C言語だけでの開発の現状を打破しよう

-プロトタイピングーリファインメントーC言語実装のアイテレーション が開発の本来あるべき姿、との立場で提起します-

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

アイシン・コムクルーズ　鈴村延保

C言語だけでの開発現状を打破しよう

事前案内

C言語は、システムプログラミング言語と分類され、2万行の小さなUNIXシステム記述から始まったアセンブラ代替言語です。（40年前の古い言語です）

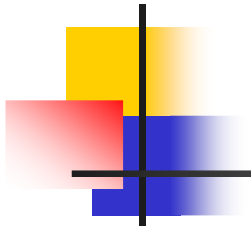
最近、対峙してフルスタックプログラミング言語は何がよいか？と言われるように、組み込みアプリケーションをC言語一辺倒で開発するには、大規模になりすぎた、と言われます。
アセンブラからC言語開発へ移行する時節を経験してきた身には、少しでも早く次世代のプログラミングへ移行すべき状況に見えます。Java、C++に移行しないからだ、という意見もオブジェクト指向の欠点から不十分、といわれます。

モデルベース開発MBDは自動車分野で応用の進む次世代への移行と見えますが、MBDだけですべて出来るとも見えていません。
またMBDとMDDも違いがわかりにくい状況です。

少し立ち戻って”C言語開発では何が足りないのか”、”MBDで何ができるのか”、”足りないか”などを整理します。
今ホットな話題のAppleの新プログラミング言語 Swiftの分析も織り交ぜて、”プロトタイピングーリファインメントーC言語実装のアイテレーションが本来あるべき姿”という立場で変化している背景や流れを含めて、世界で議論されている事実を整理し、共通認識を共有していきます。

”じゃあ明日からこれが活用出来る”という事例を皆で多く持ち帰りましょう！

この提案で討議、共通認識整理したい背景のキーワード



- 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）、
- オブジェクト指向の欠点、 アーキテクチャ指向、 Dependency-Injection 、
- 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、
- IDE ：Interactive Development Environment, Executable Specification、
- リーンプログラミング、 REPL、 Playgrounds （Swift)
- プログラムのコントロールフローと データフロー側面

進め方：（イントロ5分,10分提起＋5分討議）＊５

- **①イントロ： 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
  - 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 とは？
- **②C言語の欠点と言われている点と当面の対応（意識）**
  - C言語の弱さはMISRA－C指摘にすでに出ているが、それだけではない
  - フルスタックとは？
  - 開発に各種支援ツールが補完されていて、もちろん昔のC言語だけのままではないことは承知だが、それでも時代遅れになっていないか？
  - オブジェクト指向の欠点と対応の方向
  - 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？
- **③Apple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？(キーワードは？）**
  - Rust, Rubyを意識している、その本質は？
  - REPLとPlaygrounds
  - RuPyデイレクション
- **④パラダイムシフトの認識、 さらにサイレントパラダイムシフトの認識 （ちょっと外れますが重要です）**
  - オブジェクト指向、アーキテクチャ指向、MBSE、コミュニテイ指向、アジャイル、リーンプログラミング、関数言語の意味はデータフロー指向？
- **⑤モデルベースMBD、MDDは課題にどのように対応しているのか？**
  - MBDとMDD
  - 実は仕様書は、なかなか完成していない
  - 仕様書も、開発過程で完成されるべきもの？
  - だからMBSE
  - 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、
- **⑥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

自己紹介

1977年　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。ASEAN連携も関心事。

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

- 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
  - 「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　」
- 九州工大　産官学連携講座　非常勤講師　(’08～）
- 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～

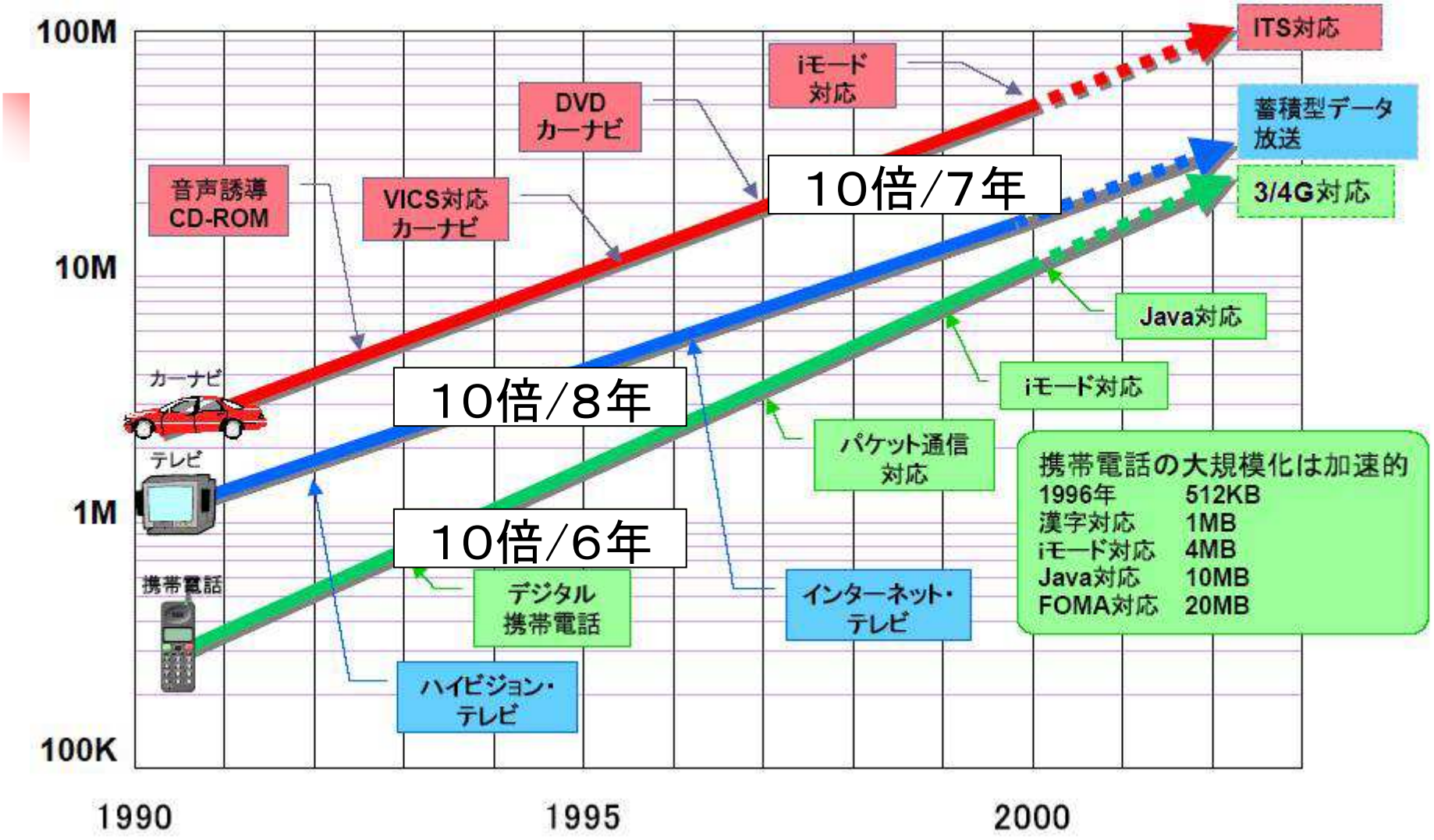
①開発がどんどん“非会話的”になっている、という認識

- 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。
- 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　とは？

本来の姿、と
思います。

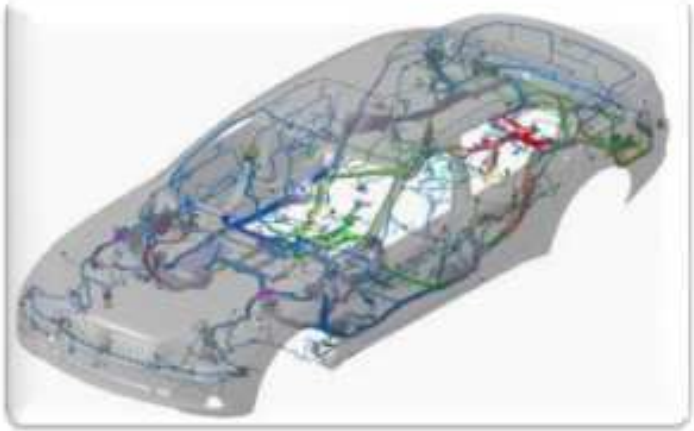
かつてTurbo-C　1パスコンパイラと
スクリーンエデイタの統合環境(IDE)の発明
が隠れた革新だった。

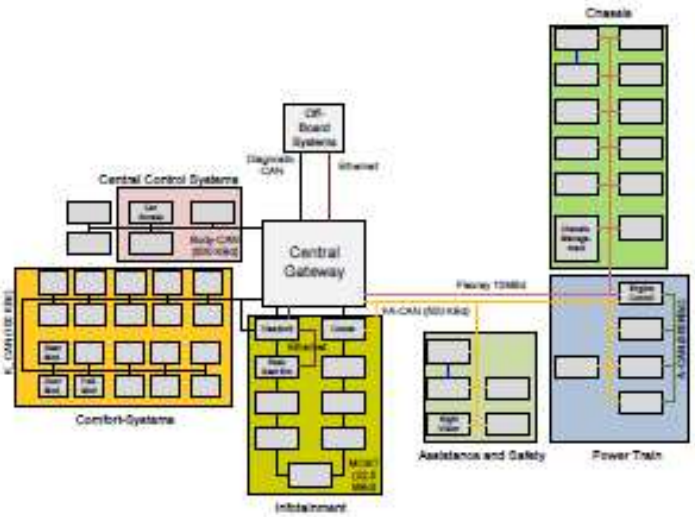
大規模化する組み込みソフトウェア



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

AUTOMOTIVE E/E-ARCHITECTURE TODAY





7 series wiring harness

Length	2751 m
Connecting Plugs	520
Weight	43 kg

Electronic Control Units (ECU)

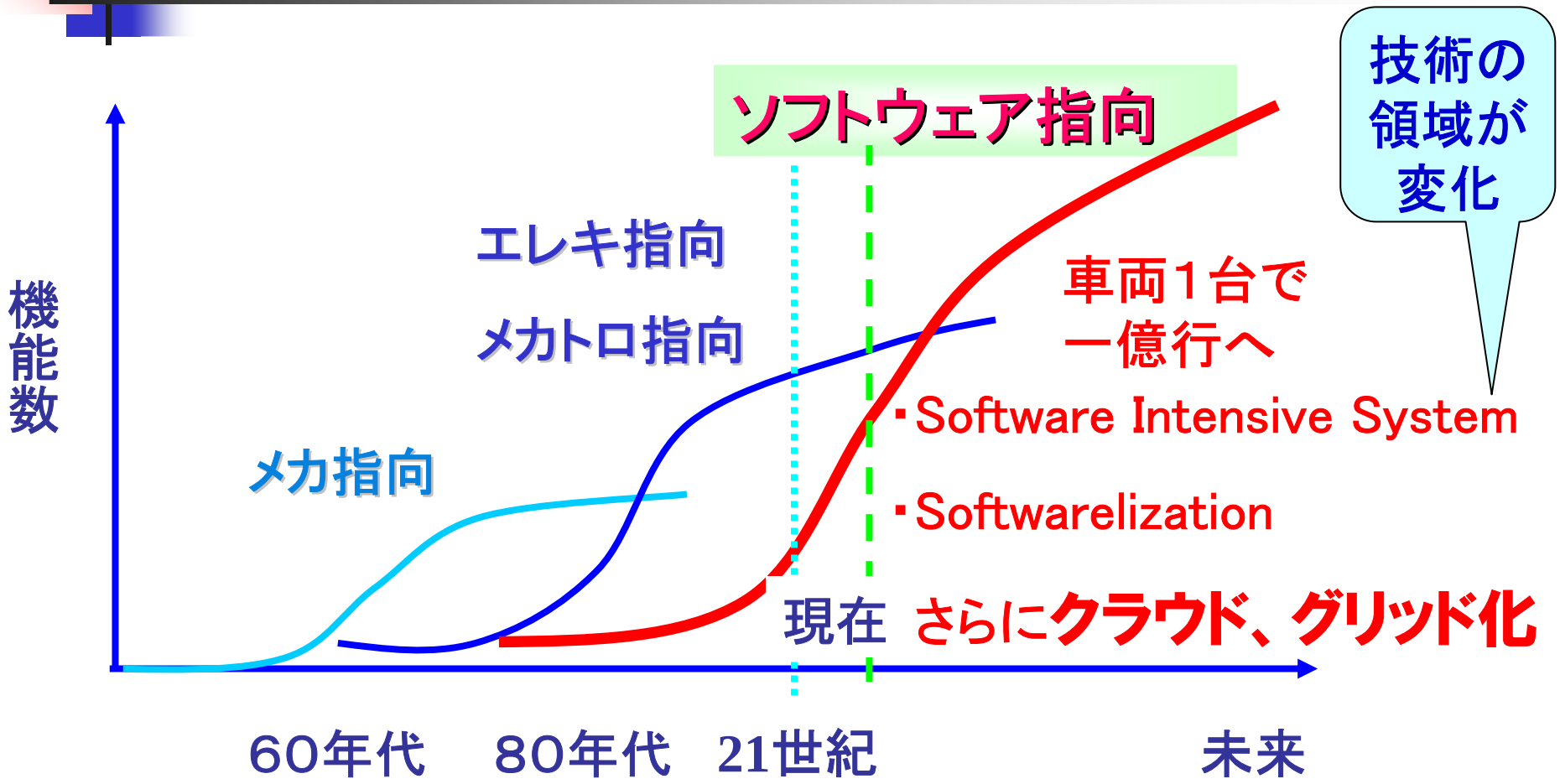
No. ECUs	28 … 74
CPUs	ca. 230
GPUs	> 5
Power PCs	3
Busse	CAN, LIN, MOST*, Flex Ray, Ethernet

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）

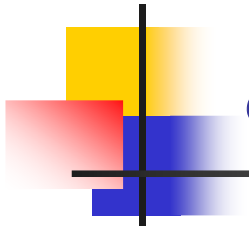
技術の
領域も
変化

効率的で高信頼な開発手法変革が求められている

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？			

②C言語の欠点と言われている点，と当面の対応（意識）



- フルスタックとは？
- C言語の弱さはMISRA－C指摘にすでに出ているが、それだけではない。
- 開発に各種支援ツールが補完されていて、もちろん昔のC言語だけのままでは無い、ことは承知だが、それでも時代遅れになっていないか？
- オブジェクト指向の欠点と対応の方向。
- じゃあ、C言語開発の現状を補完するには、あしたからまずどうする？

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

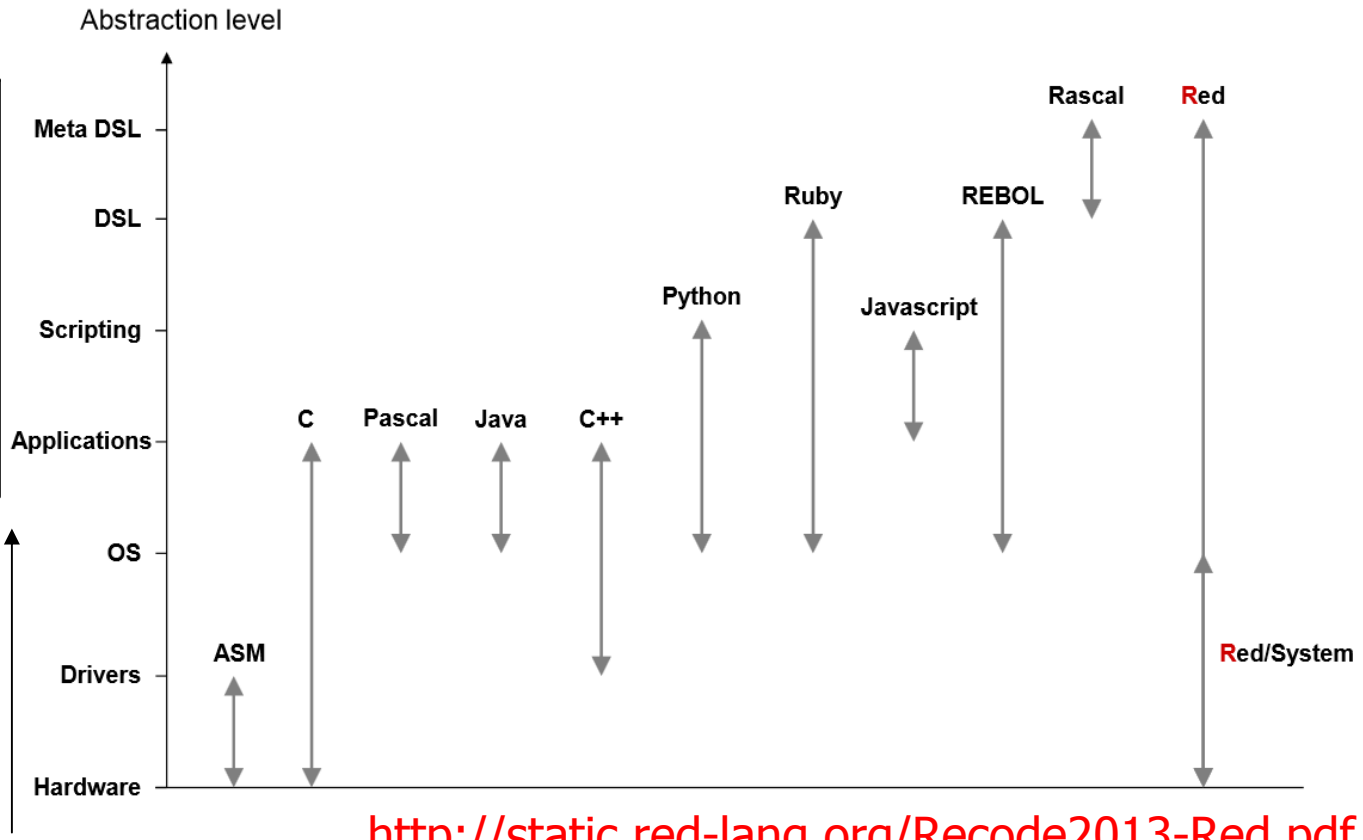
階層ともスタックとも呼ばれている

階層	詳細
ｱﾌﾟﾘｹｰｼｮﾝ	統合システム
	ｼｽﾃﾑ、ELEMENT
	APP-COMPONENT
PF	PF-API
	MIDDLEware
HAL	HAL

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
ｱﾌﾟﾘｹｰｼｮﾝ	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

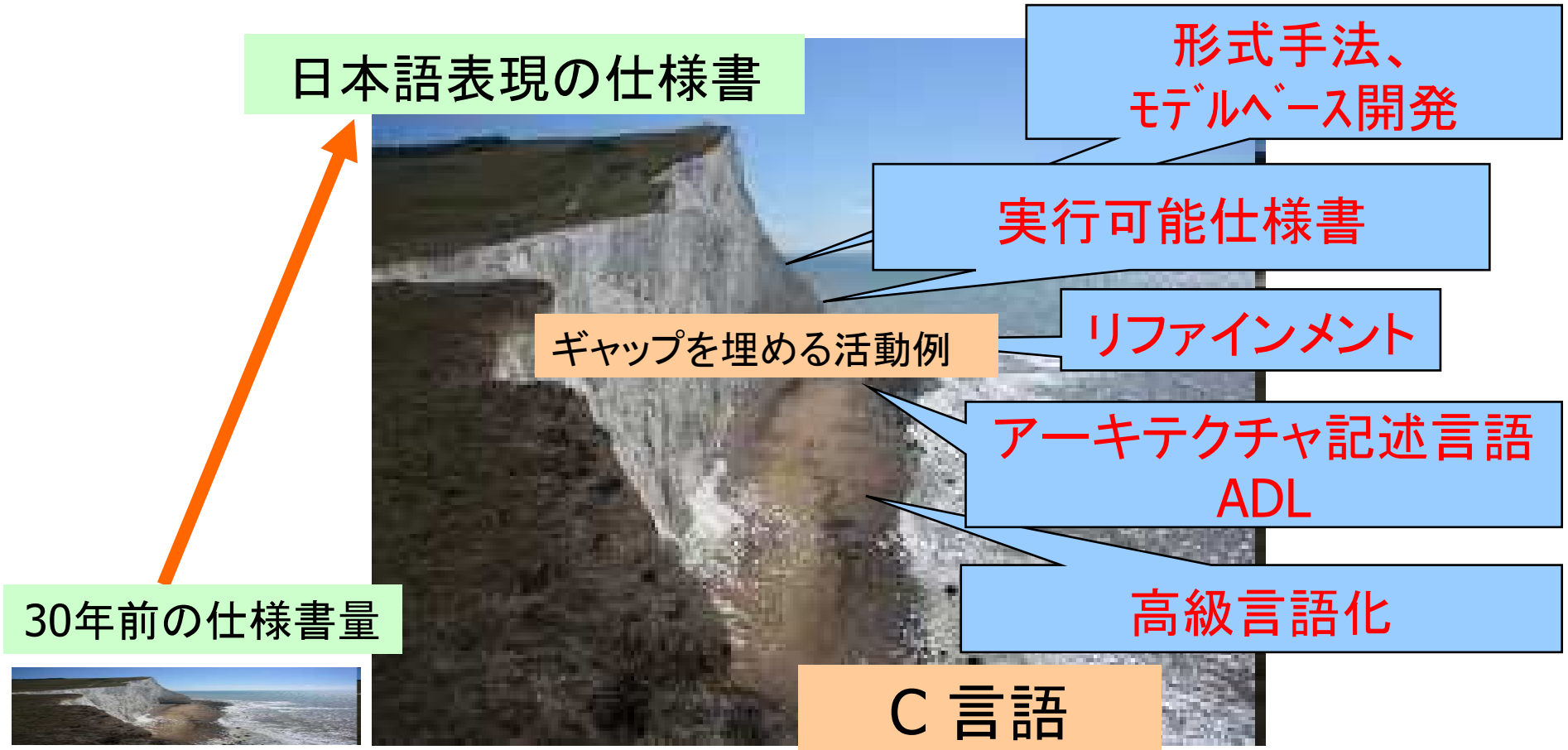
-課題例: 仕様書とプログラミング言語の大きな表現力ギャップが課題になっている-

C言語の長所と欠点。(欠点が多い)

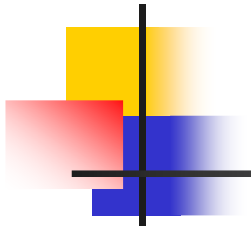
- コントロールフロー表現は得意
- データフロー表現は苦手
- 部品化が苦手
- 階層化が苦手
- アーキテクチャ表現が苦手
- 会話型開発、プロトタイピングが苦手
- 抽象化記述が苦手。
- データの状態属性が苦手(nil概念(Liveness)、mutable、参照透過性)。

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言語は部品化が苦手？



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！！

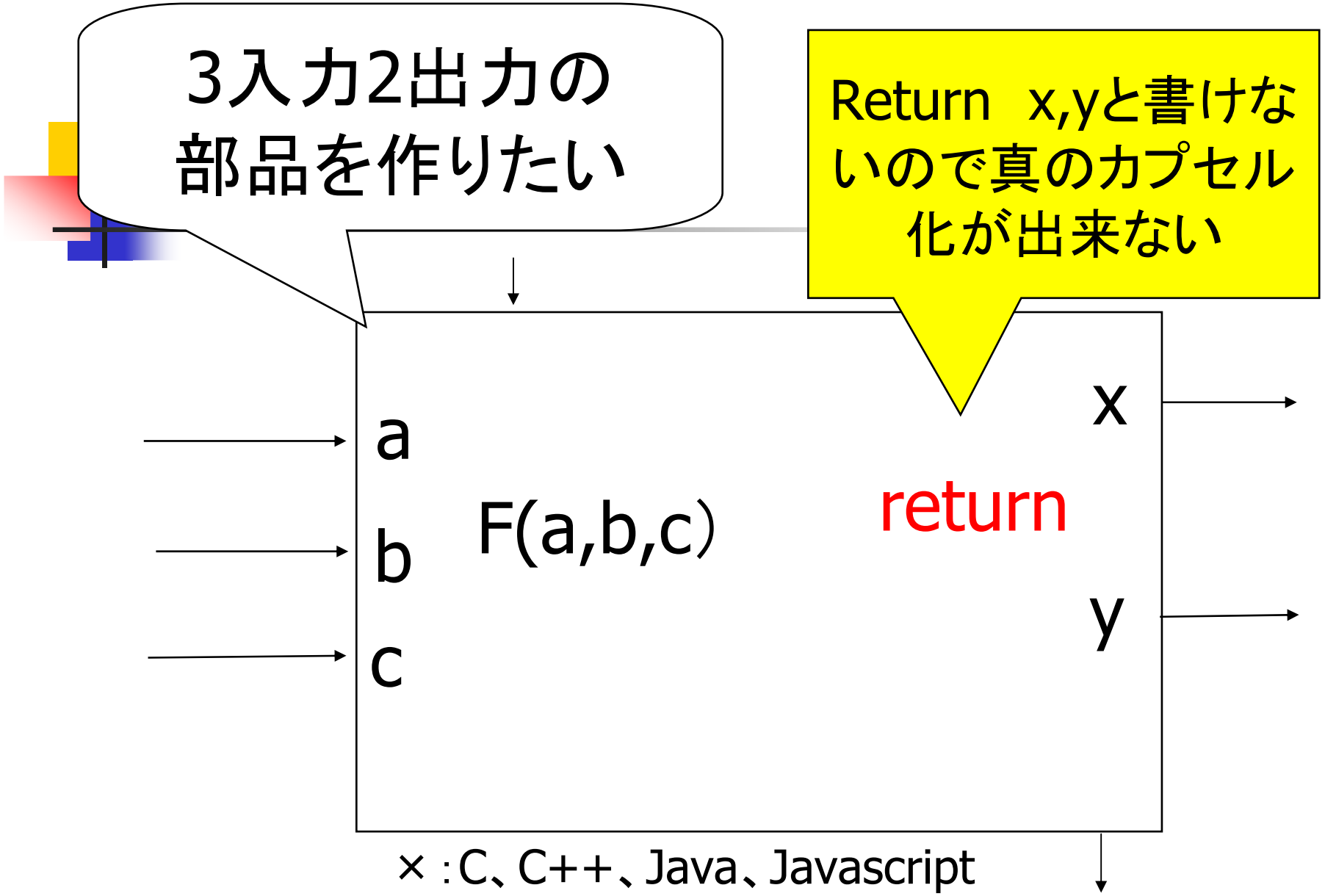
関数は、何の目的で存在するのか？

- 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）

- 関数はライブ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ブラリイー構築のねらいであり、**本来の部品化ライブ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であり、Data Flowを支援する機能が現代はより求められる。
  プログラム
  ＝　アルゴリズム　＋　データ構造
  ＝　Control Flow　＋　Data Flow

- 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）


3入力2出力の
部品を作りたい
Return x,yと書けないので真のカプセル化が出来ない
a
b
c
F(a,b,c)
return
x
y
× : 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？

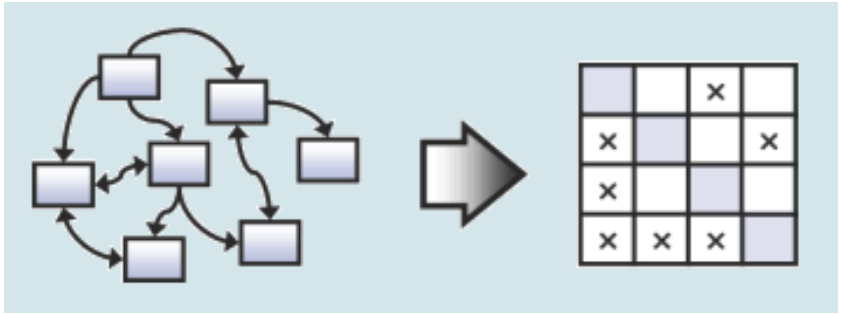
- C言語は機能の低い、危険な言語です。（機能安全の認識）
- スタイルガイドとガイドチエッカーを利用しましょう （機能安全の認識）
- 静的解析ツールを利用しましょう（型整合を含む）
- 設計にはモデリングを併用しましょう。（機能安全の認識）
- 従来のモデリング（例UML）を併用しても、見える化は、十分ではありません。（UMLは2.0以上またはsysMLを利用しましょう）
- ツールで補完しましょう
  - 上記　style_guide、static_analysisを含む
  - 部品化をグローバル変数で支援するツールがあります（Matlab/Simulink,Autosar周辺）
  - 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）

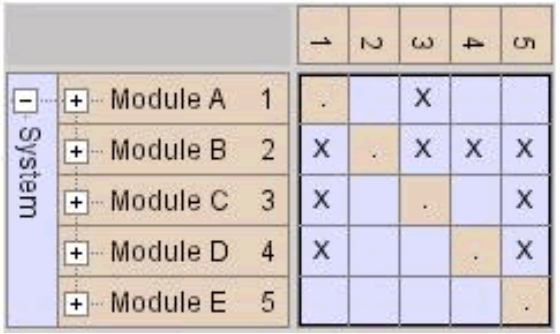
DSM手法とは

- 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。





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

LATTIX

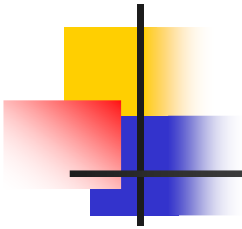
DSM手法,LATTIXツールの良い点：

①アーキテクチャの見える化

②ヴァーチャルリファクタリング

③実情容認と、管理視点分離

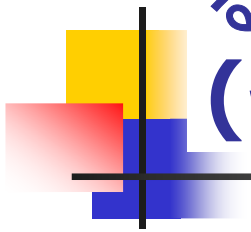
じゃあ、C言語開発の現状を補完するには、明日からまずどうする？(番外)



- 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。
  安全分析をしましょう。

- 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。



③Apple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。
  - Rust, Ruby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

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/

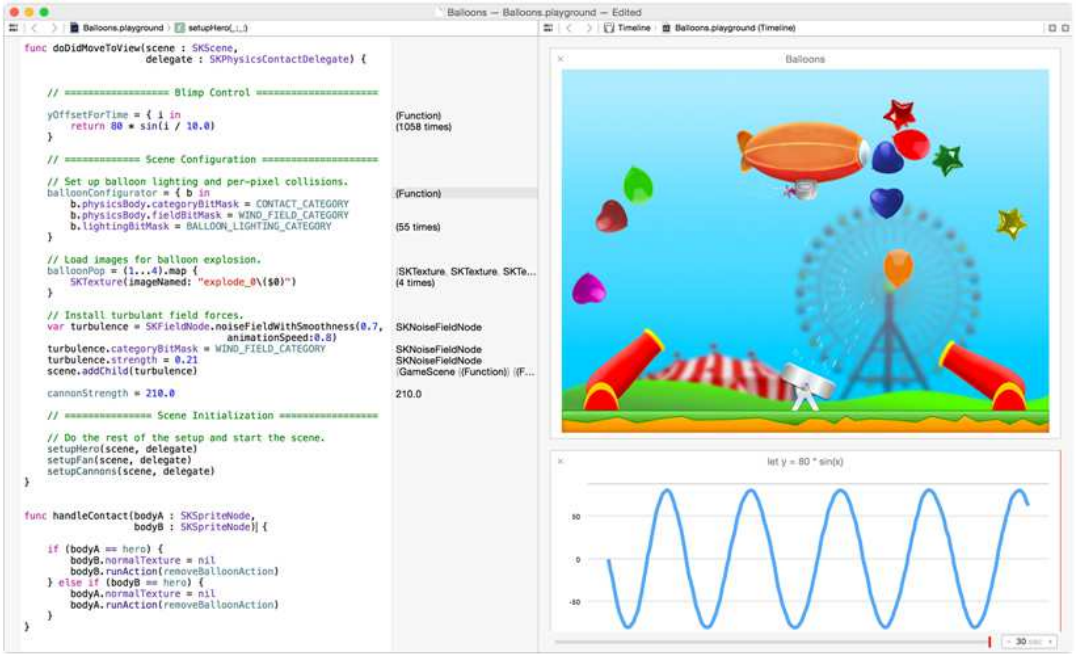
Swift言語
Designed by Chris Lattner

・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++ /11	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

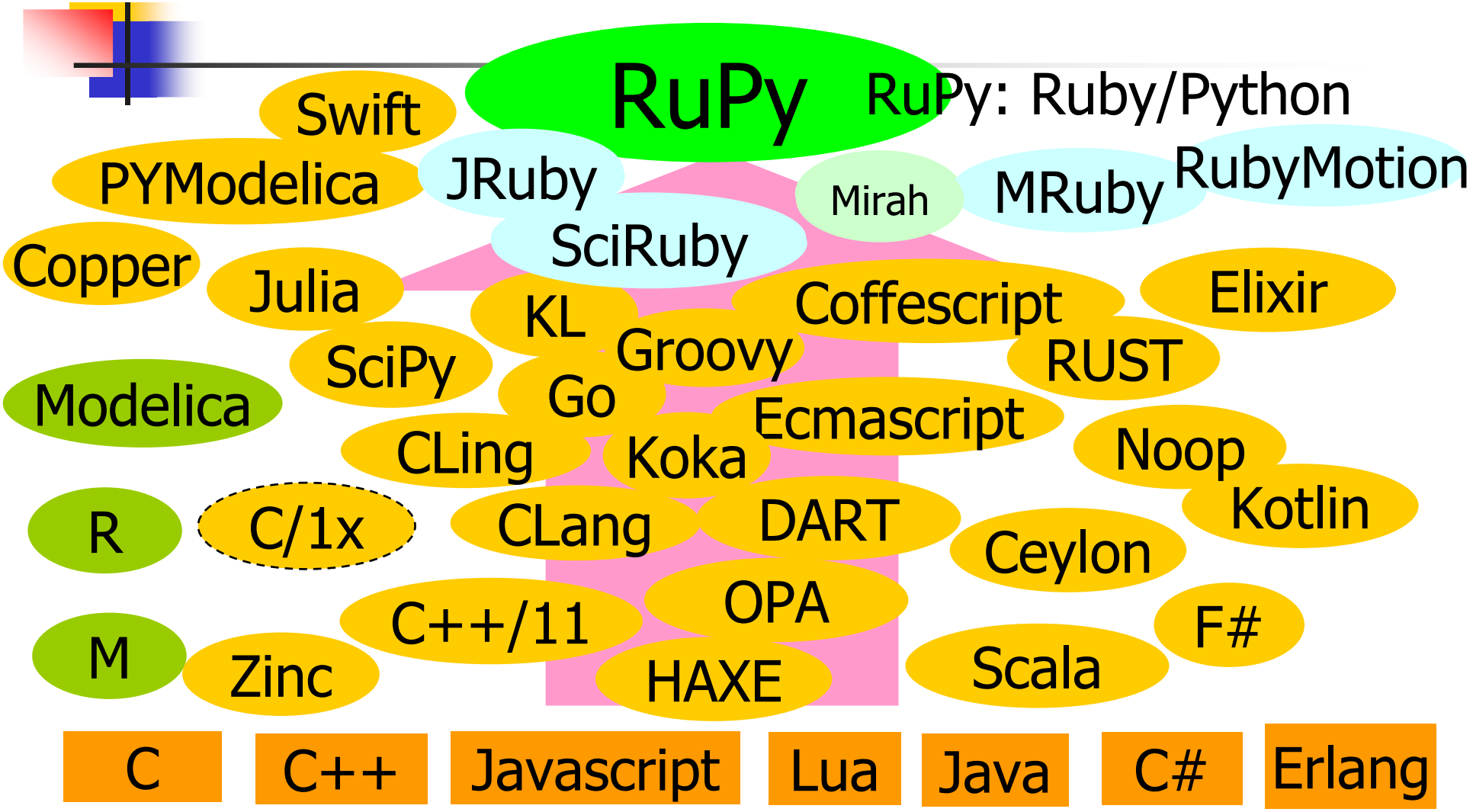
Matlabは実は昔から
・会話型
・プロトタイピング指向だった

	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)



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
- Julia is Ruby like Matlab, R
- C++/11 is RuPy like C++ (More By Boost)
- JVM7.0 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.
- Scientists respect Ruby
- Small Talk &LISP communities respect Ruby

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）
- 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？（番外）

- 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。
  安全分析をしましょう。

- 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。

じゃあ、C言語開発の現状を補完するには、あしたからまずどうする？

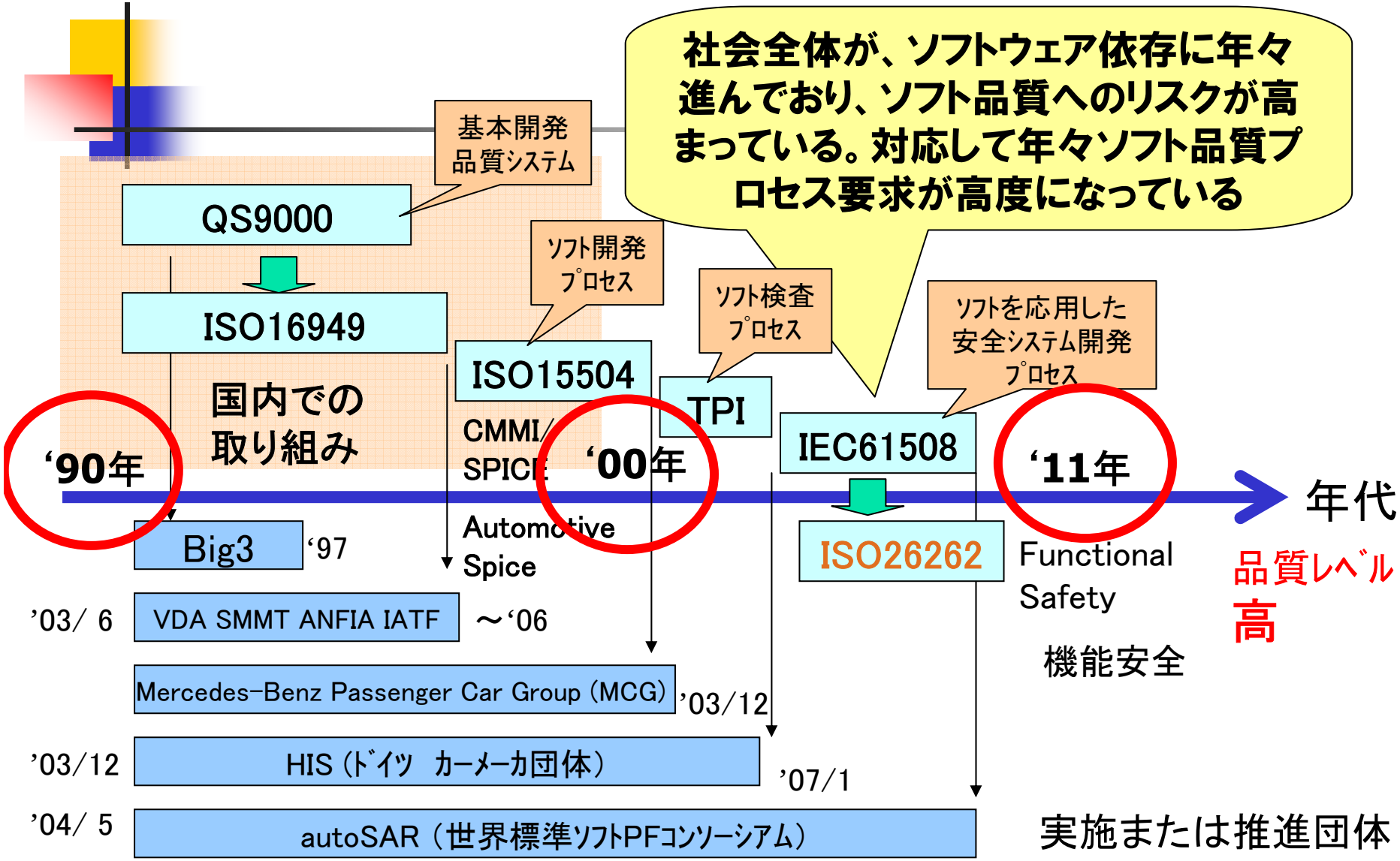
再度掲載

- 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）
- 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。

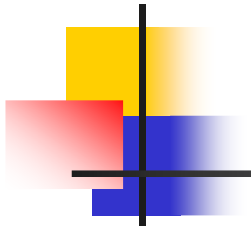
④パラダイムシフトの認識、さらにサイレントパラダイムシフトの認識
（ちょっと外れますが重要です）

- オブジェクト指向、アーキテクチャ指向、MBSE、コミュニテイ指向、アジャイル、リーンプログラミング、

車載組込動向：品質要求基準の変化

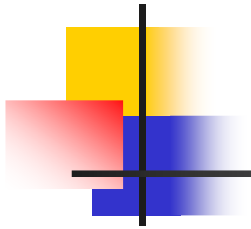


パラダイムシフト



- 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・エンジニアリング

⑤モデルベースMBD、MDDは課題にどのように対応しているのか？

- 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

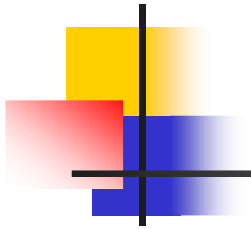
日本流のすり合わせ開発と融和性の高い
開発方法論の国際規格を提案
～ 高信頼なコンシューマデバイスを効率的に開発 ～

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



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）

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）

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）**

- 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

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

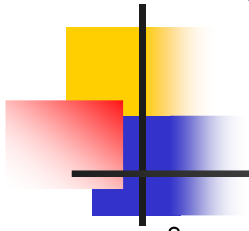
*M. Broy, M. von der Beeck, P. Braun and M. Rappl: *A fundamental critique of the UML for the specification of embedded systems, unpublished, 2000*

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

⑥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

再度掲載

⑤モデルベースMBD、MDDは課題にどのように対応しているのか？

- 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。

再度掲載

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）
- 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

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？(番外)

- 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。

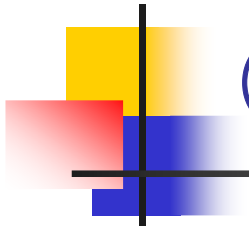
- 状態遷移表で完全であっても、安全分析をすると、システム目線では漏れがあります。
  安全分析をしましょう。

- 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。

再度掲載

じゃあ、C言語開発の現状を補完するには、あしたからまずどうする？

- 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）
- 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開発は、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（プロトタイピング－ リファインメント－ C言語実装）。



⑥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）

- 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言語も適していると感じています。
  (Matlab/simulinkの動いているM言語はRubyそっくりの言語です）

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