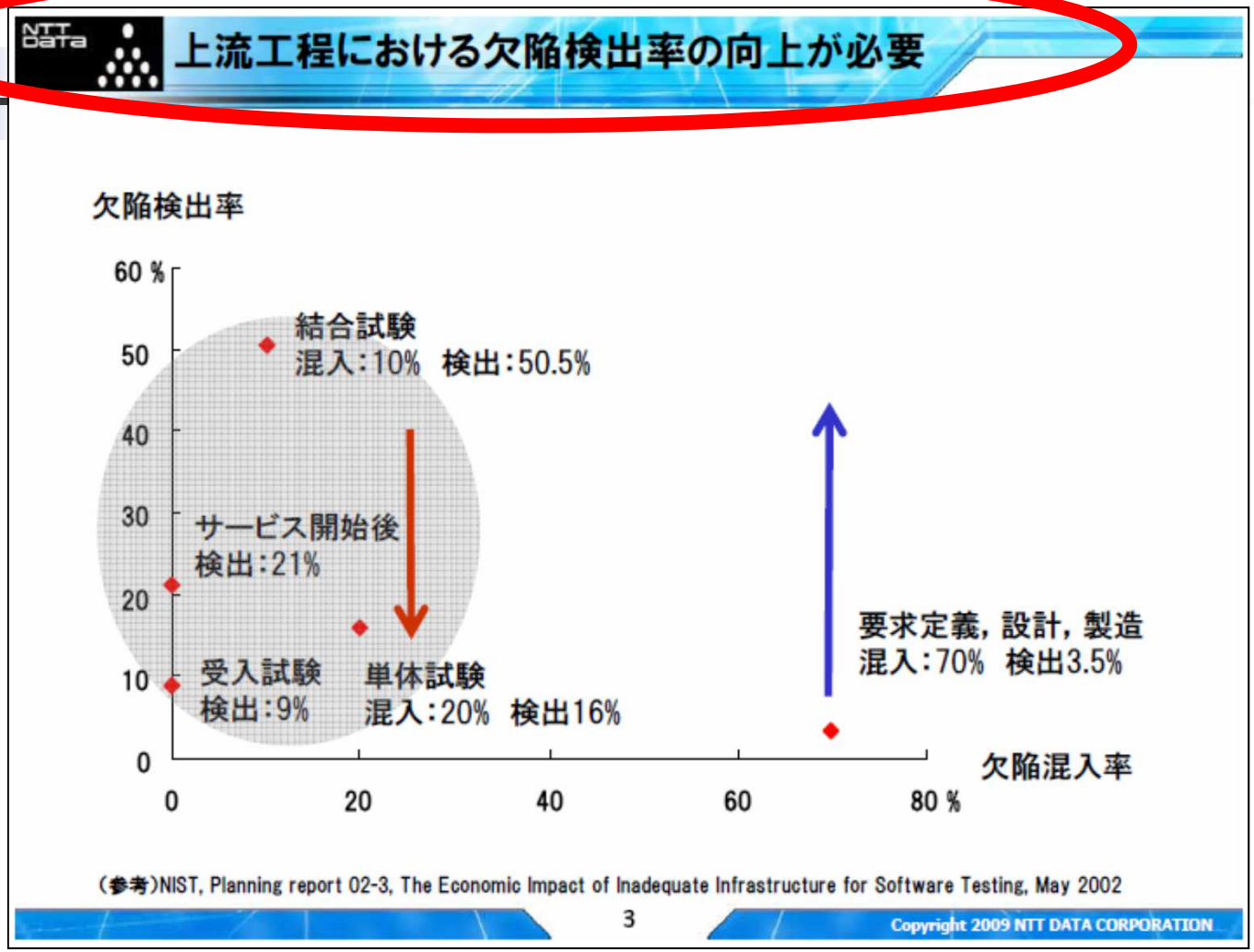


システム/アーキテクチャレベル(早い段階）設計の重要性

現在　名古屋大学
教授

http://stage.tksc.jaxa.jp/jxithp/seminar/data/wocs2009_keynote2_yamamoto.pdf

システム/アーキテクチャレベル(早い段階）設計の重要性



http://stage.tksc.jaxa.jp/jxithp/seminar/data/wocs2009_keynote2_yamamoto.pdf

MBDの根底は何か？／プログラム言語の根底は何か？

- 複雑/大規模化対応要件と、そのため開発の上流シフトが進んで変化。
- MBD/MDDとアジャイル/リーン/摺合せ/プロトタイピング

- MBDとMDD
- 実は仕様書は、なかなか完成していない。(出来るものでは無い)
- 仕様書も、開発過程で完成されるべきもの？
  - 動かしながら、visual化しながら、フィードバックをもらいながら(型情報、メトリックスだけでも重要)、完成に向かっていく。
  - ソフトウェアだけが在ってもシステムが検証できない。
- だから上流MBSE(ﾓﾃﾞﾙﾍﾞｰｽｼｽﾃﾑｴﾝｼﾞﾆｱﾘﾝｸﾞ)
- Matlabはシミュレーションが得意。
  動かして、Visual化して気づくことの大切さ。

仕様書も、開発過程で完成されるべきもの？
トレンド事例

日本流のすり合わせ開発と融和性の高い
開発方法論の国際規格を提案
～ 高信頼なコンシューマデバイスを効率的に開発 ～

2013年11月20日

独立行政法人情報処理推進機構
技術本部 ソフトウェア高信頼化センター

http://www.ipa.go.jp/files/000035408.pdf

http://www.ipa.go.jp/files/000004818.pdf

補足

MBDとMDD

- 情報システムのモデルベース開発の代表であるMDDと組込みシステムのソフトウェアにおけるモデルベース開発の代表であるMBD
- MBDは（MATLAB®/Simulink®などで）実現のためのモデルを作成して、シミュレーションなどを行いながらコードを自動生成する。「広義のMBD」はモデルで開発をすること全般（MDDや狭義のMBDが含まれる）を指し、特にMATLAB®/Simulink®を適用する場合に英語圏では、Model Based Development withMATLAB®/Simulink®と表現するのが一般的である。
- MDDはソフトウェア開発の上流工程からUMLを使って抽象的なモデルを作成し、徐々に詳細化してコードへと落としていく。

（平成23年度モデルベース開発技術部会活動報告書
http://www.ipa.go.jp/files/000026871.pdf）

補足

MBDとMDD

モデル名		MDD	MBD
構造モデル		○	―
分析モデル		○	コントロールフロー相当
シナリオモデル	ユースケース図	○	―
	シーケンス図	○	要求&シミュレーション結果
	コミュニケーション図	○	―
	アクティビティ図	○	データフロー相当
制御対象モデル		―	○
制御装置モデル		―	○

表 2-1　MDD と MBD のモデル比較

（平成23年度モデルベース開発技術部会活動報告書
http://www.ipa.go.jp/files/000026871.pdf）

補足

実は仕様書は、なかなか完成していない。(出来るものでは無い)、さらに仕様書だけでは検証できない？

- 事例:

仕様書通りのソフトウェアであるが、悪路を走ったら、センサーの誤ダイアグを検出した。

- どのくらい悪路を想定しているか、どこかに記載していなければ、検証できない。(根拠)

MBD, MDDへの要望も広がる。
一方で、UML、sysMLは
10年進化していない

15年前の議論

プログラミング記述をさらに抽象化したモデリング（例UML’97標準化）にも欠点があった。

- 当初UMLは、ただのオブジェクト指向記述モデリング手法（欧州認識：そのためUML2.0へ改良折込したが不十分）
  - **コントロールフロー表現が苦手（UML2からSDL概念：アクテイビテイ図追加）**
  - **データフロー表現が苦手（UML2からコンポジットダイアグラム追加）**
  - **部品化が苦手、デザインパターンのみを部品化手段として提供（UML2より　コンポジットダイアグラム追加）**
  - **階層化が苦手（UML2からコンポジットダイアグラム追加したが不十分）**
  - **アーキテクチャ表現が苦手（UML2からコンポジットダイアグラム追加）**
  - **ADL概念が弱い（UML2からコンポジットダイアグラム追加）**
  - **抽象化記述しにくい。（UML2からメタモデルインフラを追加）**
  - **メタプログラミング（ドメイン固有な抽象的な記述）が弱い**
  - **プロダクトライン表現が弱い（未対応：EAST-ADLではフィーチャー図を追加）**
  - **ヘテロジニアス、ハイブリッド記述が弱い（未対応：sysMLではパラメトリック図を追加）**

最近の状況

- この理由もあり、MATLAB/Simulinkの応用が拡大している。
- DSMモデリングや依存性表現モデリングも必要で拡大。
- さらにプロダクトラインのフィーチャー図、説明能力側面でGSN、D-CASEも

クルマの、メカトロニクス30年の進化に見る、今後のデザイン・トレンド

皆さんに伝えたい、30年のトレンド.

クルマの進化の源泉は継続するシリコン革命.

- 車載組み込み制御は、ソフトウェア指向へ

その特徴は,3つのH.

H＆H＆H

- 取り組み領域は,3つのE領域へ. Ｅ＆Ｅ＆Ｅ
- 取り組み領域は,3つのI 領域へ. Ｉ＆Ｉ＆Ｉ
- 取り組み領域は,3つのP領域へ. Ｐ＆Ｐ＆Ｐ
- 取り組み領域は,3つのV領域へ. Ｖ＆Ｖ＆Ｖ

車載組み込み制御の特徴

H&H&H

詳細に言うと

・ヘテロジニアス　（**異種混合**）

メカ、ハード、ソフトが強く絡んだ制御が特徴です

・ハイブリッド　　（**制御が離散系＋連続系の混合**）

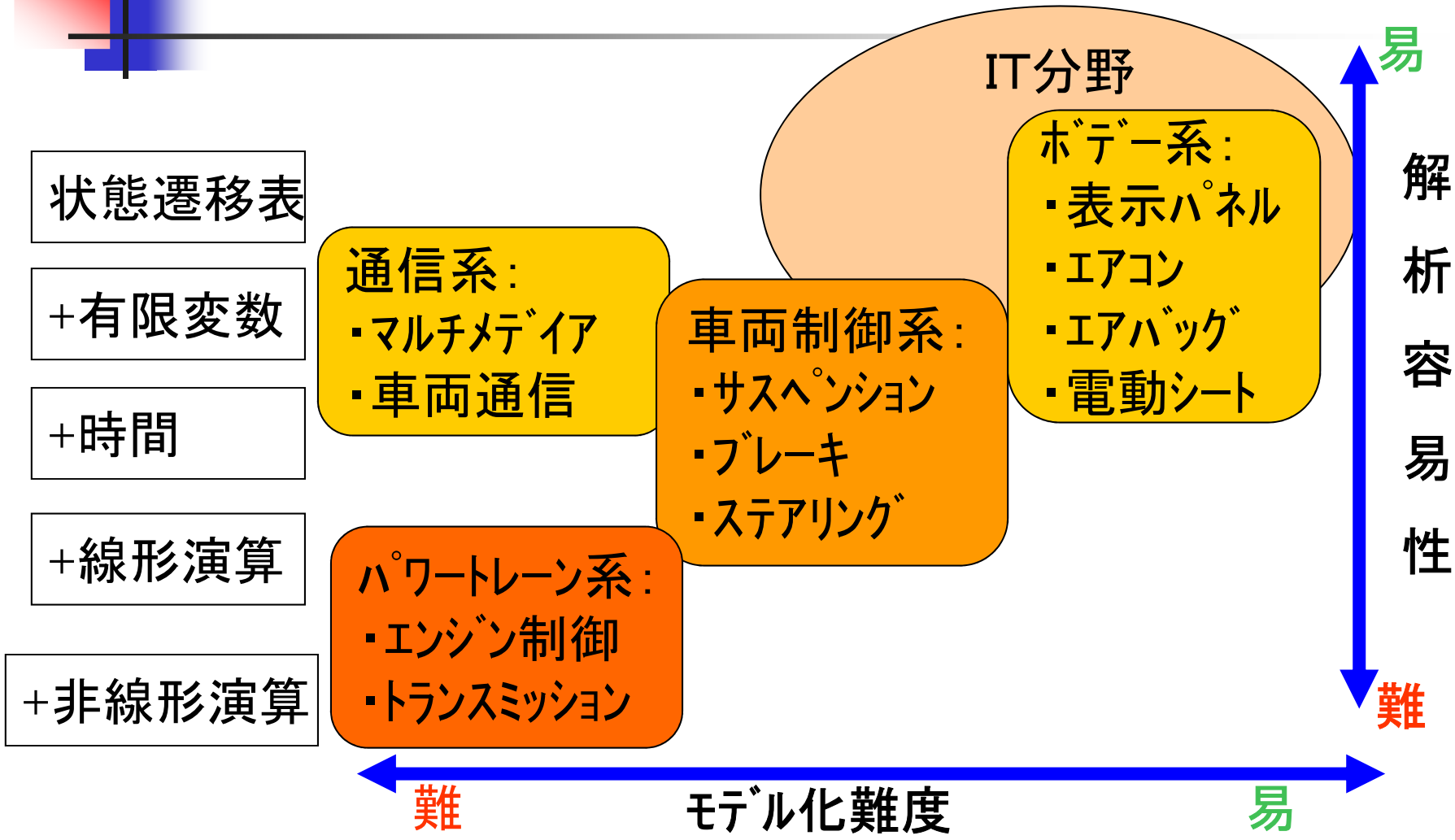
連続値フィードバック制御や状態遷移制御が強く絡んだ制御が特徴です。

・アーキテクチャ　（**階層構造**）

車両全体、システム、サブコンポーネントで連動が強い事が特徴です。

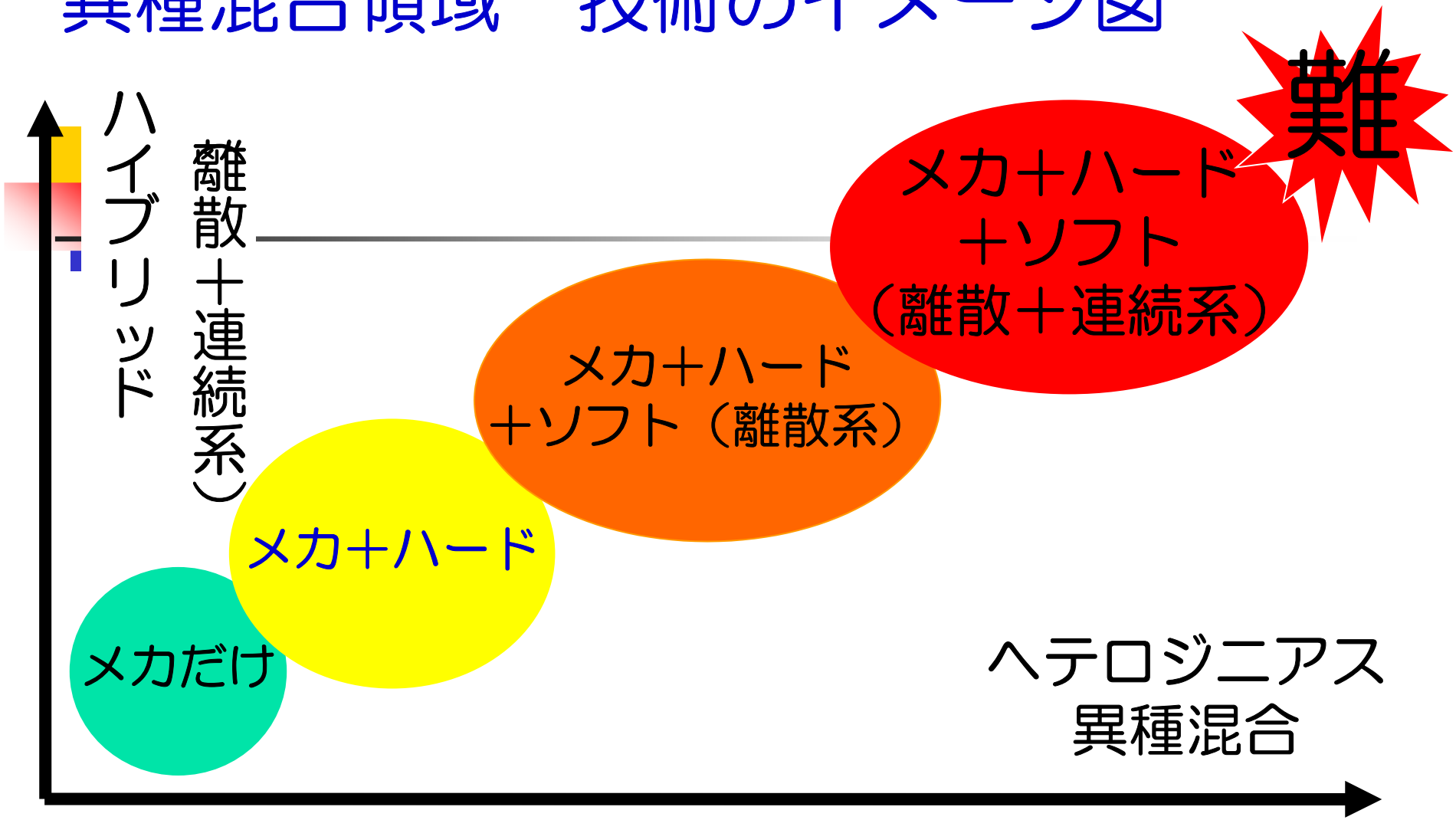
がキーワード

特徴：自動車制御とIT分野の違い



豊田中央研究所論文より

異種混合領域　技術のイメージ図



違う領域の技術が組み合されると、
問題解決は難しくなる

日本品質管理学会

第114回ｼﾝﾎﾟｼﾞｳﾑ(‘07/07/03)

「何なぜｿﾌﾄが組み込まれると品質が悪化するのか？」

車載組み込みデザインの重要な取り組み領域

３＊４の領域

① **E&E&E**へ、 とは

Software **E**ngineering & Architecture **E**ngineering & Systems **E**ngineering の略

② **I &I &I**へ、とは

ISO16949 & **I**SO15504 & **I**SO26262 の略

③ **P&P&P**へ、 とは

People & **P**rocess & **P**roduct_Technology の略

④ **V&V&V**へ、 とは

Verification & **V**alidation & **V**aluation の略

① E&E&Eへの変化
Software Engineering & Architecture Engineering & Systems Engineering

- Eの時代　(Software Engineering)
  - 実装設計が職人芸から、工学的取り組まれるようになる。
- E&Eの時代　(Architecture Engineering)
  - 設計の上流、アーキテクチャ設計が職人芸からアーキテクチャ工学的に取り組まれるようになる。
- **E&E&Eの時代 (E&E　+　Systems Engineering)**
  - さらに上流、超上流から連携して組み込みシステム工学や要件工学が意識され、設計に取り組まれている。

② I&I&Iへの変化
ISO16949 & ISO15504 & ISO26262

- Iの時代　ISO16949 (ISO9000,QS9000)
  - 品質システムを規定し実践する。
- I&Iの時代　ISO15504 (CMMI,Spice,Automotive-Spice)
  - 規定された品質システムの上で、大規模化したソフトウェア指向開発を、具体的プロセスを規定して実践する。
- **I&I&Iの時代　ISO26262(機能安全、IEC61508)**
  - 品質システム、品質プロセス、品質技法を規定して取り組んでいく。　　＊品質技法：　ﾌｫｰﾏﾙ、ｾﾐﾌｫｰﾏﾙﾒｿﾄﾞ、HAZOP、各種検証技法

③ P&P&Pへの変化
People & Process & Product_Technology

参考：BOSCH,"Future SPI at BOSCH":SEPG2003

3つの領域に、常にバランスよく取り組む事が重要。

あえて言うと

- Pの時代（People)
  - 職人芸から、工学的開発へ取り組む。
- P&Pの時代(People & Process)
  - 個人開発から組織開発へ移行し、開発プロセスを規定し(**CMMI,Automotive-Spice)**など導入していく。
- **P&P&Pの時代 (P&P & Product_Technology)**
  - さらに、再利用拡大など効率化のため、製品内部へ**AUTOSAR**などのプラットフォーム構造やその上でプロダクトライン開発から分析された構造を織り込んでいく。

④ V&V&Vへの変化

Verification & Validation & Valuation

参考　V&V：http://blues.se.uec.ac.jp/mt/swtest/

V&V&V：http://computingscience.nl/

- Vの時代
  - Ｖ字プロセス(ISO12207)を守り、ウオータフォールモデルで検証していく。
- V&Vの時代　(Verification & Validation)
  - V字プロセス上流から、各工程で順次検証しながら、完成度を早期に高めていく。 **超上流のシステムレベルから、**取り組む。（IV&Vへも進化している）
- **V&V&Vの時代　(V&V + Valuation/Variation)**
  - V&V活動をしながら、開発の当初から**プロダクトライン開発**に取り組んでいく。

再度掲載

パラダイムシフト

- 10年毎のパラダイムシフトが起きている。
- 今後も継続的に起きていく、と想定される。
- 効率的で、高信頼な開発手法変革が求められている。
- パラダイムシフトは、インターネットに潜伏して急拡大するため、見えにくくなっている（サイレントパラダイムシフト）

②機能安全(ISO26262)が定着してきて、何に困って、今何に取り組まれているのか？

- Part2：
  - ハザード分析の仕方？
  - 要件/設計のセミフォーマルモデリングの書き方？
  - 安全コンセプトSCの書き方と粒度？

- Part8:
  - セーフテイケースのまとめ方/説明の仕方？

機能安全

IEC61508　（’98-’00制定）
JISC0508　（’00制定）
ISO26262　（’09/7DIS公開
　　’11/10規格化 ）

機能安全とは？

機能安全（Functional safety）

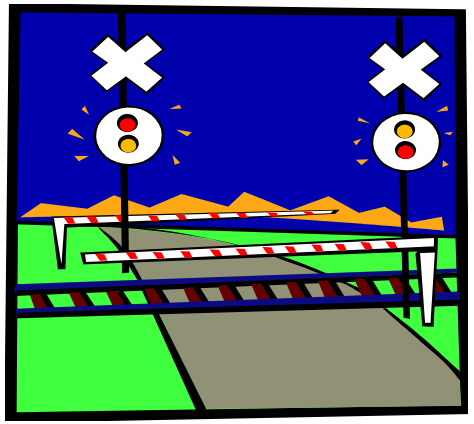
新しい用語、　本質安全と対比する用語。

本質安全と機能安全

本質安全：立体交差にすれば、踏切を渡って事故に遭遇する可能性はない　→　根源からリスクを無くして達成する安全

機能安全：現実は線路をすべて立体交差にすることはできない

本質安全を達成できない場合、信号や列車停止装置等の周辺安全機能により相対的なリスクを軽減し、許容リスク以下にする安全



機能安全とは？

機能安全（Functional safety）

新しい用語、　本質安全と対比する用語。

本質安全と機能安全

本質安全：立体交差にすれば、踏切を渡って事故に遭遇する可能性はない　→　根源からリスクを無くして達成する安全

機能安全：現実は線路をすべて立体交差にすることはできない

本質安全を達成できない場合、信号や列車停止装置等の周辺安全機能により相対的なリスクを軽減し、許容リスク以下にする安全

機能安全（Functional safety）は、本質安全と対比する。しかし機械安全の考え方もリスクが基本の考え方に既に移行し、機能安全の考え方が主となっている。

例：機械類の安全性－リスクアセスメントの原則　ISO14121(2000年制定)

機能安全規格 ISO26262 とは

- 欧州が提案し、広く検討策定された、自動車向け電気/電子/プログラマブル装置を搭載した車両を対象とする安全規格
- IEC61508規格に基づいて2011年11月に正式に規格化
- 安全度水準（ASIL; Automotive Safety Integrity Level）を4段階（ASIL D[高] ～ A[低]）で定義
- 開発製品の安全性を示す説明責任を要求
  - 開発プロセスの確立
  - 技術的な方策による安全性の実現
  - 安全性を客観的に説明するエビデンスの整備

ISO26262
E／Eシステム（電気/電子プログラマブル電子系）を含む安全関連系システムに適用する機能安全

- 第１部：用語集
- 第２部：マネジメント
- 第３部：**ASIL**の決定と機能安全コンセプトの作成
- 第４部：システム開発
- 第５部：ハードウェア開発
- 第６部：ソフトウェア開発
- 第７部：量産以降
- 第８部：サブプロセス、共通規定群
- 第９部：**ASIL**のハンドリングに関する手法など
- 第１０部：**ISO26262**のガイドライン

規格の構成、内容

Part 1. 用語集

Part 2. マネジメント

開発に関連する組織の管理要件、
開発時の管理要件、製造開始後の
管理要件が記載
・安全ﾗｲﾌｻｲｸﾙ

Part 3. ASILと機能安全構想

ｼｽﾃﾑ開発の前段階で、以降の開発
ﾌｪｰｽﾞにおける前提条件を決定する
要件を記載
・ﾊｻﾞｰﾄﾞ分析、ﾘｽｸｱｾｽﾒﾝﾄ(H&R)
・ASIL決定方法

Part 4. システム開発

・製品開発におけるｼｽﾃﾑﾚﾍﾞﾙの
開発要件
・安全機構による対策
・ﾊｰﾄﾞ(ﾒｶ、電気)・ｿﾌﾄｲﾝﾀｰﾌｪｰｽ(HIS)

Part 5. ハード開発

・ﾊｰﾄﾞｳｴｱ(ﾒｶ、電気)開発ﾌﾟﾛｾｽに対する要件
・HWｱｰｷﾃｸﾁｬ評価
・安全目標を侵害する確立の評価

Part 6. ソフト開発

・ｿﾌﾄｳｴｱ開発ﾌﾟﾛｾｽに対する要件
・ｿﾌﾄｳｴｱﾚﾍﾞﾙの各種ﾁｪｯｸ

Part 7. 量産以降

製造工程、市場運用から廃棄までに
関する要件を規定

Part 8. ｻﾎﾟｰﾃｨﾝｸﾞﾌﾟﾛｾｽ

全章に関わる共通的なﾌﾟﾛｾｽを
まとめたもの

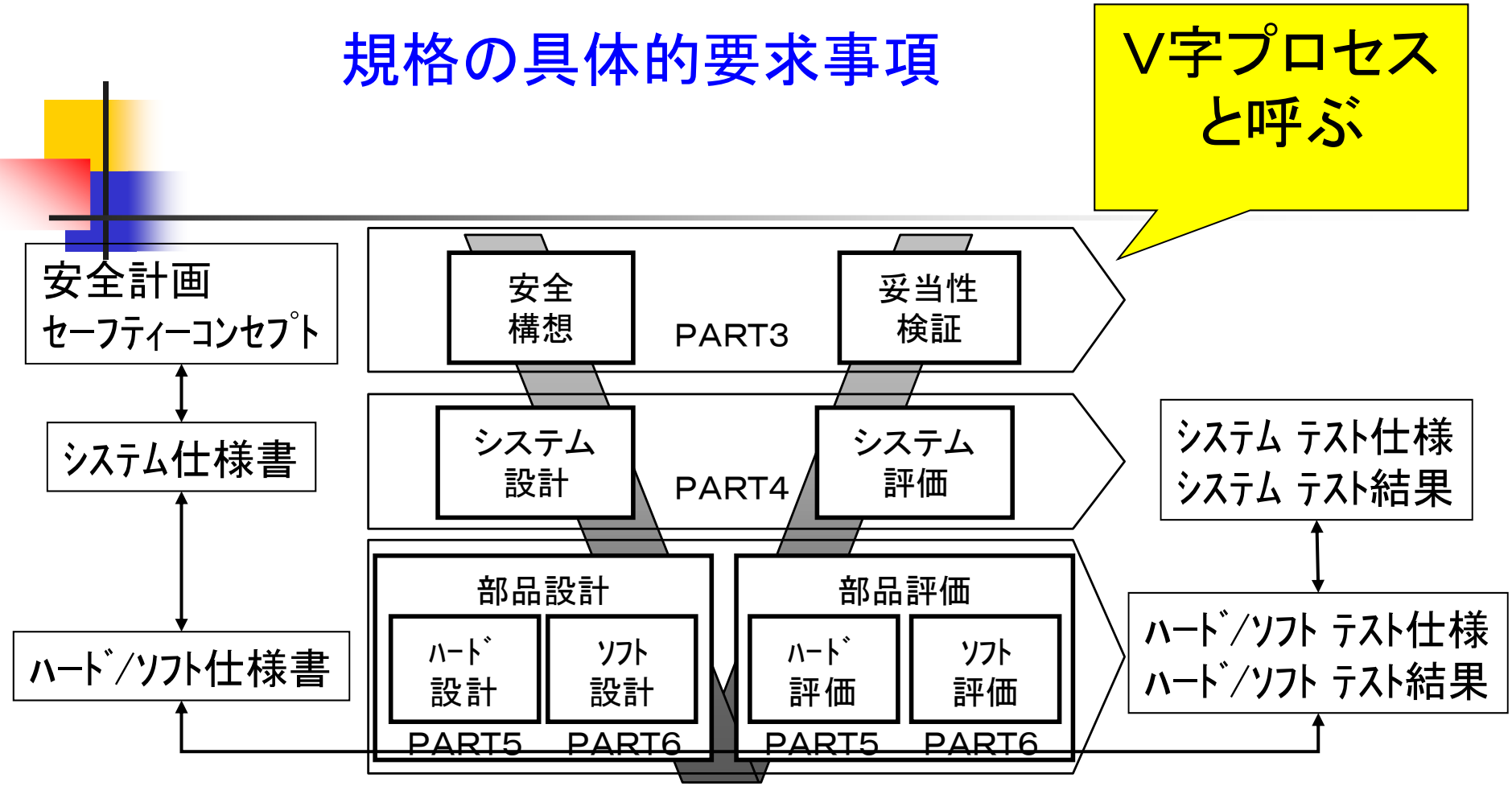
Part 9. ASILのﾊﾝﾄﾞﾘﾝｸﾞ他

ASILまたはsafetyに関係する要件
をまとめたもの

Part 10. ｶﾞｲﾄﾞﾗｲﾝ

ISO26262を適用する際のｶﾞｲﾄﾞﾗｲﾝ
(要求事項は記載なし)

規格の具体的要求事項



成果物のトレーサビリティ

- 作業成果物は、システムから部品まで一貫性、正確性、遵守性が要求される
- カーメーカーの監査では、成果物で機能安全目標達成の説明責任がある

機能安全はすべてをカバーしているわけではない

機能安全を補完する手法の必要性例

- 自動車機能安全規格の解釈はJasPar,JARI活動で共有されてきた。
  そこから踏み込んだ相場観は、すべては議論できない。　　（各社活動）
- 一方で、ある程度相場観を支援できないか、提起が始まっている。

- GSN, D-Case, SafeML, SCDL

GSN: Goal Structure Notation (2011)

システムが達成すべき目的や性質について、その達成を導く方法・思考を　可視化する際に用いる記法

D-Case: Dependability Case （2013）

GSNを利用した合意の表記法（一般社団法人 ディペンダビリティ技術推進協会：IEC 62853/62741,IEC60300-1,ISO/IEC15026)

SafeML: Safety Modeling Language

産総研活動：安全関連情報をコミュニケーションよく交換するモデリング記法

SCDL: Safety Concept Description Language（2013～）

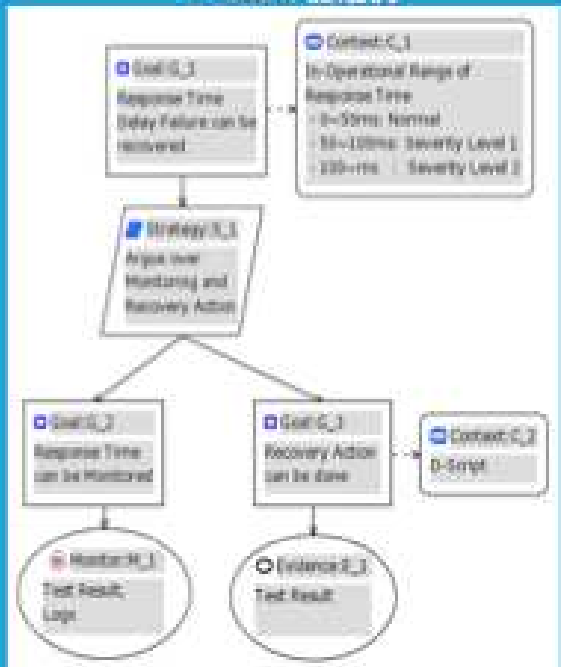
エレメントの表記法、安全要求の表記法、それらを統合した安全コンセプトの表記法

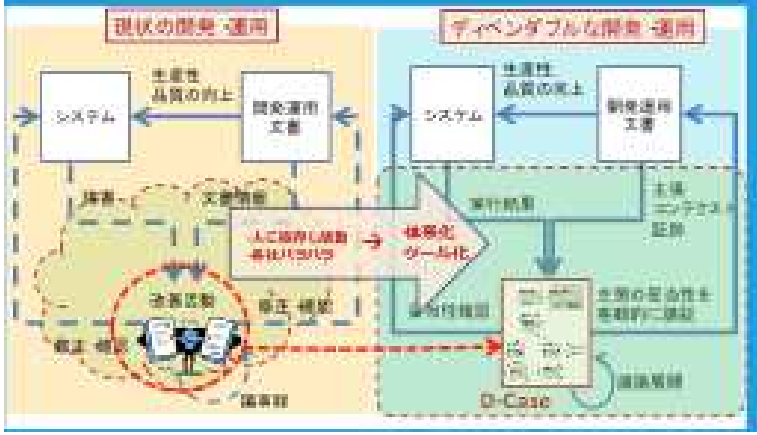
DCASEの基本であるGSNは非常に簡易な基本表記
(4種の構造を結ぶ:2011年イギリスで提案)であり、マインドマップの様に一般化すれば、白板を前にした討議でも利用可能

D-Case

- GSN (Goal Structuring Notation) をベースとした合意形成のための表記法
- GSNを拡張し、システムの運用状況の監視と詳細な記録を実現
- 開発・運用を含む全ライフサイクルでのステークホルダー間の合意形成
- D-Case記述の合意に基づく変更履歴が説明責任遂行を支援

D-Caseの記述例





- D-Caseの活用により、属人性の強い改善活動から、体系化・ツール化されたプロセスへ

http://deos.or.jp/index-j.html

③何が停滞し、何がブレークスルーしているのか？

- モデリング標準化の停滞、認識遅れ、
- コンパイラ技術/CAEの進化、
- アイデアの積み上げ。

モデリング標準化の停滞、認識遅れ

- UML1.0(‘97)、2.0大幅変更**(‘04)その後大幅変更無し**
  - ローエンドMARTE、fUML、EAST-ADLの動きは活発だった。
- sysML1.0**(‘04)その後大幅変更無し**
- UML2.0、SysML1.0の功労者は両方ともCris Kobryn氏

- xUML.org　vs　U2Partner.org　/　SysML.org

ステファン・J・メラー氏　Cris Kobryn氏 http://kobryn.com/

https://en.wikipedia.org/wiki/Stephen_J._Mellor

- **Cris Kobryn氏の意思を引き継ぐ人が現れていない**
- **日本が次世代をリードもありえる！**

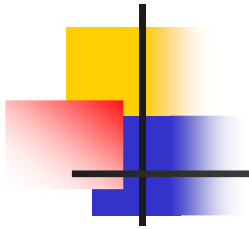
モデリング標準化の停滞、認識遅れ

- プロダクトラインのフィーチャー図を織り込む作業が進んでいない
- 状態遷移表は織り込まれるのか？
- DSM提案（状態遷移表同様表的表現）は？
- Matlab/Simulinkスタイルガイドは進んでいるが、sysMLスタイルガイドが標準化に至っていない（safeML提案あり）
- MBSEを補完するGSN、D-Case、SCDL（SCN）などの提案が始まって、融合が歩みだした。
- MBSEもMBD/MDDも安全分析をどう支援／織り込むかは途上（STAMP、安全分析）

コンパイラ技術/CAEの進化

- 2010年頃からプログラム言語提案が活発化
  - Javascript/Webエンジンの競争過熱。Google/Javascript-V8
  - Microsoft/F＃，Google/Go，Apple/Swift、Firefox/Rust ,MIT/Julia、Oracle/Java-Truffle

- Meta情報解析が進化
- AST(Abstract Syntax Tree)構文解析が進化
- JIT(Just InTime Compiler)を含め型情報解析推定、フロー解析が進化
- コンパイラ基盤LLVM(Low Level Virtual Machine)が発展、普及



アイデアの積み上げ

- プログラム言語ではオブジェクト指向にアーキテクチャ指向(ADLまたはDependency Injection)を組み入れ、関数言語指向を織り込むアイデアや、さらにプロトタイピング指向といえる型推定（SoftTyping、DuckTyping）や、会話的なReplを織り込む技術が見えてきた。
- そしてオープンイノベーションの時代に必要な
  コミュニテイ指向が広まってきた。

- いまとなっては40年前真空管時代に提案された
  C言語が、新たな息吹で変わる時に来た、といえます。

④じゃあ明日からどうする？

“より上流”はエレキだけではない世界

機械や電子分野のコミュニケーション、連携、は、共通認識できる言葉、表現が大切。

＝システム記述、アーキテクチャ記述／技法

これを“ヘテロジニアス”な世界と言います。

再度掲載

じゃあ、C言語開発の現状を補完するには、明日からまずどうする？

- C言語は機能の低い、危険な言語です。（機能安全の認識）
- スタイルガイドとガイドチエッカーを利用しましょう （機能安全の認識）
- 静的解析ツールを利用しましょう（型整合を含む）

- **設計にはモデリングを併用しましょう。（機能安全の認識）**
- 従来のモデリング（例UML）を併用しても、見える化は、十分ではありません。（UMLは2.0以上またはsysMLを利用しましょう）さらに要件周辺でGSN,D-Case.）
- ツールで補完しましょう
  - 上記　style_guide、static_analysisを含む
  - 部品化をグローバル変数で支援するツールがあります（Matlab/Simulink,Autosar周辺）
  - アーキテクチャADLを意識した見える化ツールを利用しましょう。
- 広義の階層化、依存関係の見える補完手段など、UMLでは不十分で、さらに改良（イノベーション）が重要です。
  - DSM手法は新しいモデリング提案ともいえます。利用しましょう。（matrix、表表現は欧米ではモデリングと言いませんが）
  - プロダクトライン表現ではフィーチャー図が必要です。（未だUML標準化されていませんが、欧州提案EAST－ADLでは標準提案されています）

再度掲載

じゃあ、C言語開発の現状を補完するには、明日からまずどうする？

- 状態遷移図で完全であっても、状態遷移表で確認すると、ユーザ目線では漏れがあります。　状態遷移表で確認をしましょう。
- 状態遷移表で完全であっても、安全分析をすると、システム目線では漏れがあります。　多種の安全分析をしましょう。
- 派生開発、車種展開ではプロダクトライン開発を意識し、フィーチャー図で確認をしましょう。
- これらはC言語開発の課題ではありませんが、カスタマー目線の基本課題です。
- **モデリングsysML**は上流要件を記載する場ですが、結果を記載する場であって、上記を推進する道具ではありません。**良いテンプレートが必要です**。さらに要件や設計が妥当である説明が求められてきて、わかりやすく、**説明のためのモデル表記を併用する**方向になってきています。

じゃあ、C言語開発の現状を補完するには、明日からまずどうする？

- 従来のモデリング（例UML）を併用しても、見える化は、十分ではありません。（UMLは2.0以上またはsysML、MATLABを利用しましょう）。フィーチャー図、状態遷移表も利用しましょう、説明のためのモデル表記を併用しましょう（GSN,D-CASE）。
- プロトタイピング－ リファインメント－ C言語実装のアイテレーションが本来あるべき姿と確信します。
- プロトタイピングを意識して、そうであるべきタイミングでは、プロトタイピングに便利な言語、ツールで取り組みましょう。 しかしその言語、ツールで最後まで量産に利用するわけではありません。
- 開発スルーで異なる言語／モデリングを併用するわけですが、トータル効率では、効果的。
- MATLAB開発が、ﾓﾃﾞﾙﾍﾞｰｽ開発―C言語生成量産であり、その意味で同様。

再度掲載

機能安全はすべてをカバーしているわけではない

機能安全を補完する手法の必要性例

- 自動車機能安全規格の解釈はJasPar,JARI活動で共有されてきた。
  そこから踏み込んだ相場観は、すべては議論できない。　　（各社活動）
- 一方で、ある程度相場観を支援できないか、提起が始まっている。

- GSN, D-Case, SafeML, SCDL

GSN: Goal Structure Notation (2011)

システムが達成すべき目的や性質について、その達成を導く方法・思考を　可視化する際に用いる記法

D-Case: Dependability Case （2013）

GSNを利用した合意の表記法（一般社団法人 ディペンダビリティ技術推進協会：IEC 62853/62741,IEC60300-1,ISO/IEC15026)

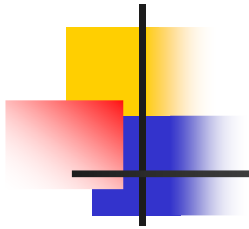
SafeML: Safety Modeling Language

産総研活動：安全関連情報をコミュニケーションよく交換するモデリング記法

SCDL: Safety Concept Description Language（2013～）

エレメントの表記法、安全要求の表記法、それらを統合した安全コンセプトの表記法

機能安全の補足(1)



- 機能安全は、車両レベルの安全要件、設計から
  システム、コンポーネント、電子回路、ソフトの設計の
  手順をベストプラクテイスstate of artでまとめている。

- そこに関係する多数のステークホルダ、企業、組織、
  メンバーの間で**交換**する必要ドキュメントは100以上
  で、それをすべて固有の名前で定義し、どのように
  処理すべきか記載している。
  (プロセス、ガイド、テンプレート)

機能安全の補足(2)

- 文書(要件）交換の基本は、受け取ったら鵜呑みにせず、必ずレビューを含む要件確認、理解、その段階での検証を行い、交換元（ステークホルダー）にフィードバックをすること。 フィードバックされたことを必ず確認し、次の工程へ進む。

  (進み方は任されている、 後述のポイントがある）

- これは、複数の階層を経て上流へフィードバックされることも、当然重要で必要と記載されている。

⇨ アジャイル、TPS（トヨタ生産方式）、リーンプログラミング、摺り合わせ、コンカレント設計、とも合致。

機能安全の補足(3)

アジャイル的キーワードを実現するための根底で重要で、当然機能安全で重視されている事項

-これらは時代で進化しているため、アジャイルが提案された頃(2000)と情況は変化している-

- 構成管理、文書管理、課題管理、変更管理、トレーサビリテイ管理、ベースライン管理が大切。

- コンピュータの進化によるシミュレーション利用、単体テスト、レビューなどV＆Vの手法、ツールの進化、活用が大切。

じゃあ明日からどうする？

- プロトタイピング－ リファインメント－ C言語実装のアイテレーションが本来あるべき姿。
- うまく回っていますか？会話的開発IDEも本当にうまく回っているか、よく考えて対応しよう。
- その上で、MBD，MDD、シミュレーション、IDE、言語、ツールを組み合わせて選択しよう。
- ひとつの言語で、すべて(フルスタック)開発するのがベストと言えるか、場面場面で自問自答してみよう。
- ADLを意識しアーキテクチャ設計を意識しよう（機能安全でも意識している）。
  プログラム＝アーキテクチャ＋振る舞い
- C言語で最初から最後まで開発にとどまらないように、プロトタイピングを意識しよう。

じゃあ明日からどうする？

開発初期は、

- 早くフィードバックすることを意識
  ー＞プロトタイピング、会話型

- キーワードは、動く、見える（ヴィジュアライズ）、反応をもらう
  （型情報、コンパイルエラー情報、静的解析情報、メトリックス）

- MBD，MDD、MBSEを活かす。 アジャイルも受け入れられる時代（シミュレーションや構成管理が進んだ）。Matlab/simulinkはもちろんお奨めですが、私は、プロトタイピング段階では、Ruby言語等も適していると感じています。

- サイレントパラダイムシフトは10年ごとに来ます。自ら情報を取りに行って、備えましょう。

21世紀に求められる人材
グローバル対応できる人、新技術に怖がらずどんどんチャレンジできる人、自分で切り開ける人、の次は

- “擬似仮想体験からの習熟能力”が重要になっている

- 情報収集の頭でっかちではなく、インターネットの世界の中で、あたかも自分が経験したような、スキルに昇華させる能力

持ち帰るものは、見つかりましたか？

見つかったら幸いです

suzumura-nobuyasu@aisin-comcruise.com　鈴村延保