OMGにおけるUnified Modeling Language(UML) の位置付け

Yurie Ito
(yurie_ito@omgj.org)
Director of Marketing
OMG Japan

OBJECT MANAGEMENT GROUP

OMGとは

- オブジェクト・マネジメント・グループは
  - 世界最大のソフトウェア標準化コンソーシアム
  - 全世界800の企業、政府機関、大学等が参加
  - 12年間の発展の歴史
  - ユーザーとベンダーが対等な立場で参加
  - 国際的な非営利ソフトウェアコンソーシアム
  - ソフトウェア標準の仕様を開発、管理

2002 (C) OMG Japan

ミッション

- 分散オブジェクト技術を利用し、異なるプラットフォームを含むネットワーク環境において、
  —再利用可能
  —相互運用性があり
  —移植可能な
  分散アプリケーション開発、統合のためのシングルアーキテクチャを開発する。
- 互換性のある各アプリケーション開発のためのフレームワークの普及を促進する。
- 多国籍、多種言語環境における、異種ネットワークシステム上のアプリケーションの連係を可能にする。

2002 (C) OMG Japan

OMGとは

- 1989年4月設立
- 国際ソフトウェア標準化コンソーシアム
- 本部：米国ボストン、ジャパンオフィス：東京赤坂
- インターナショナル・パートナー(英、独、印、中東、ブラジル＋US Government Rep)
- 約800会員：政府機関、大学研究所、企業等
- 日本国内メンバー：日立、富士通、東芝、日本電気、日本ユニシス、NTTはじめ約40社
- 10年間の活動を通し、ネットワーク環境におけるインフラを確立

2002 (C) OMG Japan

業績

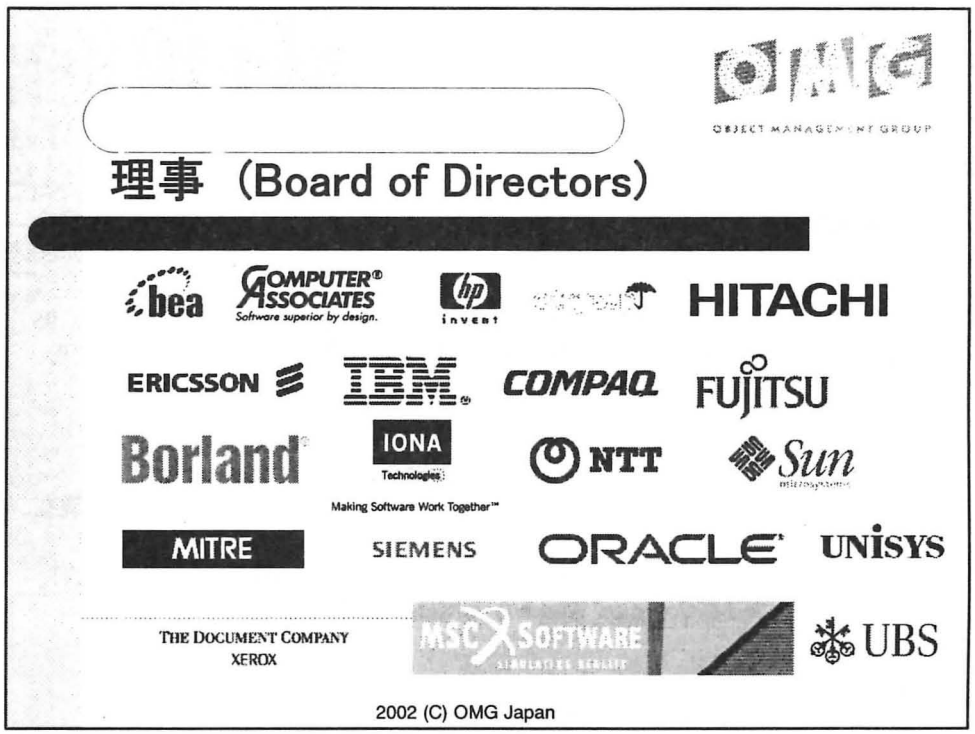
- OMGは下記の業績を含んで成長:
  - 約87の仕様を策定
  - 現在約99の仕様がテクノロジープロセスにおいて進行中
- プラットフォーム標準化活動
  - CORBA、UML、XMI、CWM、MOF、MDA等々
- ドメイン標準化活動
  - 医療、製造、金融、通信、運輸、交通、宇宙、防衛、等々
- ISO PAS (Publicly Available Standard)
  - OMGの標準は国際標準化機構のファストトラックで標準化審議
- 広く受け入れられる標準を18ヶ月以内に作成

2002 (C) OMG Japan

ミーティング/コンファレンス

- 年6回の技術会議(TCミーティング)
  - 欧米、アジア含む各国で
  - 今後予定:
    - 2002年4/22-26ー横浜パシフィコ
    - 2002年6月24-28ー米国フロリダ
    - 2002年9月 ーフィンランド
    - 2002年11月18-22 – 米国ワシントンDC
- 米国でのコンファレンス
  - [2002,3/4-7] OMG Workshop on Transitioning to Web Services
  - [2002,3/18-21] DOCsec 2002 – Security Conference – Baltimore
  - [2002,7/15-19] Realtime & Embedded Workshop - Washington, DC USA

2002 (C) OMG Japan


OBJECT MANAGEMENT GROUP
理事 (Board of Directors)
bea
COMPUTER ASSOCIATES
Software superior by design.
hp invent
HITACHI
ERICSSON
IBM
COMPAQ
FUJITSU
Borland
IONA
Technologies
Making Software Work Together™
NTT
Sun
MITRE
SIEMENS
ORACLE
UNISYS
THE DOCUMENT COMPANY
XEROX
MSC SOFTWARE
UBS
2002 (C) OMG Japan

OBJECT MANAGEMENT GROUP
リエゾン
• ISOのPAS指定団体の認定
—ISO分散コンピューティングフレームワークの仕様化
• IEEE、ECMA、ACM等の標準化グループと密接な協力関係
AIIM International
IEEE
NCPDP
International Middleware Association
IMWA
ECMA
SWIFT
ISO
2002 (C) OMG Japan

OMGプラットフォーム技術群

- ミドルウェア標準
  - 91年CORBA(Common Object Request Broker Architecture)1.0, 95年CORBA2.0、2000年CORBA2.4を標準化、2001年CORBA2.5を標準化
  - 2002年、CORBA3.0をリリース予定
- モデリング標準
  - 97年よりUML(Unified Modeling Language)の標準化を開始。現在、UML1.4、2.0の標準化作業中
  - XMI(XML Metamodel Interchange)、MOF(Meta-Object Facility)、CWM(Common Warehouse Metamodel)

2002 (C) OMG Japan

OMGの目指す方向性

2002 (C) OMG Japan

エンタープライズシステム開発において

- Standish Group の1999年の 調査によると、大規模政府、民間システムは、
  - 16%が納期までに予算内で開発
  - 31%キャンセル
  - 53% の開発が、予算を89%超過
  - 開発完了したケースの内、61%が仕様の要求に見合うもの

2002 (C) OMG Japan

異種環境

- プログラミング言語
  - ～3M COBOLプログラマー
  - ～1.6M VBプログラマー
  - ～1.1M C/C++プログラマー
- オペレーティングシステム
  - Unix、MVS、VMS、MacOS、Windows、PalmOS
  - Windows3.1
  - Embedded devices (mobile, set-top, etc)
- ネットワーク
  - Ethernet, ATM, IP, SS7, Firewire, USB
  - Bluetooth, 802.11b, HomeRF

2002 (C) OMG Japan

OMGの導くミッション

- 今後ハードウェアプラットフォームの一致→見込みなし
- 今後オペレーティングシステムの一致→見込みなし
- 今後プログラミング言語の一致→見込みなし
- 今後ネットワークプロトコルの一致→見込みなし

- インターフェースと相互運用性のコンセンサスを!!!!!!

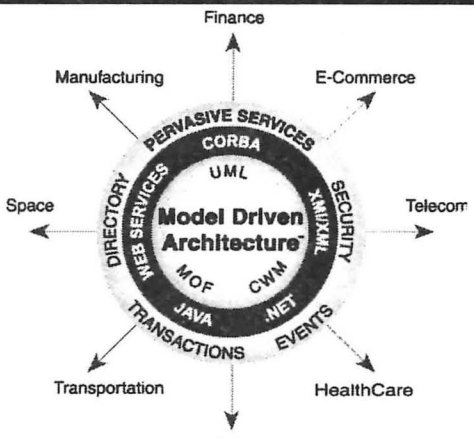
2002 (C) OMG Japan

OMGの目指す方向性:MDAとは

- システムを特定し、構築する新しい方法
- UMLによるモデリングが基盤
- システムのライフサイクルを完全にサポート:分析、設計、実装、稼動、メンテナンス、将来システムとの統合、発展
- 相互運用性、ポータビリティーを確保
- 初期に設定したコストを下げ、ROIを最大化する
- 特定の技術に依存しないモデルの構築
- 様々な産業分野における、標準的なモデルをサポート

2002 (C) OMG Japan

MDA



2002 (C) OMG Japan

Leveraging the UML is critical

- UMLは、90年代前半の、オブジェクト指向分析設計ノーテーションに代わるもの
- 96年から標準化が始まり、97年に採択
- リポジトリ—標準であるMOFと、XMLメタデータ仕様のXMIとともに、MDAの核となる標準
- 市場の成熟
  - 100種類以上ものUMLの書籍
  - 100を越える製品ベースのツール
  - トレーニング市場の充実

2002 (C) OMG Japan

Analysis and Design Platform Task Force:
分析設計タスクフォース動向

UML Textual Notation RFP *(Status: Initial submissions have been received.)*
UML 2.0 Infrastructure RFP *(Status: Initial submissions have been received.)*
UML 2.0 Superstructure RFP *(Status: Initial submissions have been received.)*
UML 2.0 OCL RFP *(Status: Initial submissions have been received.)*
UML 2.0 Diagram Interchange RFP *(Status: Initial submissions have been received.)*
CWM MIP RFP *(Status: Initial submissions have been received.)*
CWM Web Services RFP *(Status: Initial submissions have been received.)*
UML Testing Profile RFP *(Status: Letters of Intent have been received.)*
MOF 2.0 XMI RFP *(Status: RFP has been issued; responses pending.)*
MOF 2.0 IDL RFP *(Status: RFP has been issued; responses pending.)*
MOF 2.0 Core RFP *(Status: RFP has been issued; responses pending.)*

2002 (C) OMG Japan

プロファイルの拡張

- UML for EAI FTF *(Status: No deadlines have passed.)*
- UML Profile for EDOC FTF *(Status: No deadlines have passed.)*
- UML Profile for Scheduling FTF *(Status: No deadlines have passed*
- UML for Systems Engineering RFI *(Status: RFI has been issued; responses pending.)*

2002 (C) OMG Japan

SE/DSIG

- 議長サンフォード・フリーデンタール氏 (Lockheed Martin)
- International Council on Systems Engineering (INCOSE)とのリエゾン
- 2000年9月DTFとして発足
  - 目的:UML for システムエンジニアリングの作成
  - ロードマップ:RFI to survey industry on use of UML for system engineering . Semantic mapping between AP-233 and UML
  - 問題とする領域:
    - 異なる分野のシステム間のコミュニケーションの限界
    - 複雑なシステムの情報を捕らえ分析する限界
    - ツール間の情報の厳密な転送能力の限界
- アナハイムTC議事録
  - http://www.omg.org/cgi-bin/doc?syseng/2002-01-02
- プレゼンテーション
  - ftp://ftp.omg.org/pub/docs/syseng/02-01-03.ppt

2002 (C) OMG Japan

UMLフォーラム Tokyo 2002

- 2002年3月26日—27日 有明TFTホールにて
- コンファレンスホットトピック
- MDA特別対談:「MDAはUMLを次のモデリング言語にするか」
  - OMG会長リチャード・ソーリー、ヤコブソン法創始者イヴァー・ヤコブソン
- MDAテクノロジートラック
- 事例発表1:エンタープライズ系
  - SE動向—ロッキード・マーチン社、OMG/SEDTF議長
  - ウェルス・ファーゴ銀行MDA開発事例
  - テレコム事例、モトローラ社
- 事例発表2:リアルタイム、組込み系
  - 富士ゼロックス、リコー、ソニー

2002 (C) OMG Japan

UMLロボットコンテスト

- 3月26日開催
- 11:00～:
  - コンテスト基調講演 「Executable UML」著者
    レオン・スター氏

競技種目

- スプリント: ハードウェアは、大会標準仕様を使用する。モデルは、UMLクラスモデルまでを公開、状態モデルは競技者にのみに開示。走行前にコードをダウンロードしてからスタートし、トラッキングコース1週を完走する時間を競う。(標準機で1周36秒程度)参加者には、2回の走行機会があり、その時間の短いものが高順位。
- ロングディスタンス: ハードウェア構造は制限を設けず任意。プログラムは公開する。トラッキングコース10週を完走する時間を競う。参加者には、2回の走行機会があり、その時間の短いものが高順位。
- フリースタイル
  構造、意匠、動作のユニークさを一般投票で決定する。

2002 (C) OMG Japan

次回OMGテクニカルミーティング

- OMG Technical Meeting
  - 2002年4月22-26日
  - 横浜、横浜パシフィコ
  - スポンサーDOPG
- レジストレーション
  - http://www.omg.org/registration/registration-tc.htm
- スペシャル・イベント

2002 (C) OMG Japan

参照URL:

- OMG, OMGジャパンとは
  - http://www.omgj.org/about/omg.html
- OMGメンバーになるメリット
  - http://www.omgj.org/about/member/benefits.html
- 会員申込アプリケーション
  - http://www.omg.org/registration/becomeamember.htm
- OMG標準仕様の仕様書
  - http://www.omg.org/technology/documents/formal/
- 新しく標準化された仕様の仕様書
  - http://www.omg.org/technology/documents/recent/
- 現在標準化作業中の仕様とその状況
  - http://www.omg.org/technology/documents/recent/

2002 (C) OMG Japan

参照URL:

- 各種ドメインタスクフォースの一覧
  - http://www.omg.org/homepages/
- CORBA準拠製品リスト
  - http://www.omgj.org/technology/products/corba.html
- CORBA関連製品リスト
  - http://www.omgj.org/technology/products/corba_related.html
- UML準拠製品リスト
  - http://www.omgj.org/technology/products/uml.html
- CORBA事例集
  - http://www.omgj.org/technology/success/

2002 (C) OMG Japan

UMLを用いたシステム開発事例
- TradeOneの場合 -

2002年2月8日
日本フィッツ株式会社
トレードワン事業部
石橋 民男

JFITS

JFITS
日本フィッツ株式会社

日本フィッツ株式会社
会社概要

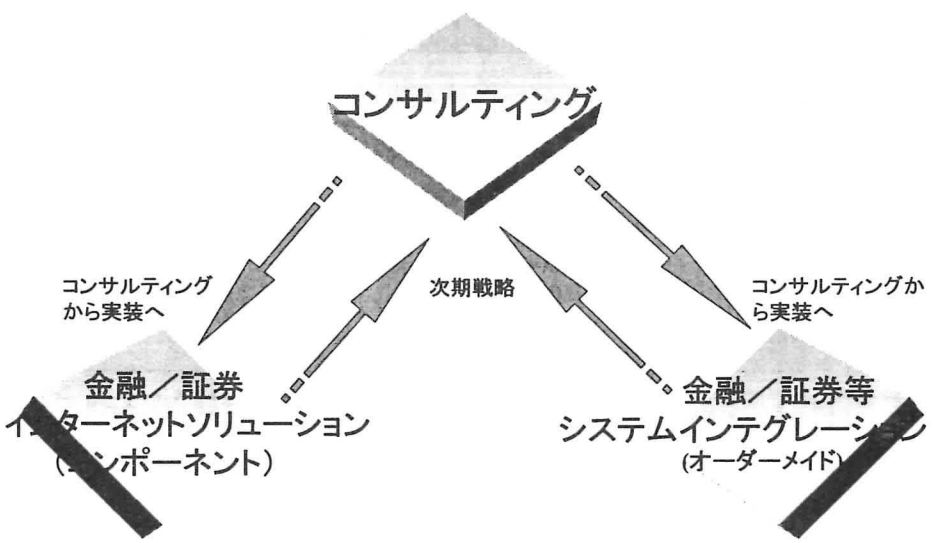
JFITS
日本フィッツ株式会社

【会社プロフィール】

◇名　　称　：日本フィッツ株式会社（略称：ＪＦＩＴＳ）
（ Japan Future Information Technology & Systems Co.,Ltd. ）
◇設　　立　：平成１０年2月2日
◇所在地　：東京都墨田区（錦糸町）
◇資本金　：１０億４千万円　（ＪＡＳＤＡＱ上場）
◇事業内容：ビジネス戦略／ＩＴに関するコンサルティング
システムインテグレーション
◇役　員　：代表取締役社長　村住 直孝（野村證券出身）
取締役副社長　有賀 貞一（野村総研出身）
常務取締役　口村 俊次（山一情報ｼｽﾃﾑ出身）
常務取締役　高原 俊介（山一證券出身）
◇社員数　：約３５４名　（うち技術者数２９１名）
平均年齢32.0歳

JFITS 日本フィッツ株式会社 Page3

【3つの事業セグメント】



JFITS 日本フィッツ株式会社 Page4

TradeOne とは？

JFITS 日本フィジー株式会社

Page5

トレードワン誕生の背景

金融ビッグバンによる新しい証券基幹システムへの需要

- 戦略を即時実行させるスピード
- 一般管理費を押さえるローコスト
- 投資家の信用を守る高い信頼性

JFITS

最先端テクノロジー　最新業務モデル

TradeOne

JFITS 日本フィジー株式会社

Page6

トレードワン コンセプト

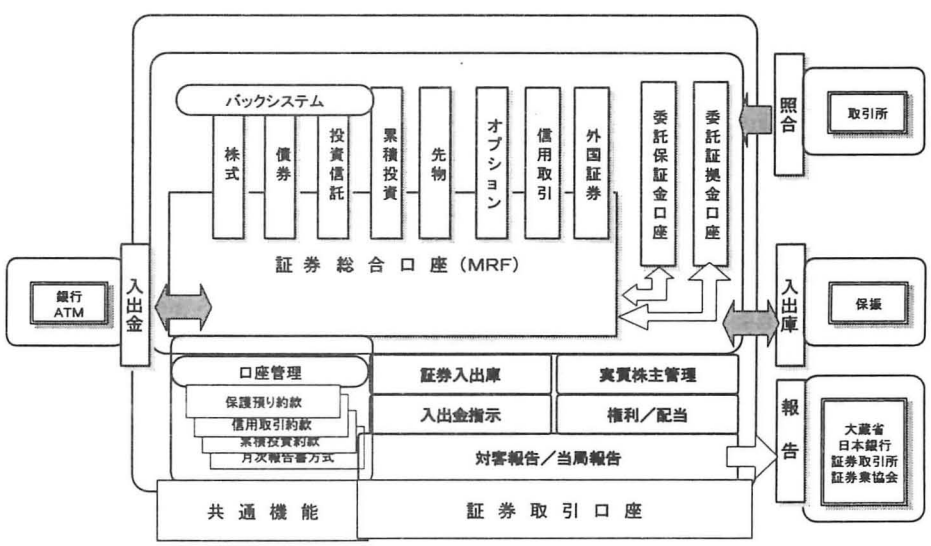
■システムの特長

①拡張機能
　各種商品にそれぞれ対応したコンポーネント群
　　→国内投信(含 累投)、外国投信、国内株式、外国株式等、証券総合口座(MRF)…
②国際基準の取り組み
　　→約定基準・時価主義会計、多通貨会計
　　→決済慣行の変化への適応(当日主義:T+1)
③顧客への利便性向上
　　→顧客報告の迅速化
　　→証券総合口座＝ひとつの口座で全商品を総合的に管理(横断管理)
④事務処理の合理化を推進
　　→現金レス　銀行振込の利用
　　→現物レス　機構振替の利用(振替決済)
　　→ペーパレス　預り証発行をせず、月次報告書方式に限定
　　→営業店事務レス　営業店事務の廃止＝営業店後方事務の本社集中 →店頭精算機能なし
　　→自動化　MRF自動買付(お預り金)、MRF自動売却(顧客の買付代金) 他
⑤拡張性・柔軟性・信頼性の高いシステムデザイン
⑥トータルなサポート
　　→証券業務フロー、事務マニュアル、帳票等の整備・提供

JFITS 日本フィッツ株式会社

Page7

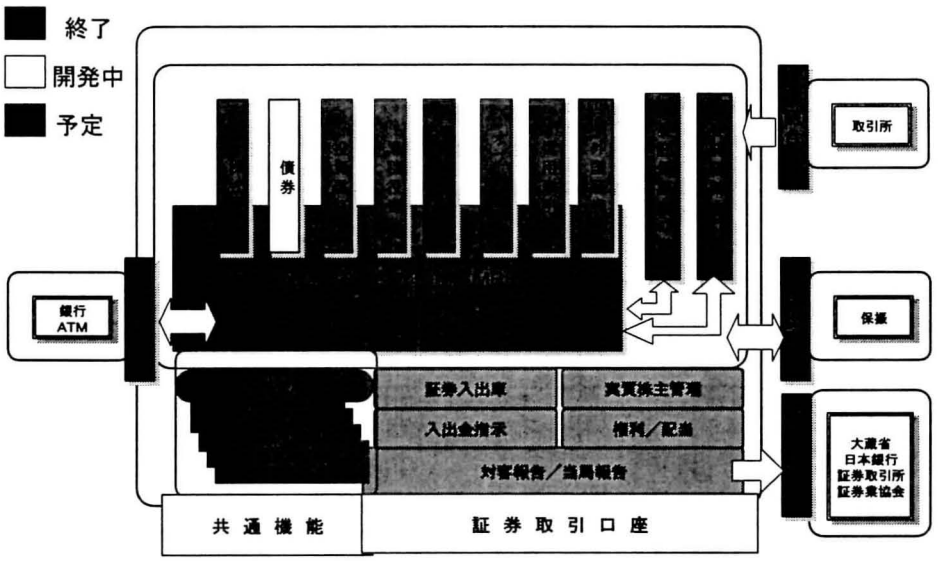
トレードワン 機能構成



JFITS 日本フィッツ株式会社

Page8

トレードワン 開発状況



JFITS 日本フィジック株式会社

Page9

システム規模(開発終了分)

- 設計/開発手法　RUP
- 設計表記法　UML
- 開発言語　C++、JAVA、等
- ソースコードライン数(コメント行除く)
  - JSP　100Kstep
  - JAVA　250Kstep
  - C++　1,500Kstep
  - ---
  - 合計　1,850Kstep
- 開発期間　1999年10月～2001年12月 (26ヶ月)

- 現在、2社目導入カスタマイズを実施中

JFITS 日本フィジック株式会社

Page10

プロジェクト・イニシエーション

JFITS
日本ファンツ株式会社

事前の考慮点

- 開発工程
  - 開発プロセスの定義
  - 成果物の表記
  - 上記を補助するツール
- 要員教育
  - 各種アーキテクチャに精通している各協力会社と共同作業によるノウハウの習得
  - 各種標準化文書の作成とこれに関するセミナー開催
- 品質向上対策
  - プロトタイプ作成
  - アーキテクチャ部分を隠蔽したコンポーネントの作成
  - 各種標準化ドキュメントの作成

JFITS
日本ファンツ株式会社

Page12

開発工程に対する考慮点

- 各開発工程のプロセスの定義が必要
- 各種成果物の表記法の標準化
- 上記プロセスや成果物作成の補助ツールの必要性

- ✓ 開発プロセス全般としてRUP（Rational Unified Process）
- ✓ 表記法としてUML（Unified Modeling Language）
- ✓ RUPでの手法およびUMLを記述するのを補助するツールとして事実上の標準であるRose（Rational Rose）

JFITS 日本フィッツ株式会社

Page13

- **RUP**
  - 基本的な概念はRUPをベースとするが、詳細部分などに関しては独自にカスタマイズを行うこととした
- **UML**
  - 主に静的な構造はクラス図、振る舞いに関してはシーケンス図を利用することとした
  - クラスなどの詳細な記述に関しては日本語でRose上に記述を行うこととした
  - バッチ処理に必要なドキュメントなどUMLで表現しづらい部分などは独自フォーマットで成果物を作成することとした
- **Rose**
  - UML表記を行う成果物は基本Rose上に定義し、コードのスケルトン生成を自動的に行うこととした（ビジネスロジック部分のみ）
  - Rational SoDAを利用してWord文書変換などを行うこととした

JFITS 日本フィッツ株式会社

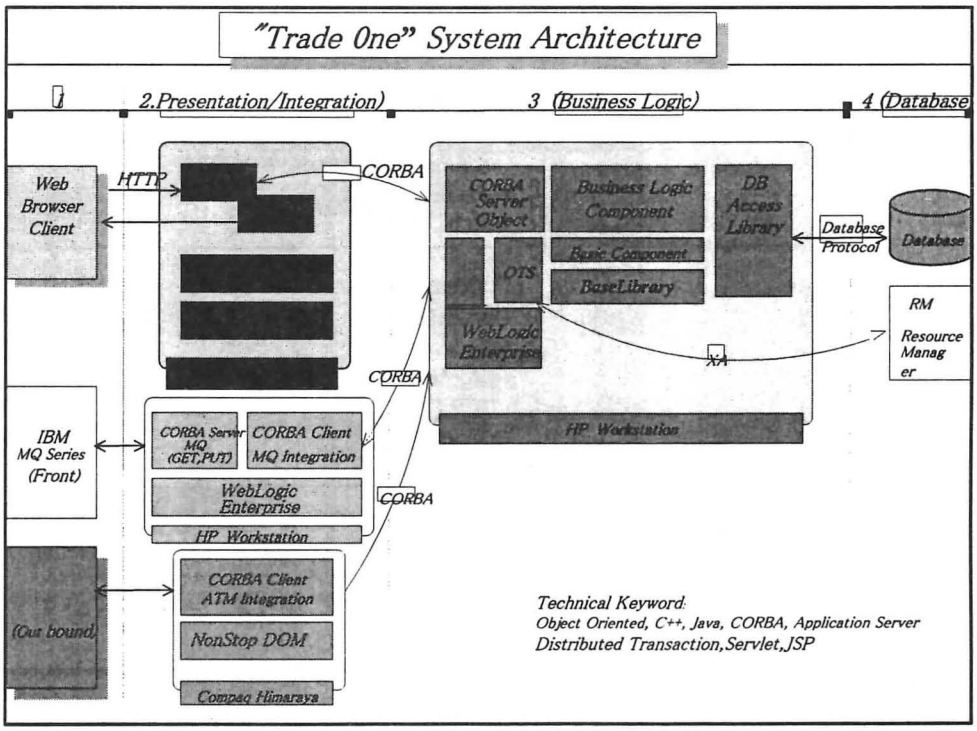
Page14

アーキテクチャの選定方針

- Webアーキテクチャ
  - アプリケーションの配布・端末設定などの運用コストを低減可能なアプリケーション稼働環境であるため
- 3層モデル
  - プレゼンテーション・ビジネスロジック・データアクセスの3層に分離することにより、画面・機能・データの変更に対して独立性を高め、仕様変更等の保守性に優れたモデルとするため
- オープンシステム
  - 証券ビジネスの今後のニーズに柔軟かつスピーディに対応するため、OS・RDB・通信インフラなどの分野でシステムの陳腐化の防止およびDWH等のCRMシステムなどの構築が容易な技術要素（UNIX、ORACLE、CORBA）を採用した
- 高可用性
  - バックオフィス業務の重要性を重視し、H/W・S/W障害時の業務の中断を最小限とする為、従来の汎用機システムと同等の高可用性（スタンバイ）システムを採用した

JFITS 日本ビジネス株式会社

Page15

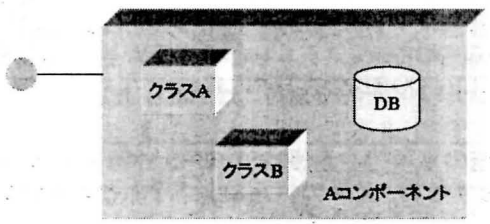


アプリケーション構造コンセプト

- ◆ 商品や機能の変更に対し容易にカスタマイズができるような仕組みが必要
- ◆ 機能に対し独自戦略的な機能を容易に組み込める仕組みが必要

⇩

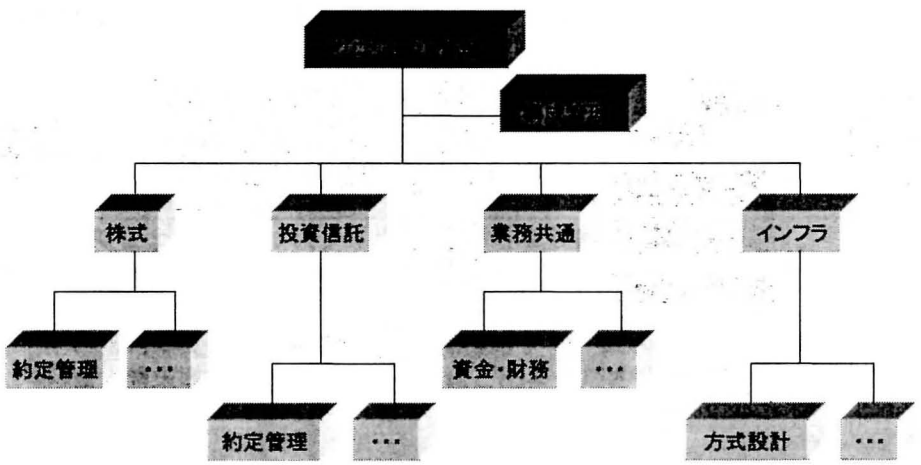
- ✓ 「業務上の意味ある塊」に対しコンポーネントという概念の導入
- ✓ コンポーネント間の依存関係を疎に保つためインターフェースと実装の分離
- ✓ コンポーネント内の機能のカスタマイズ容易性



JFITS 日本ファンズ株式会社

Page17

開発体制

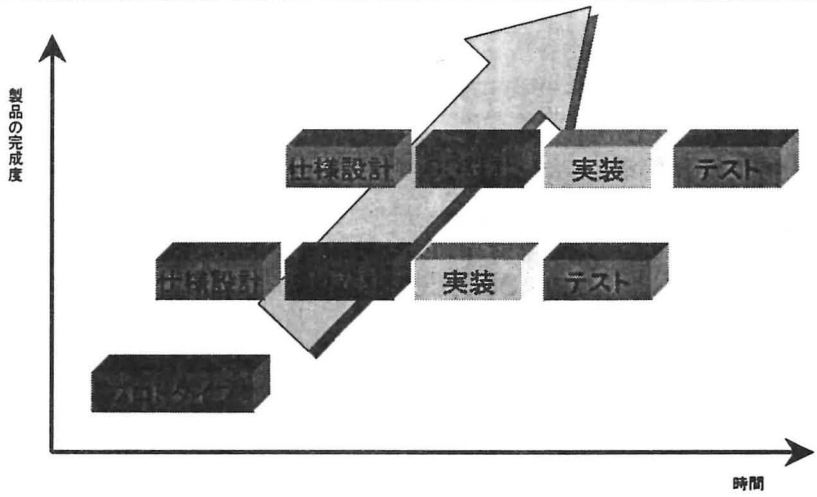


※各業務チーム内に業務設計者とOO設計、開発者が混在
※システムアーキテクチャに関わることはすべてインフラチームに集約

JFITS 日本ファンズ株式会社

Page18

開発工程

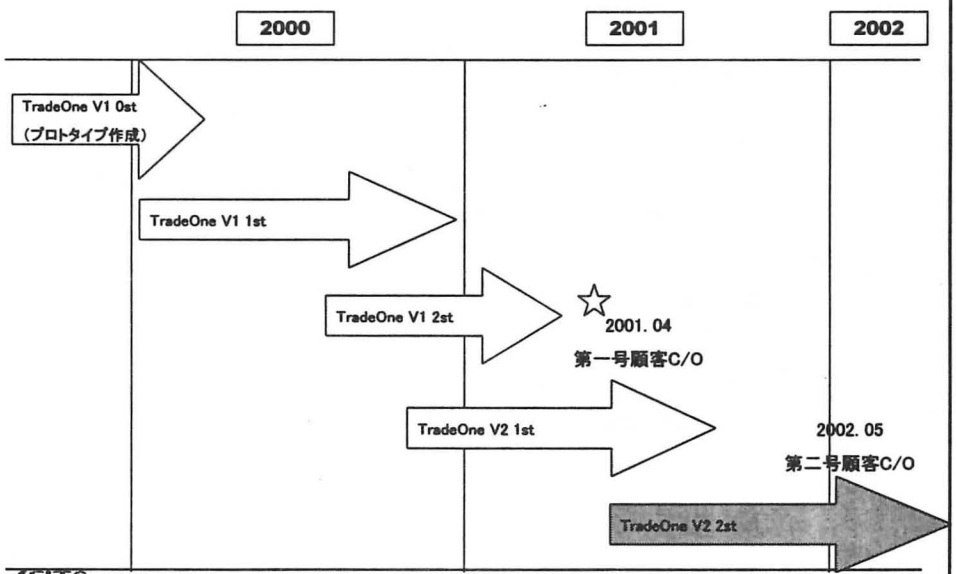


※見積りがしづらい等の管理のしづらさや等の理由により、無理に繰り返し開発をする手法ではなく、ウォーターフォール的なプロセスで進行することとした

JFITS
日本ファジィ株式会社

Page19

開発スケジュール



JFITS
日本ファジィ株式会社

Page20

開発プロジェクトの経過

JFITS
日本フィッツ株式会社

開発プロジェクトの経過

- 成功した点
  - コンポーネントという単位で分離したため、設計・開発などを並列に進行することができた
  - コンポーネント内部での不具合等に関しても最小限の影響で進行することができた
  - アーキテクチャ部分も隠蔽しているため、アーキテクチャ部分での修正個所を最小限にすることができた
- 問題点
  - 仕様記述が不十分
  - 各種標準化決定の遅延
  - 品質確保のための工数の増大
  - 模倣されたコードの増大

JFITS
日本フィッツ株式会社
Page22

問題点(その1)

仕様記述が不十分

- 日本語での業務仕様記述によるあいまいさのため、業務設計者からオブジェクト指向設計者への情報伝達不足
  - 手続き的な記述しているため「意味」がオブジェクト指向設計者へ完全には伝わらなかった。
- 業務コンポーネントのインターフェース間でやり取りされるデータの詳細な利用方法に関する記述不足

⇩

- ✓ 仕様書の記述を形式的記述を模倣した弊社独自の記述方式を策定し、仕様書の記述を行った
- ✓ シナリオにそって机上で各々のコンポーネントの役割などについて確認会を行った

JFITS 日本ファジー株式会社 Page23

問題点(その2)

各種標準化決定の遅延

- プロトタイプで発見できなかった問題の発覚
- アーキテクチャ要素に関する決定の遅延
- 技術要素を隠蔽した共通コンポーネントの利用者と提供者側での差異が発生
- 各コンポーネントでほぼ同等の機能を持つが、微妙に異なる役割のクラスが散在
  - 各設計者間でのオブジェクト指向に対しての考え方が微妙に異なっていた

⇩

- ✓ 共通コンポーネントに関するドキュメントの整備
- ✓ 散在していたサンプルコードの整備
- ✓ ある程度標準的に利用されるクラスに関しては、クラスの役割を明確化し標準化

JFITS 日本ファジー株式会社 Page24

問題点(その3)

品質確保のための工数の増大

- ◆ 再テストを行う場面が頻繁にあり、その工数が増大
  - – 仕様修正・追加
  - – 設計ミス
  - – プログラミングバグ

- ✓ 弊社で独自のテストフレームワークを開発(以下各テストケースごとに設定可能)
  - ■ 事前条件としてデータベース上へレコード挿入・更新・削除など
  - ■ 対象プログラム実行(プログラムへの引数などを設定)
  - ■ 事後条件としてプログラムの状態確認およびデータベースのレコード状態確認などをCSVへ書き出し

JFITS 日本ファイン株式会社

Page25

問題点(その4)

模倣されたコードの増大

- ◆ ユーザーインターフェースの画面制御や入力されたデータのチェック(桁数や型など)などはほぼ同一のコードとなっている
- ◆ 基本的な処理はスーパークラスで定義されているため、開発しなければいけないコードは限定されていた

- ✓ ユーザーインターフェースでの可変部分のパターン化
  - ■ 画面遷移
  - ■ 入力データのチェックロジック
  - ■ etc
- ✓ ユーザーインターフェース(JSP、Servlet)のコード自動生成ツールの開発
  - ■ 上記のパターン化されたものを組み合わせた定義ファイルより、ユーザーインターフェース部分のコード(HTML、JSP、Servlet)を生成する

JFITS 日本ファイン株式会社

Page26

UMLの適用

JFITS
日本ライン株式会社

UMLの適用

- 経緯
  - プロジェクト発足
    - オブジェクト指向による構築が決定
  - 設計技法、ツールの決定
    - UML手法の全面適用決定
    - ROSEの全面適用決定
  - プロトタイプによる検証
    - 部分的な機能を設計～実装
  - 評価
    - UML手法の部分的適用決定
    - 開発プロセスの確定
  - 全体開発着手
  - 完成
  - 現在も引き続き、同様の手法にて開発を継続

JFITS
日本ライン株式会社
Page28

UMLの適用

- プロトタイプによる検証
  - 以下の観点でプロジェクトの適用可能性を検証した。
    - 設計の有効度合
    - 実装～テストの実現可能性
    - トレーニング
    - 要員の調達可能性
    - 生産性
  - プロトタイプの体制
    - 現存する要員で模索するのでなく、外部から優秀な支援者を受け入れて、効率的に実施。

JFITS 日本ファジィ株式会社 Page29

UMLの適用

- 適用範囲の決定
  - 評価
    - 設計の有効度合、実装～テストの実現
      - 有効であり実現可能であることが検証できた。
    - トレーニング
      - 設計可能なレベルに達するには、十分な期間が必要。
      - 仕様確認レベルへの到達は短期間で可能
    - 要員の調達可能性
      - 大規模な開発であり、かつ短期間の開発であるため、UMLの設計を全面的に適用した場合の要員調達は不可（既存要員の教育を含めて）。
    - 生産性
      - UMLの適用が直接原因ではないが、オブジェクト指向を採用したことにより設計工程の生産性は落ちた（業務知識保有者とオブジェクト指向設計者が別の要員が担当）。

JFITS 日本ファジィ株式会社 Page30

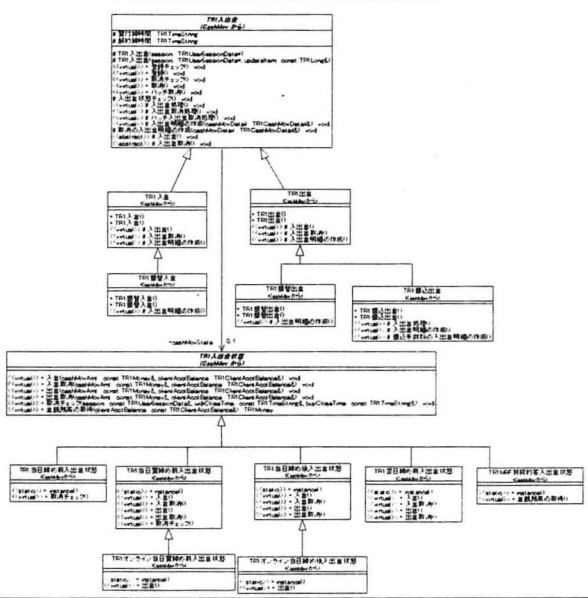
UMLの適用

- 適用範囲の決定
  - 適用範囲
    - 開発期間、要員の調達可能性、既存要員のトレーニングの期間等を考慮し、全面手法適用を見送り以下の成果物のみを対象とした。いづれもROSEにて定義し、仕様書の出力はSODAによって行った。
      - ユースケース図
      - クラス図
      - シーケンス図
    - その他のUML成果物の規定事項については、UMLの体系をカスタマイズし、独自仕様書にて補完を行うこととした。

JFITS 日本ファジー株式会社

Page31

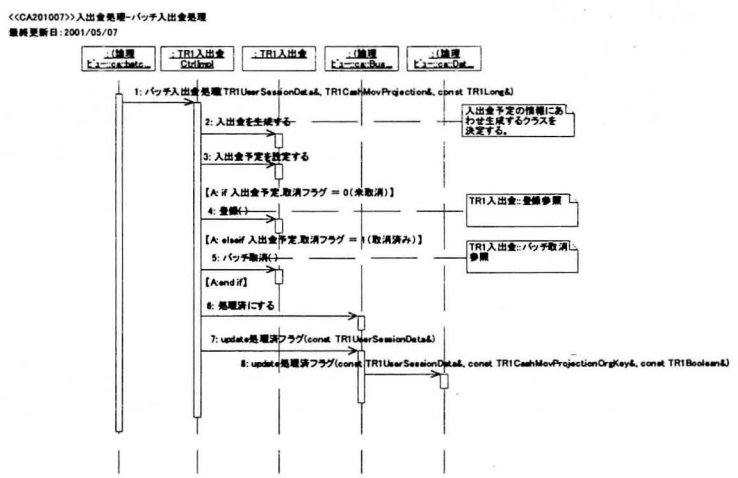
【参考】クラス図



JFITS 日本ファジー株式会社

Page32

【参考】シーケンス図



JFITS 日本フィッツ株式会社

Page33

UMLの適用

- 適用後の結果
  - いくつかの問題が発生したものの、結果的にはプロジェクトの成功に大きく寄与できたと考える。
    - 良かった点
      - 仕様記述が統一でき、サブシステム間の不整合等による手戻りが抑えれられ、実装移行のフェーズが非常に効率的に実施できた。
    - 発生した課題／問題
      - UML技術者の育成
        - 証券システムの構築経験者は、多くは汎用機の経験者であり、設計を実施するレベルまでに達することは困難。仕様確認レベルまでの育成にとどまった。
      - UMLを含めたトータル開発プロセスの決定遅延
        - 大規模プロジェクトが故にルールの早期確定と伝達が重要

JFITS 日本フィッツ株式会社

Page34

UMLの適用

- おわりに
  - プロジェクト適用時のポイント
    - ―プロトタイプによる部分適用による事前検証
      - ―問題点の早期洗出しと、全体適用の意思決定ポイントの評価実施
    - ―要員の確保と既存要員の育成
      - ―早期に計画化し、要員確保を計画的に実現
    - ―プロジェクトの状況にあった開発プロセスの定義
      - ―全てを適用することは考えず、実態に合わせてプロセスをカスタマイズ(プロセスの明確な定義は必須)
        - ―小規模開発には、全面的な適用が効果的
        - ―大規模開発時は、現実的な適用が効果的

―以上―

JFITS 日本フィッツ株式会社

Page35

Trade One

Best solution for the best practice …

presented by JFITS 日本フィッツ株式会社

JFITS 日本フィッツ株式会社

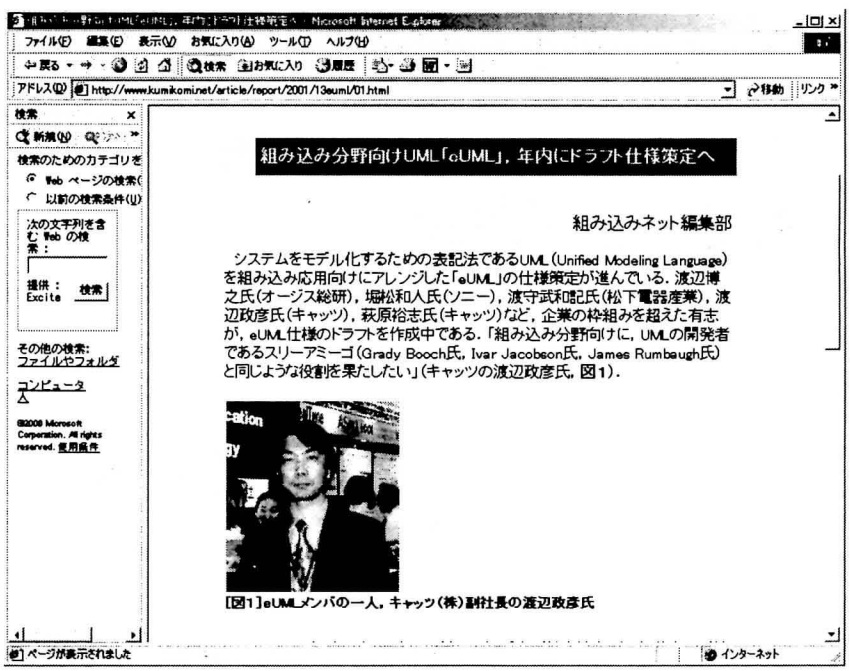
SEA 特別ワークショップ

組込み用にアレンジしたeUMLの狙い

2002/2/8
CATS 渡辺政彦

Communication And Technology Systems

©CATS



組み込み分野向けUML「eUML」，年内にドラフト仕様策定へ

組み込みネット編集部

システムをモデル化するための表記法であるUML（Unified Modeling Language）を組み込み応用向けにアレンジした「eUML」の仕様策定が進んでいる．渡辺博之氏（オージス総研），堀松和人氏（ソニー），渡守武和記氏（松下電器産業），渡辺政彦氏（キャッツ），萩原裕志氏（キャッツ）など，企業の枠組みを超えた有志が，eUML仕様のドラフトを作成中である．「組み込み分野向けに，UMLの開発者であるスリーアミーゴ（Grady Booch氏，Ivar Jacobson氏，James Rumbaugh氏）と同じような役割を果たしたい」（キャッツの渡辺政彦氏，図1）．

[図1]eUMLメンバの一人，キャッツ（株）副社長の渡辺政彦氏

©CATS

eUML活動概要

2001年6月26日キックオフ

member

渡辺@オージス
堀松@ソニー
渡守武（ともたけ）@松下電器
渡辺@キャッツ

schedule

2002年GW頃に書籍化してオープン化

©CATS

*** ワークショップの狙い ***
UMLが何者で，どこから来て，どこへ行こうとしているかを
下記の観点から理解できる.

項目	eUML
コンピュータサイエンス：プログラミング言語	アクションシンタックス
モデル	システムモデル
フォーマルメソッド	詳細化証明
ソフトウェアエンジニアリング：モデル	問題発見
開発方法論	ガイドライン
ツール	検証
プロセス	コミュニケーションツール

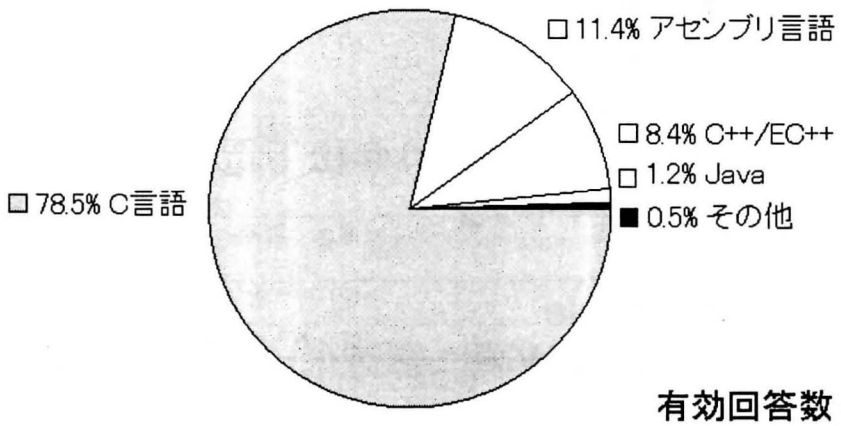
©CATS

コンピュータサイエンス: プログラミング言語

使用したプログラム言語

調査活動

2000年度のアンケート調査結果（調査期間: 2000年11月～2001年1月）

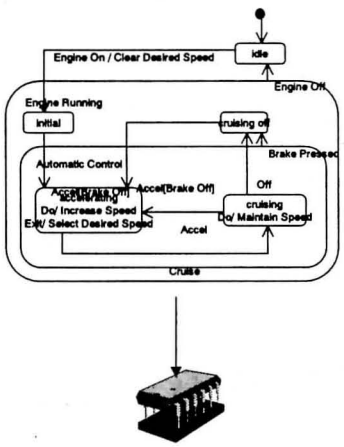


有効回答数 428

©CATS

http://www.itron.gr.jp/survey2001/graph-j/13.gif

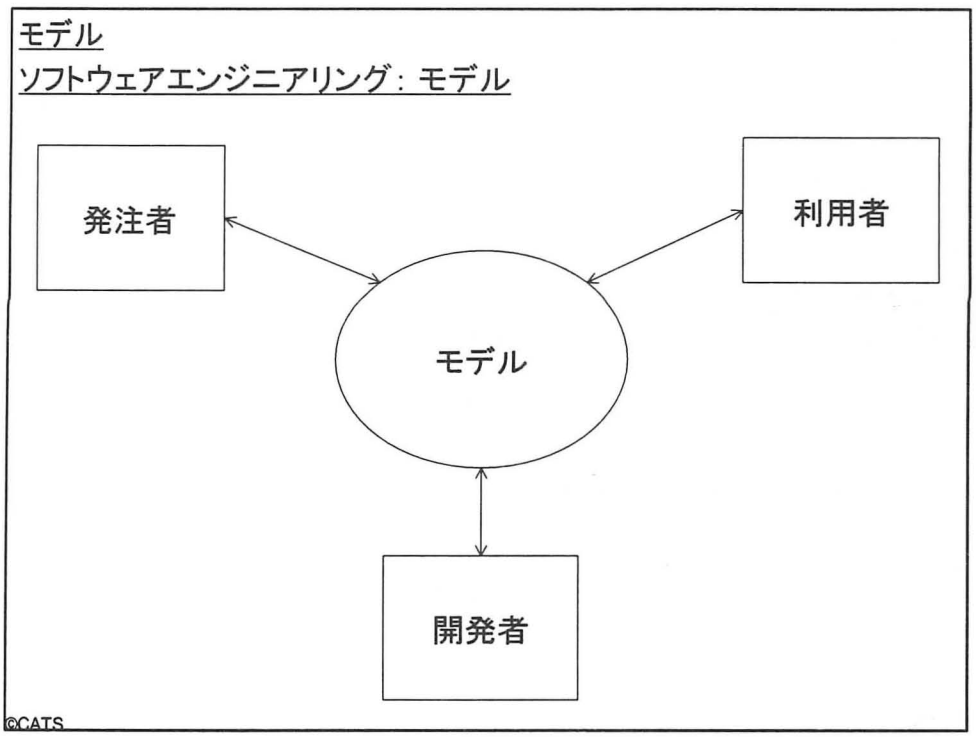
1	System	50,000	特定の問題解決に関連するプログラムの集まり
2	Program	10,000	コンピュータ上で機能を独立して実行可能な単位
3	Component	2～3,000	特定データに関連するセクションの集まり
4	Section	600	コンパイル単位、単体テスト単位
5	Unit	125	机上レビューの単位
6	Segment	25	最小単位モジュール
7	Basic Segment	5	最小処理単位PAD等の1ボックス
8	Source Code	1	ソースコードの1ステップ

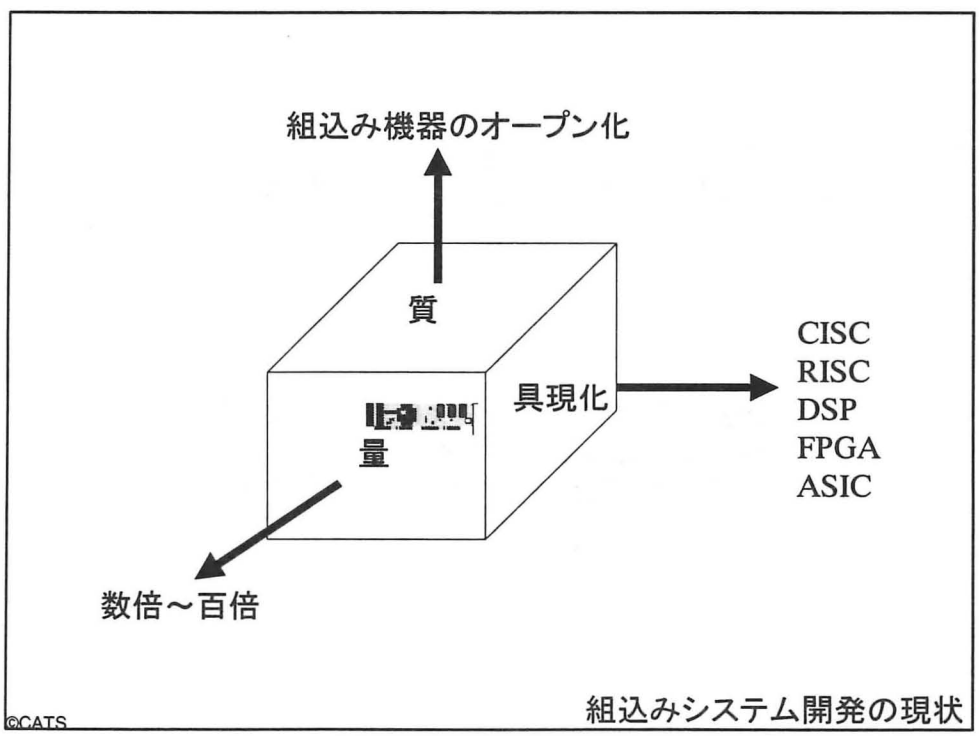


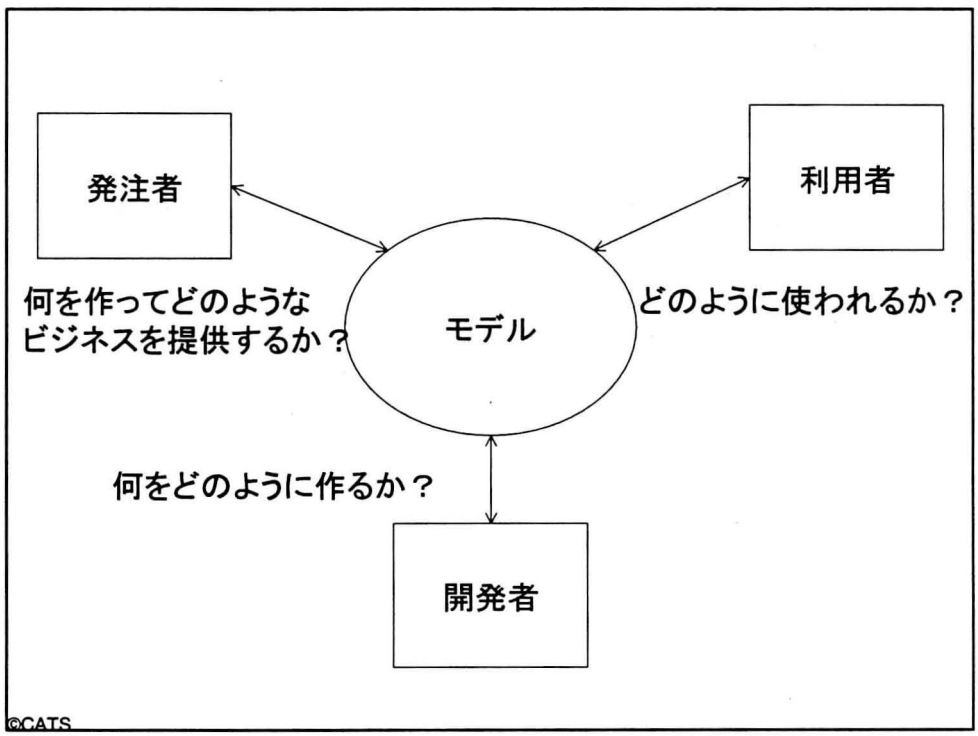
	UML	C
シンタックス	規定なし	ANSI-C
セマンティクス	アクションセマンティクス	ANSI-C
処理系	規定なし	コンパイラ

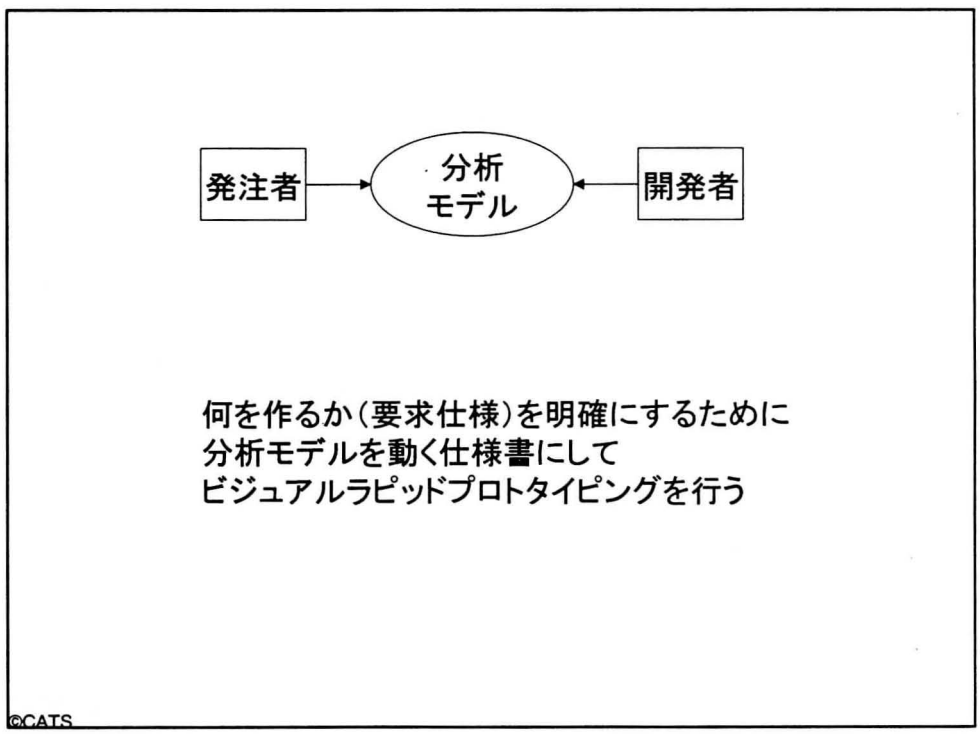
Klatzky,R.L:箱田裕司訳: 記憶のしくみ: サイエンス社: 1982
Dijkstra,E.W,野中浩平訳: 構造化プログラミング論: サイエンス社: 1975
菅野文友: ソフトウェアの品質管理: 日科技連: ISBN4-8171-6101-9

©CATS


モデル
ソフトウェアエンジニアリング：モデル
発注者
利用者
モデル
開発者
©CATS

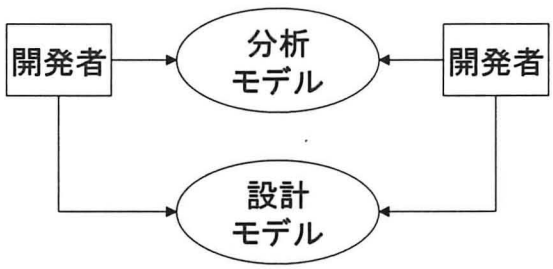

組込み機器のオープン化
質
具現化
量
CISC
RISC
DSP
FPGA
ASIC
数倍～百倍
©CATS
組込みシステム開発の現状


発注者
利用者
何を作ってどのような
ビジネスを提供するか？
モデル
どのように使われるか？
何をどのように作るか？
開発者
©CATS


発注者
分析
モデル
開発者
何を作るか（要求仕様）を明確にするために
分析モデルを動く仕様書にして
ビジュアルラピッドプロトタイピングを行う
©CATS

利用者がシステムに何を求めるか
どのような操作が行われるか
制御対象がどのような挙動をするかを
を明確にするために
利用モデルを動く仕様書にして
ビジュアルラピッドプロトタイピングを行う

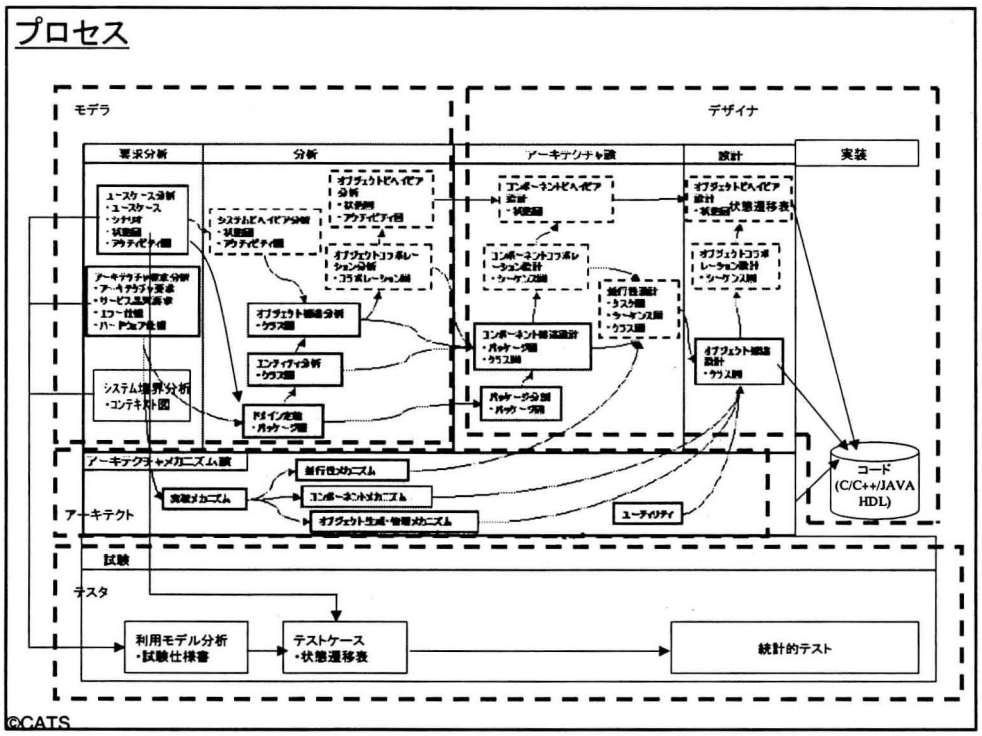
©CATS

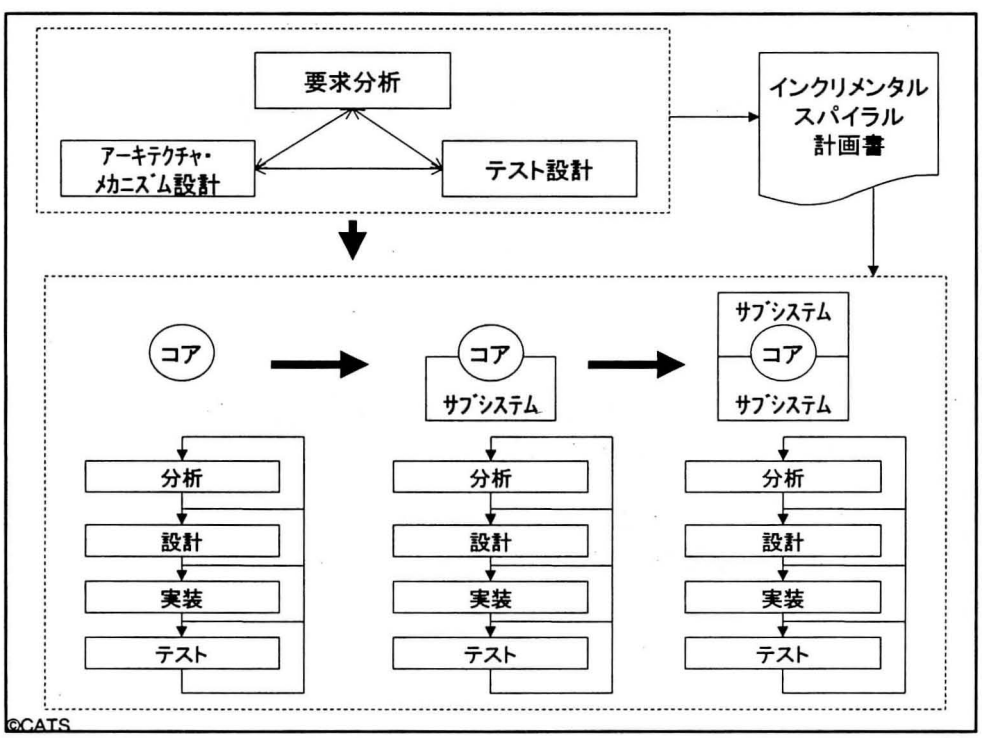


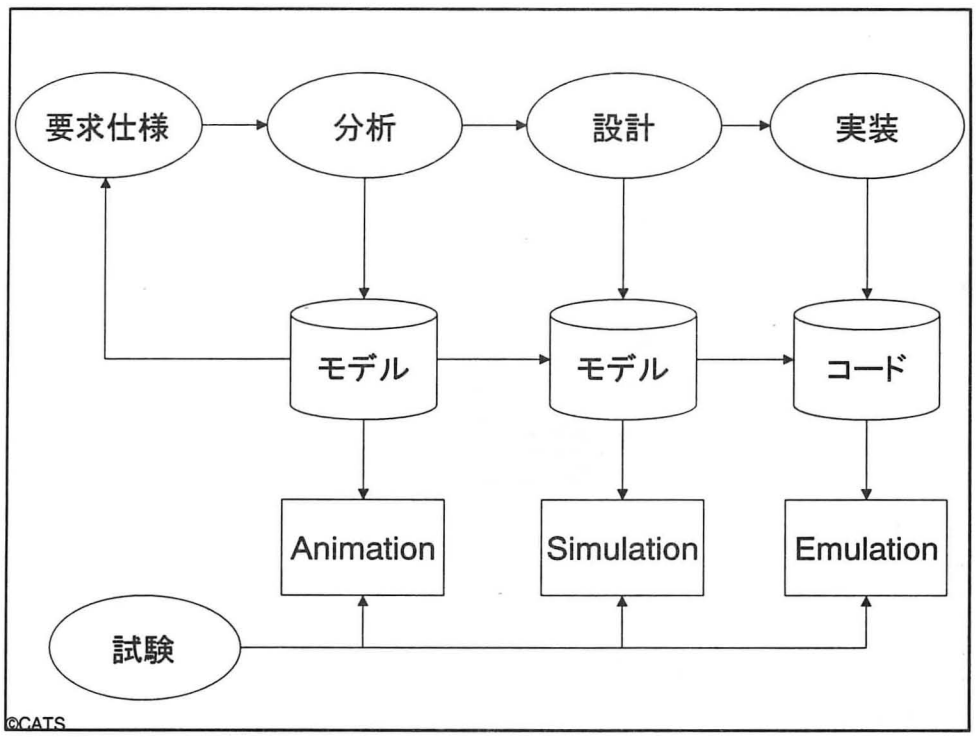
分析モデルを動く仕様書、共通モデル言語を使用して
何を作るかを開発者に共有する

何をどのように作るかを明確にするために
共通モデル言語を使用してモデルシミュレーションを行う

©CATS


プロセス
モデラ
デザイナ
要求分析
分析
アーキテクチャ設計
設計
実装
システム境界分析
・コンテキスト図
アーキテクチャ・メカニズム設計
アーキテクト
コード
(C/C++/JAVA
HDL)
試験
テスタ
利用モデル分析
・試験仕様書
テストケース
・状態遷移表
統計的テスト
©CATS


要求分析
アーキテクチャ・
メカニズム設計
テスト設計
インクリメンタル
スパイラル
計画書
コア
コア
サブシステム
サブシステム
コア
サブシステム
分析
設計
実装
テスト
分析
設計
実装
テスト
分析
設計
実装
テスト
©CATS



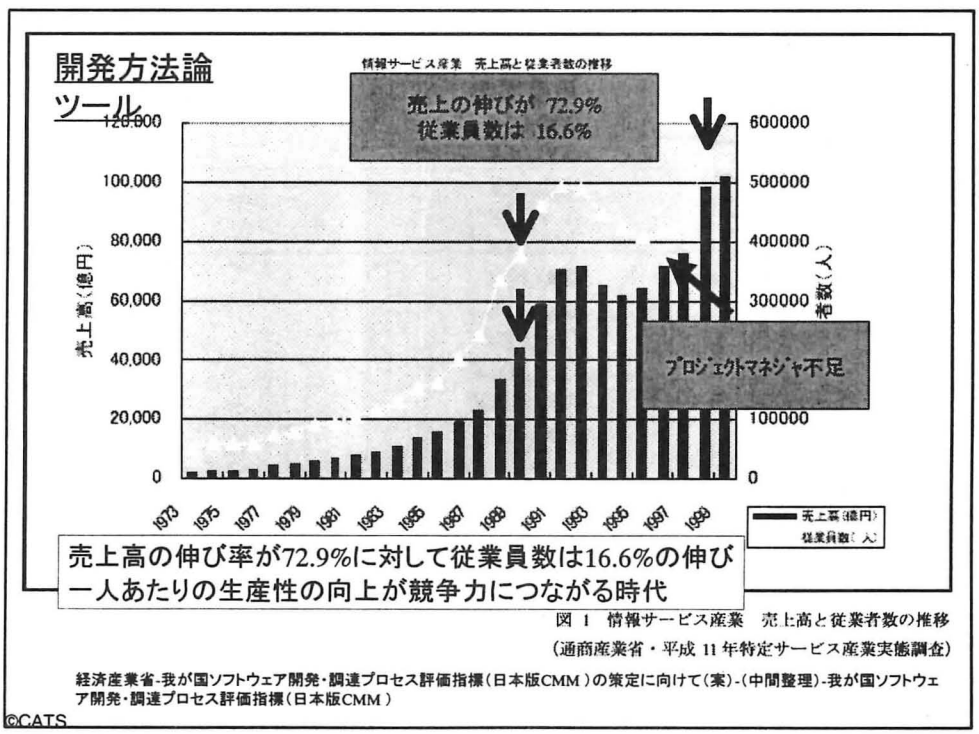
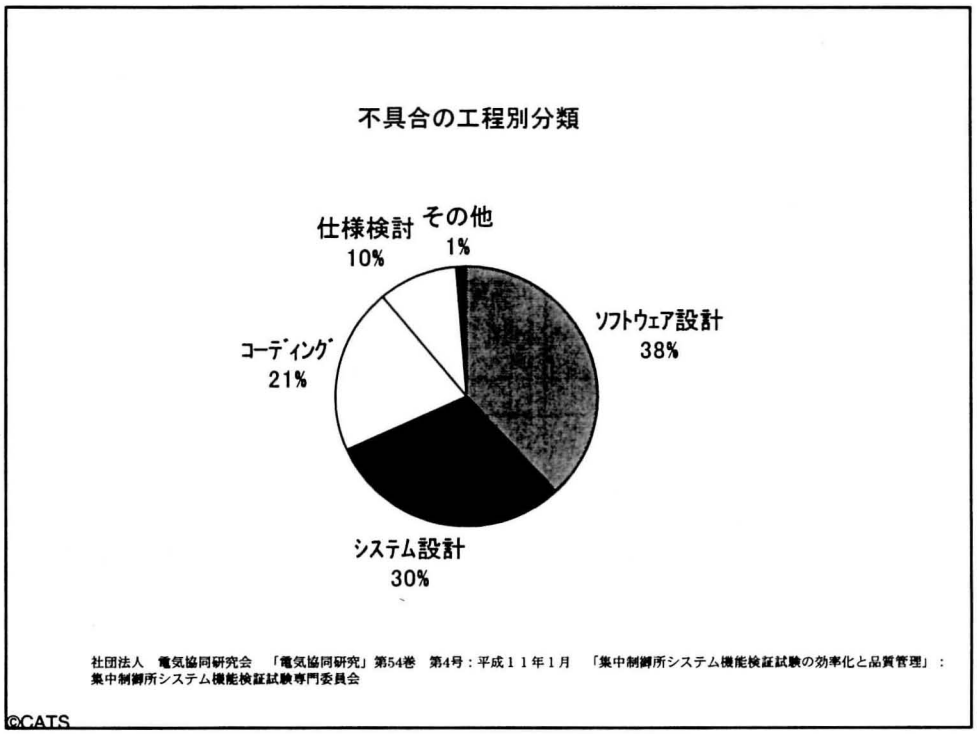


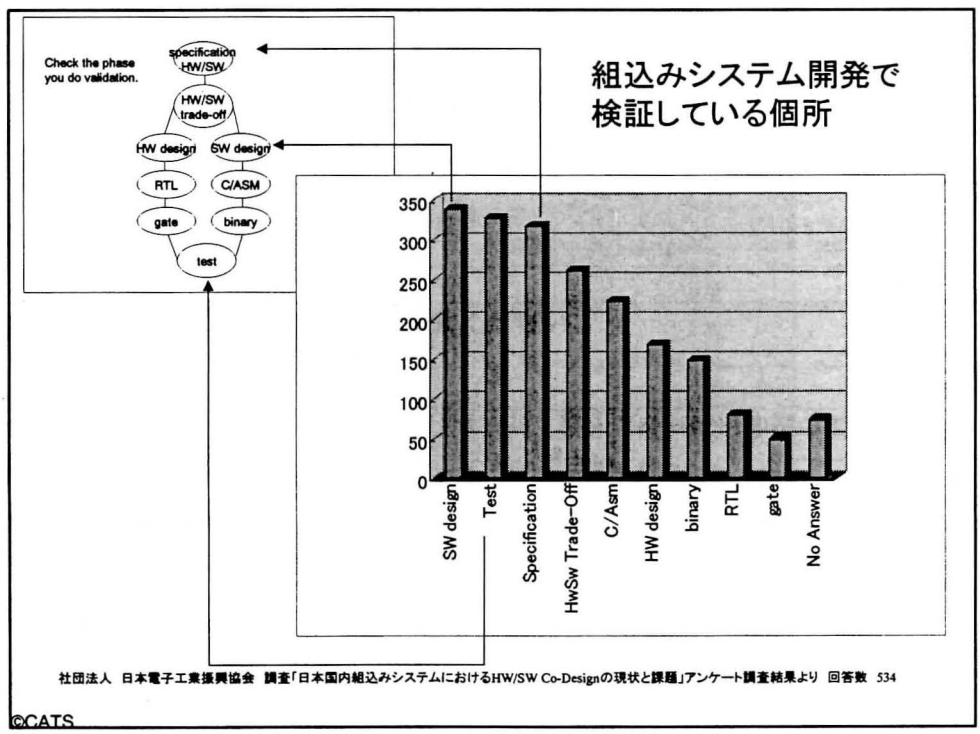
図 1 情報サービス産業 売上高と従業者数の推移
(通商産業省・平成 11 年特定サービス産業実態調査)

経済産業省-我が国ソフトウェア開発・調達プロセス評価指標(日本版CMM)の策定に向けて(案)-(中間整理)-我が国ソフトウェア開発・調達プロセス評価指標(日本版CMM)



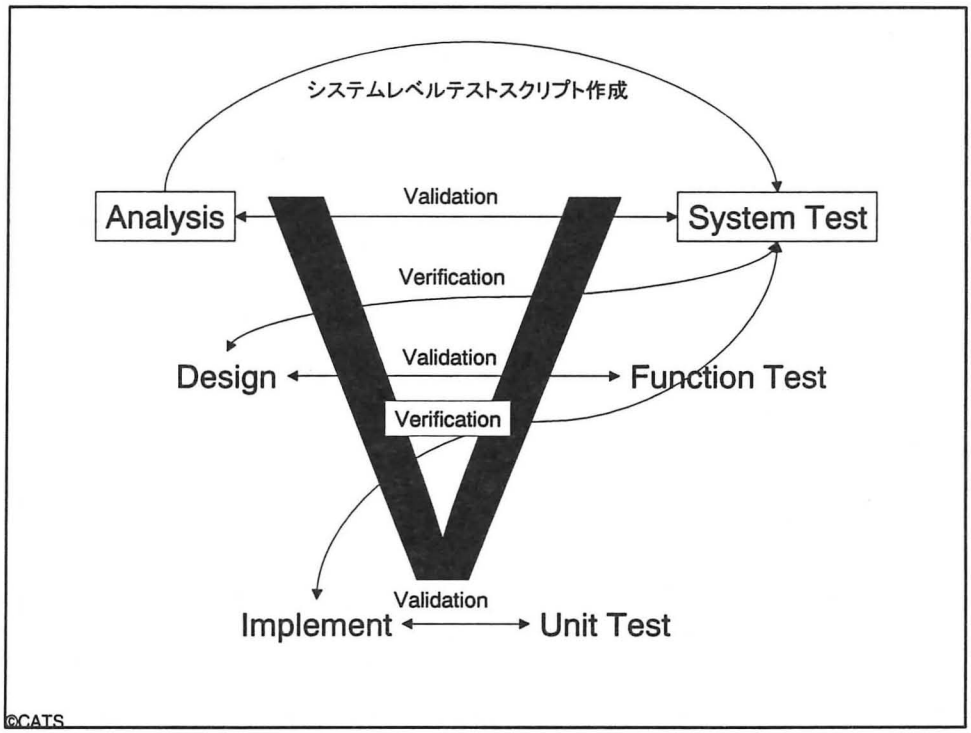
社団法人　電気協同研究会　「電気協同研究」第54巻　第4号：平成１１年１月　「集中制御所システム機能検証試験の効率化と品質管理」：集中制御所システム機能検証試験専門委員会

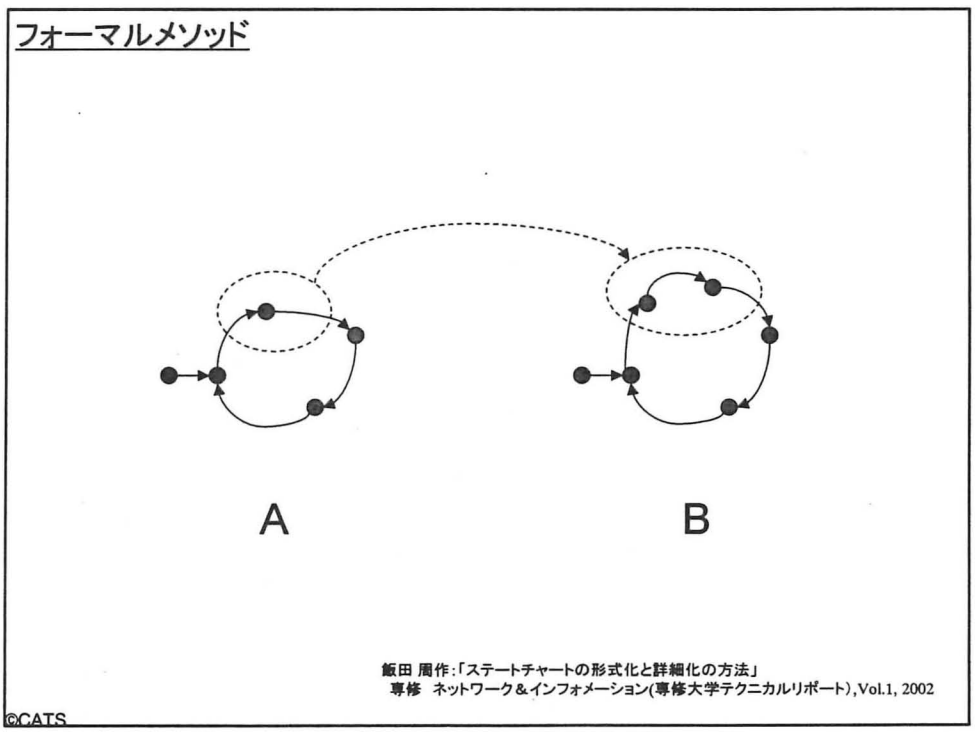
©CATS



社団法人　日本電子工業振興協会　調査「日本国内組込みシステムにおけるHW/SW Co-Designの現状と課題」アンケート調査結果より　回答数　534

©CATS


システムレベルテストスクリプト作成
Analysis
Validation
System Test
Verification
Design
Validation
Function Test
Verification
Validation
Implement
Unit Test
©CATS

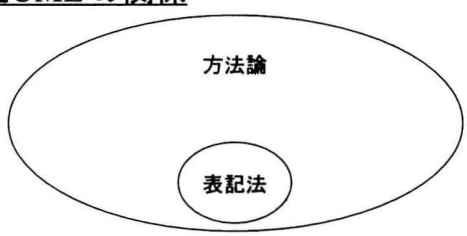

フォーマルメソッド
A
B
飯田 周作:「ステートチャートの形式化と詳細化の方法」
専修 ネットワーク＆インフォメーション(専修大学テクニカルリポート),Vol.1, 2002
©CATS

*** ワークショップの効果 ***
下記のような観点から，UMLを総合的に理解し，
仕事に有効に使用できるようになる．

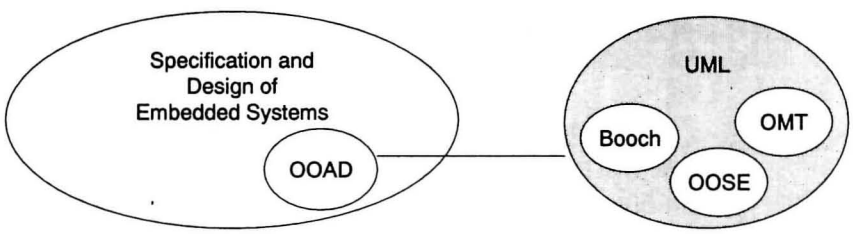
項目	eUML
UMLとはなにか	表記法
種々の方法論とUMLの関係	コミュニケーション
どのような分野にどのように使えば有効か	演繹的アプローチ
システム開発のライフサイクル上でのUMLが果たす役割	コミュニケーション
UMLの過大評価と過小評価	コミュニケーション
UMLに欠けているもの（機能，表現，考え方など）	テスト、STM、TC

©CATS

UMLとはなにか
種々の方法論とUMLの関係

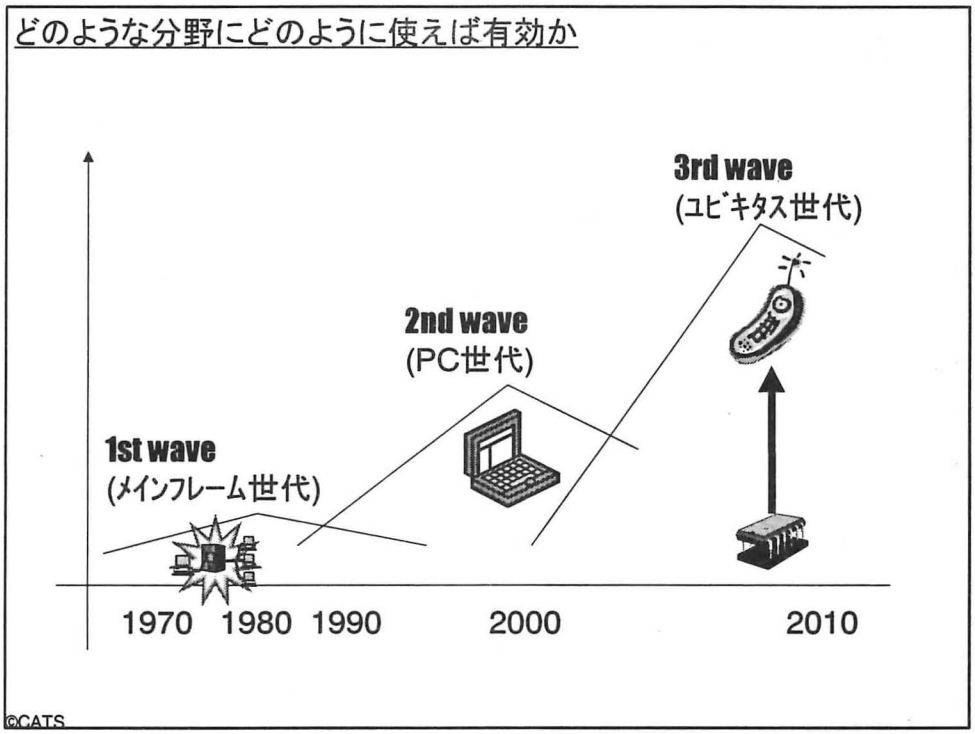


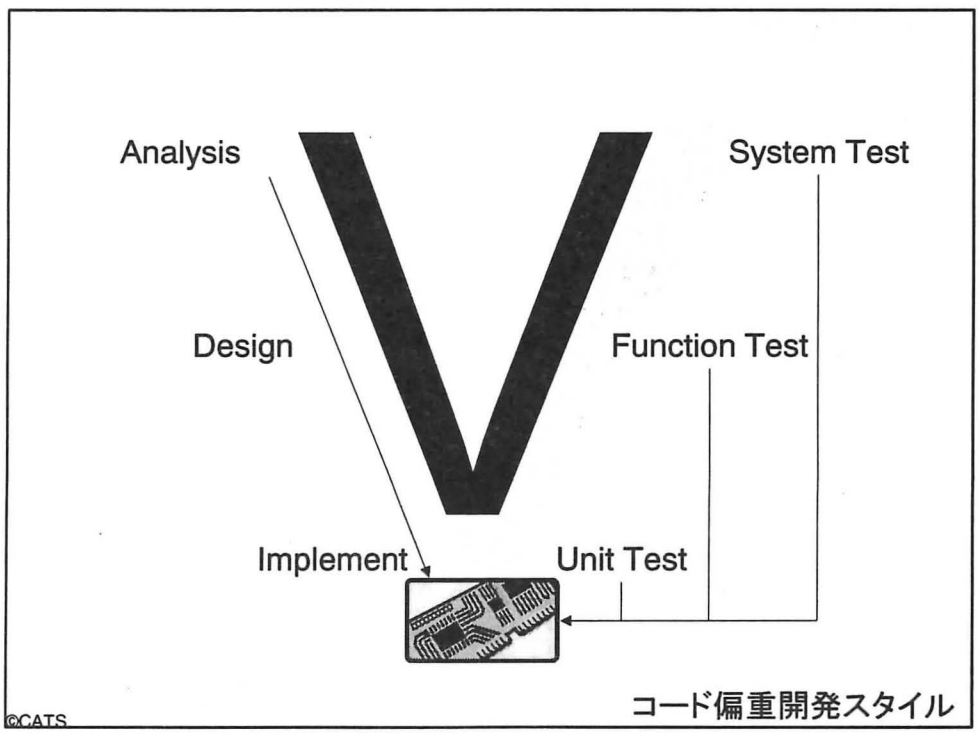
"OO doesn't slove all problems, and hence, neither does the UML."

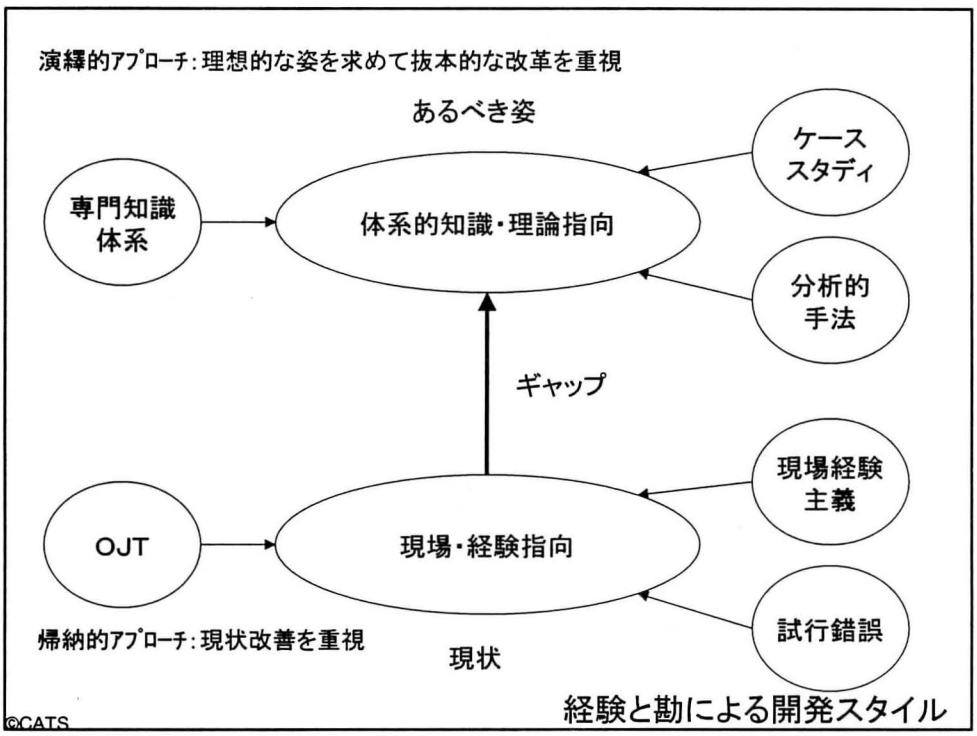


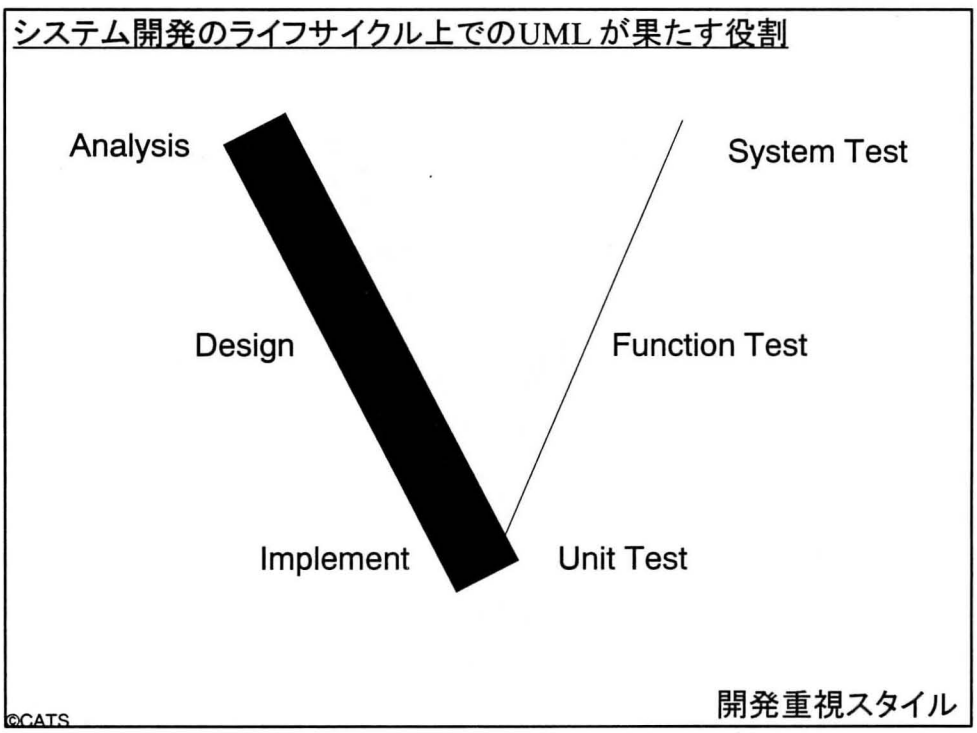
Foreword by David Harel; Bruce Powel Douglass "REAL-TIME UML DEVELOPING EFFICIENT OBJECTS FOR EMBEDDED SYSTEMS"; Addison Wesley:ISBN 0-201-32579-9

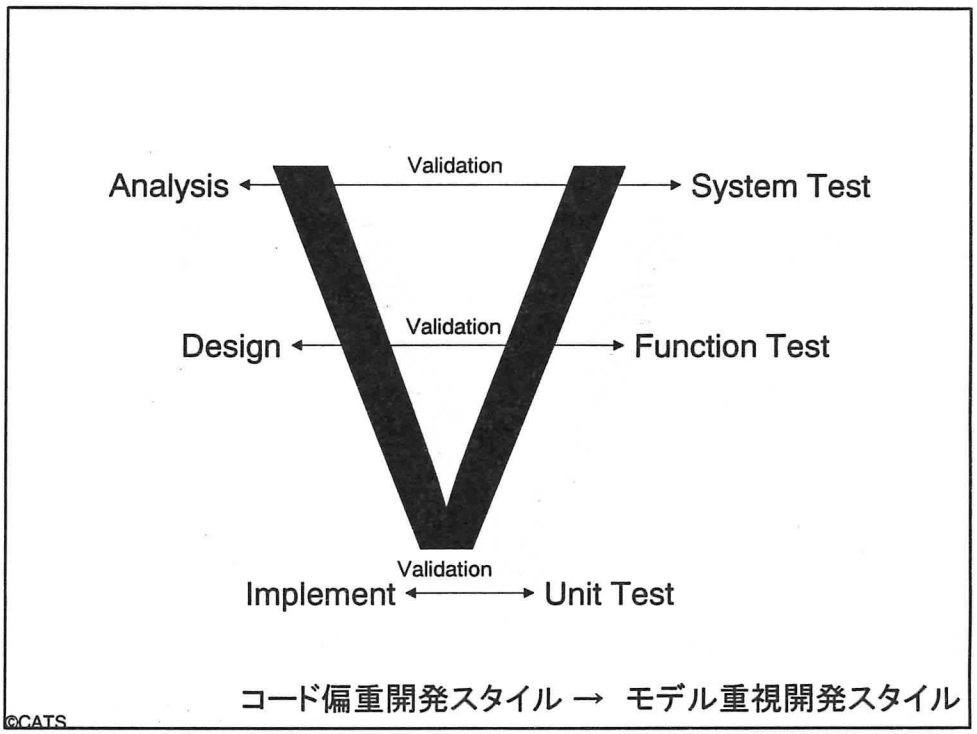
©CATS


どのような分野にどのように使えば有効か
3rd wave
(ユビキタス世代)
2nd wave
(PC世代)
1st wave
(メインフレーム世代)
1970
1980
1990
2000
2010
©CATS

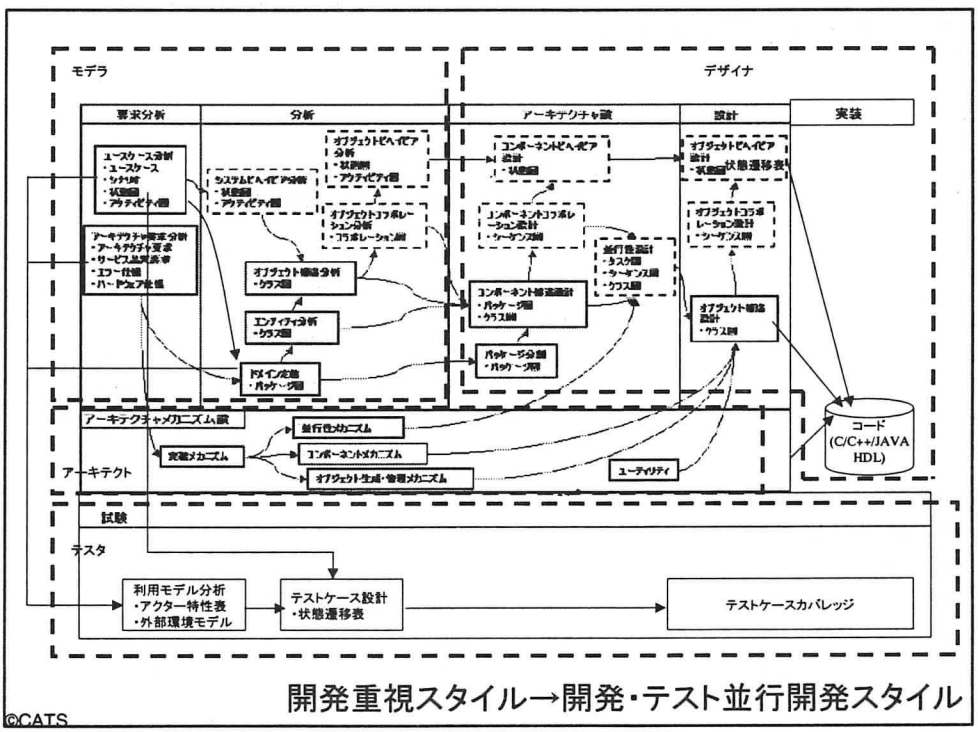

Analysis
System Test
Design
Function Test
Implement
Unit Test
コード偏重開発スタイル
©CATS


演繹的ｱﾌﾟﾛｰﾁ：理想的な姿を求めて抜本的な改革を重視
あるべき姿
ケース
スタディ
専門知識
体系
体系的知識・理論指向
分析的
手法
ギャップ
現場経験
主義
OJT
現場・経験指向
試行錯誤
帰納的ｱﾌﾟﾛｰﾁ：現状改善を重視
現状
©CATS
経験と勘による開発スタイル

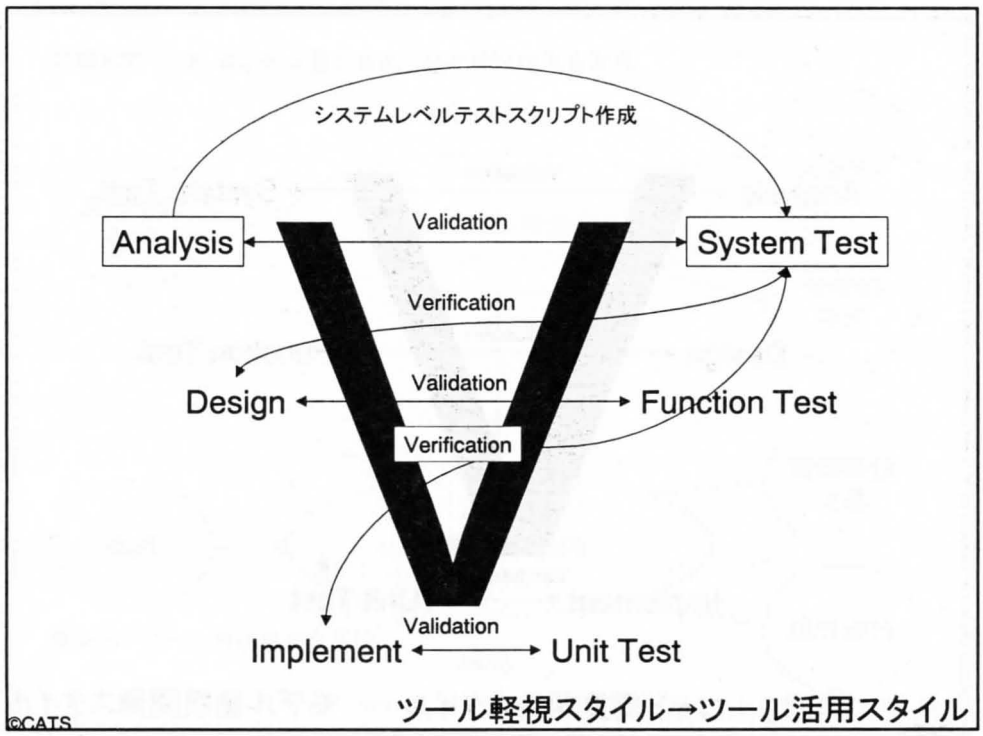

システム開発のライフサイクル上でのUML が果たす役割
Analysis
System Test
Design
Function Test
Implement
Unit Test
©CATS
開発重視スタイル

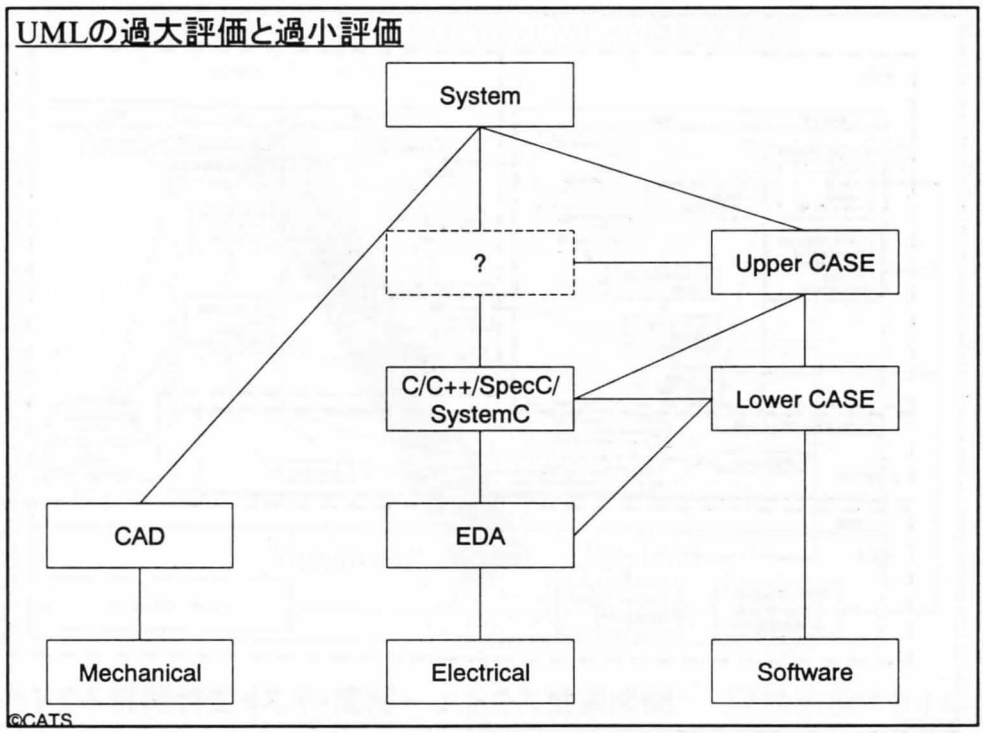


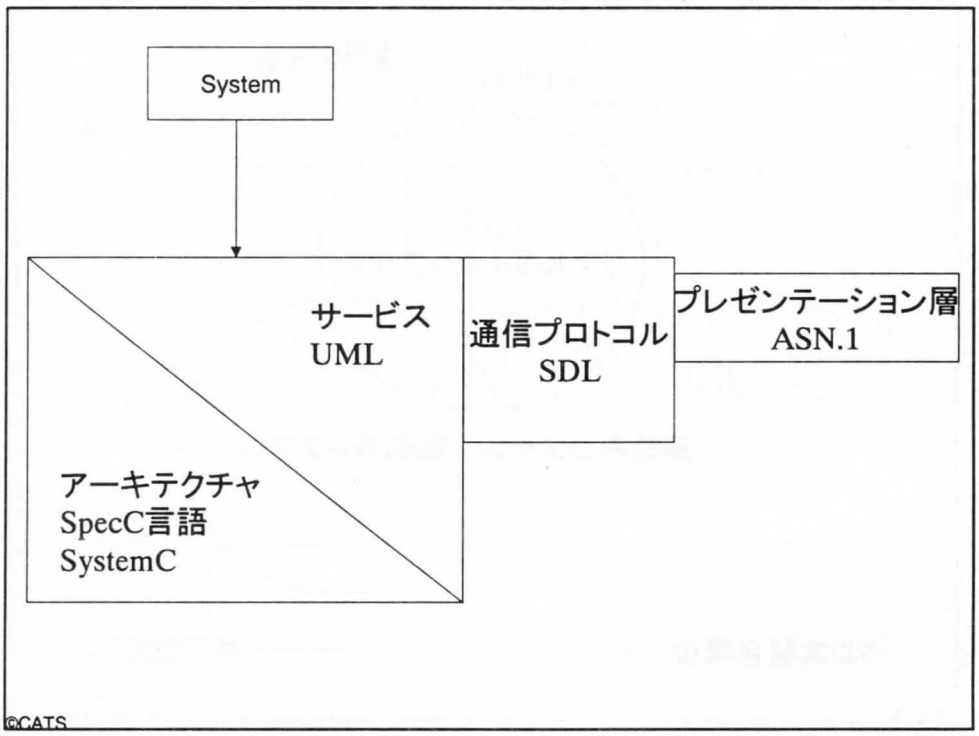
コード偏重開発スタイル → モデル重視開発スタイル

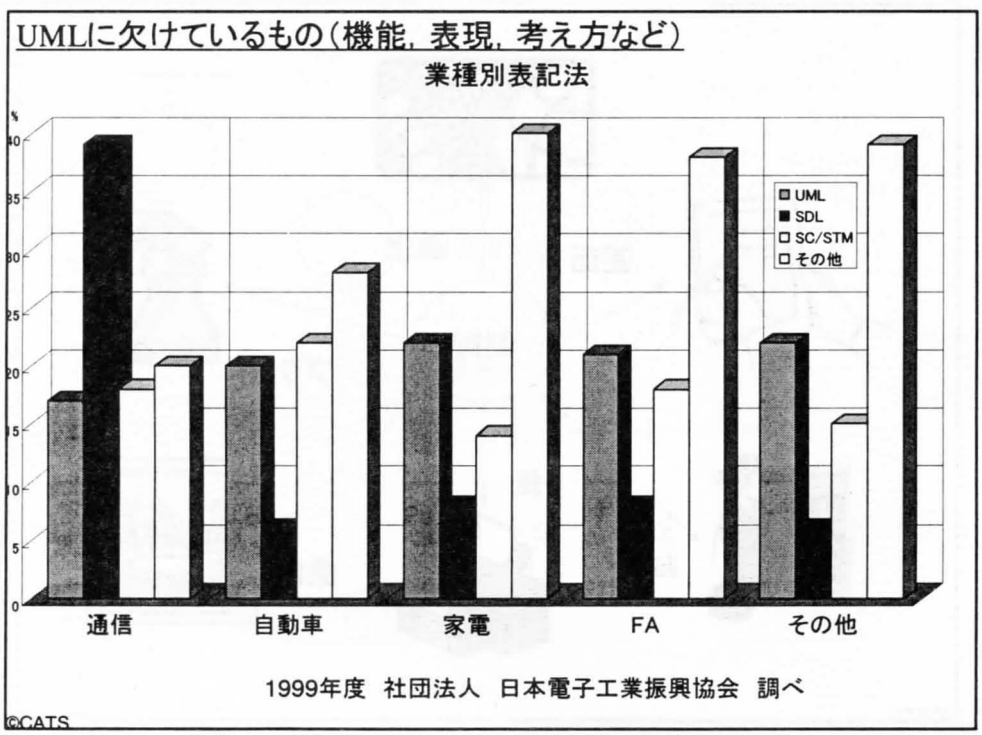


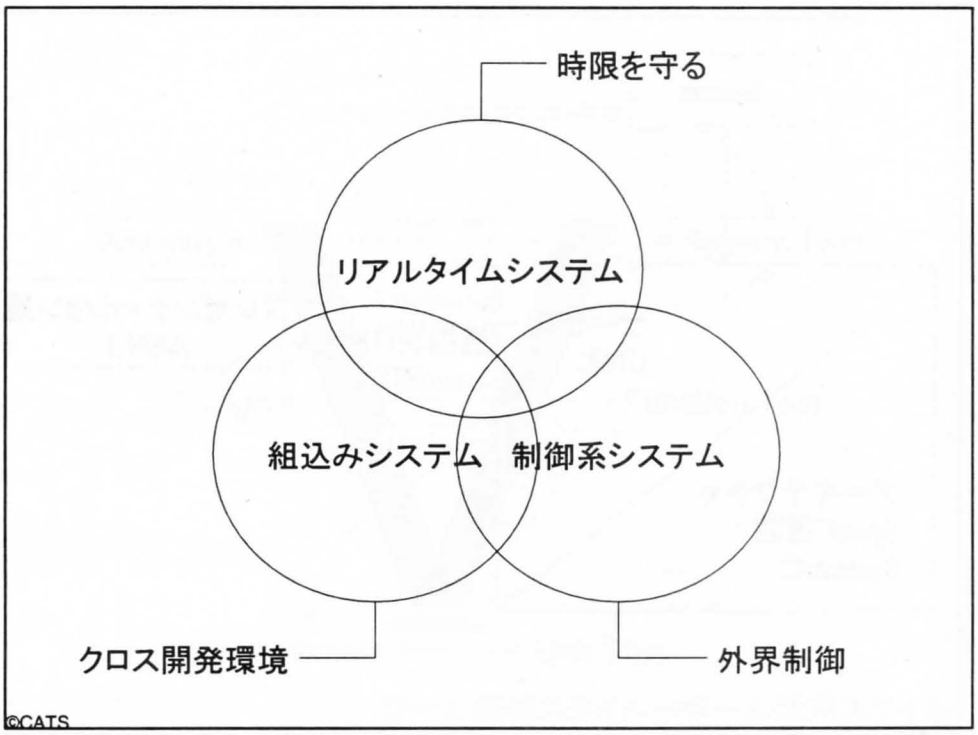
開発重視スタイル→開発・テスト並行開発スタイル

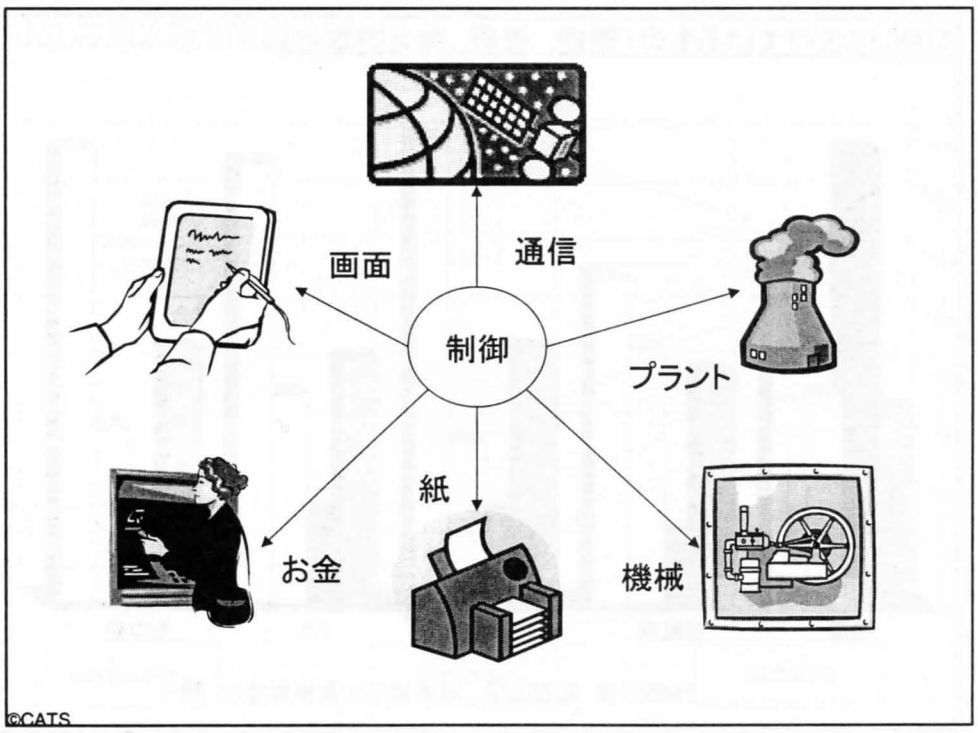

システムレベルテストスクリプト作成
Analysis
Validation
System Test
Verification
Design
Validation
Function Test
Verification
Validation
Implement
Unit Test
ツール軽視スタイル→ツール活用スタイル
©CATS

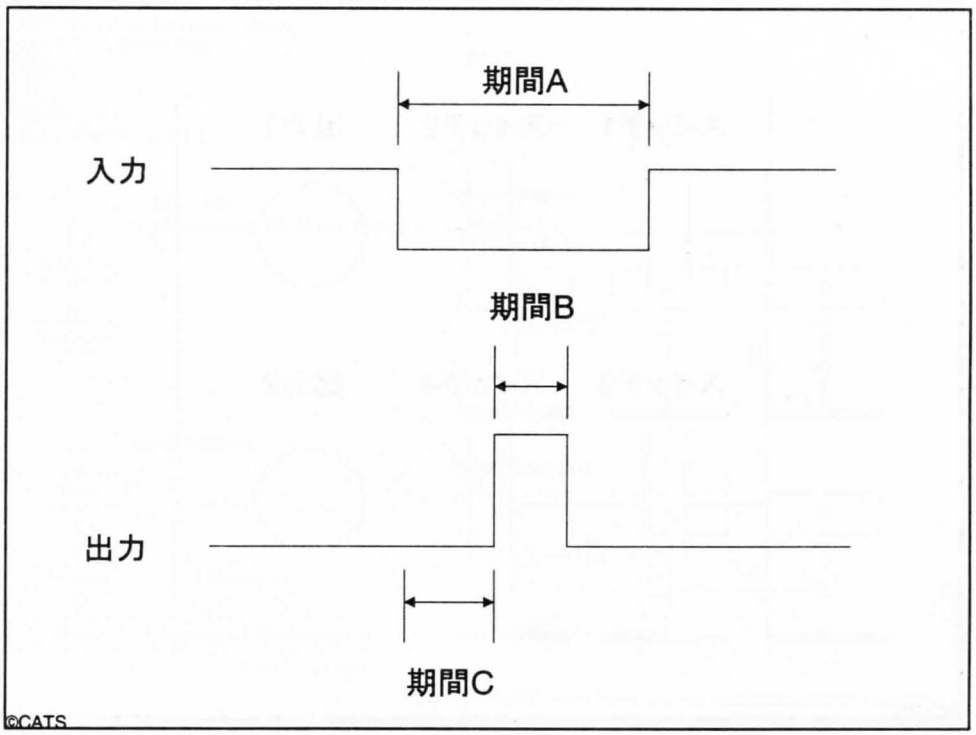

UMLの過大評価と過小評価
System
?
Upper CASE
C/C++/SpecC/
SystemC
Lower CASE
CAD
EDA
Mechanical
Electrical
Software
©CATS

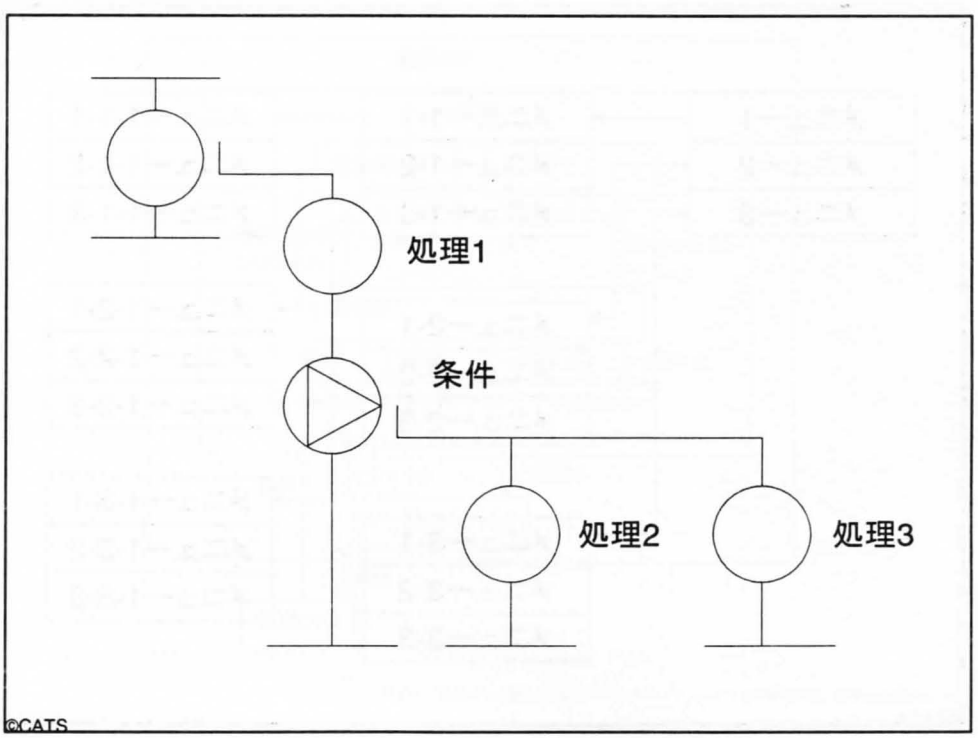

System
サービス
UML
通信プロトコル
SDL
プレゼンテーション層
ASN.1
アーキテクチャ
SpecC言語
SystemC
©CATS

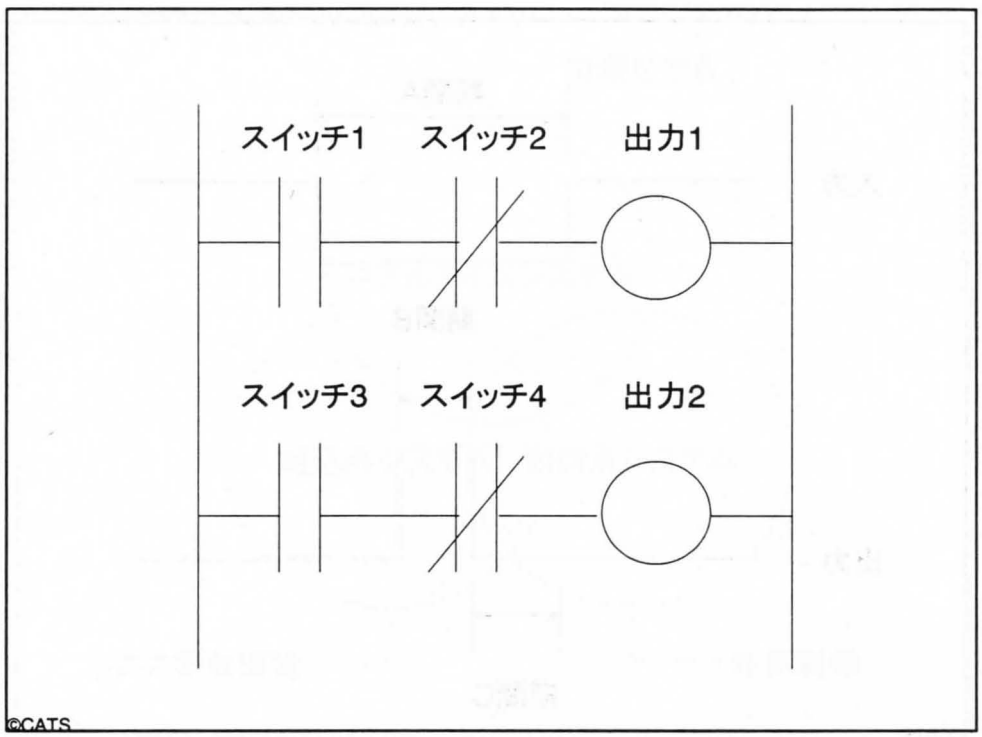

UMLに欠けているもの（機能，表現，考え方など）
業種別表記法
%
40
35
30
25
20
15
10
5
0
UML
SDL
SC/STM
その他
通信
自動車
家電
FA
その他
1999年度 社団法人 日本電子工業振興協会 調べ
©CATS

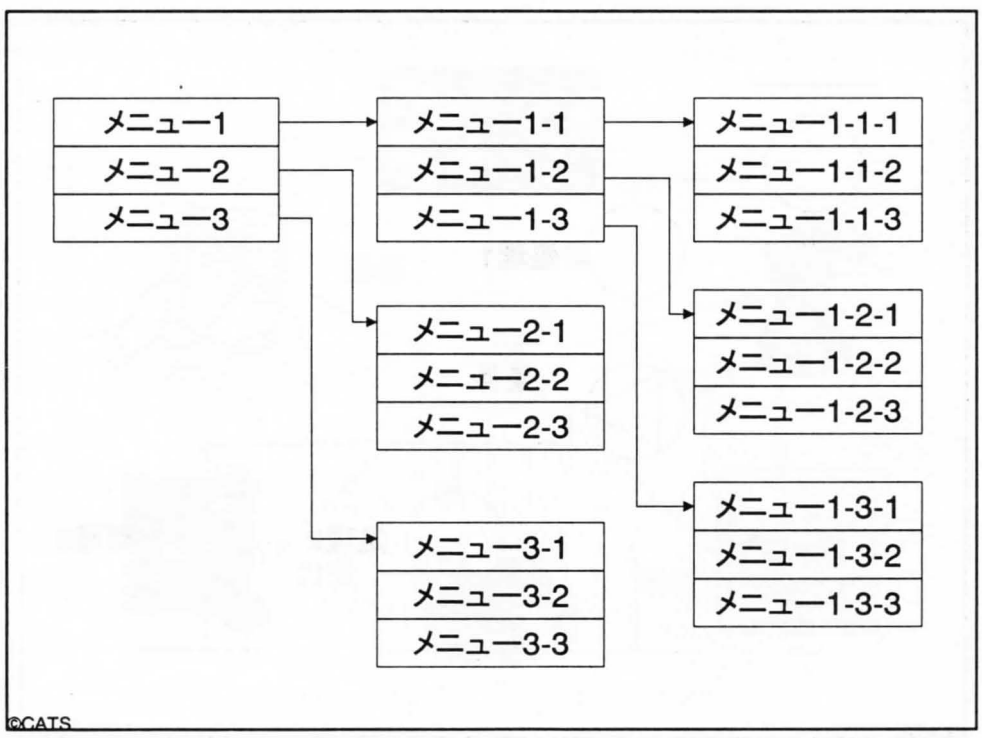

時限を守る
リアルタイムシステム
組込みシステム
制御系システム
クロス開発環境
外界制御
©CATS

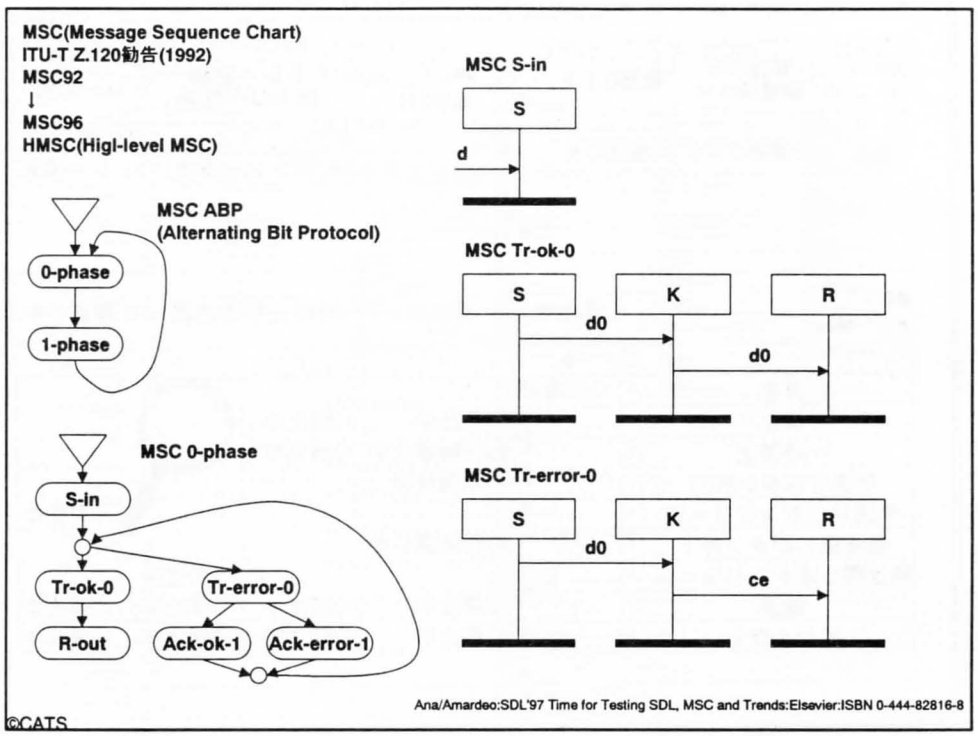

画面
通信
制御
プラント
紙
お金
機械
©CATS

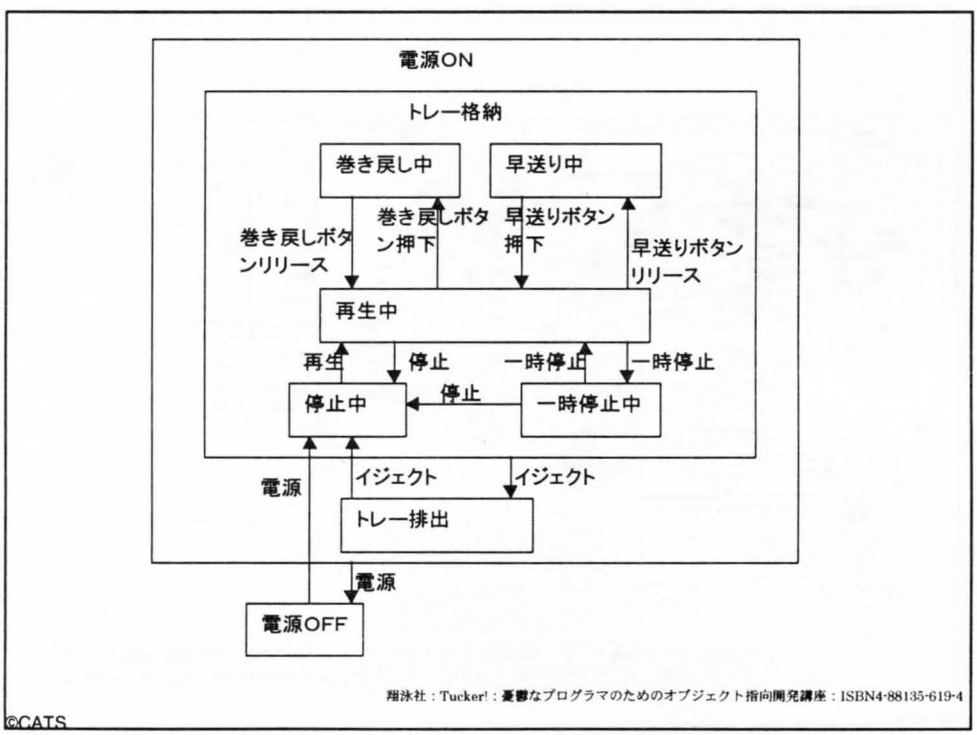

期間A
入力
期間B
出力
期間C
©CATS


処理1
条件
処理2
処理3
©CATS


スイッチ1
スイッチ2
出力1
スイッチ3
スイッチ4
出力2
©CATS


メニュー1
メニュー2
メニュー3
メニュー1-1
メニュー1-2
メニュー1-3
メニュー1-1-1
メニュー1-1-2
メニュー1-1-3
メニュー2-1
メニュー2-2
メニュー2-3
メニュー1-2-1
メニュー1-2-2
メニュー1-2-3
メニュー3-1
メニュー3-2
メニュー3-3
メニュー1-3-1
メニュー1-3-2
メニュー1-3-3
©CATS


MSC(Message Sequence Chart)
ITU-T Z.120勧告(1992)
MSC92
↓
MSC96
HMSC(Higl-level MSC)
MSC ABP
(Alternating Bit Protocol)
0-phase
1-phase
MSC 0-phase
S-in
Tr-ok-0
Tr-error-0
R-out
Ack-ok-1
Ack-error-1
MSC S-in
S
d
MSC Tr-ok-0
S
K
R
d0
d0
MSC Tr-error-0
S
K
R
d0
ce
Ana/Amardeo:SDL'97 Time for Testing SDL, MSC and Trends:Elsevier:ISBN 0-444-82816-8
©CATS

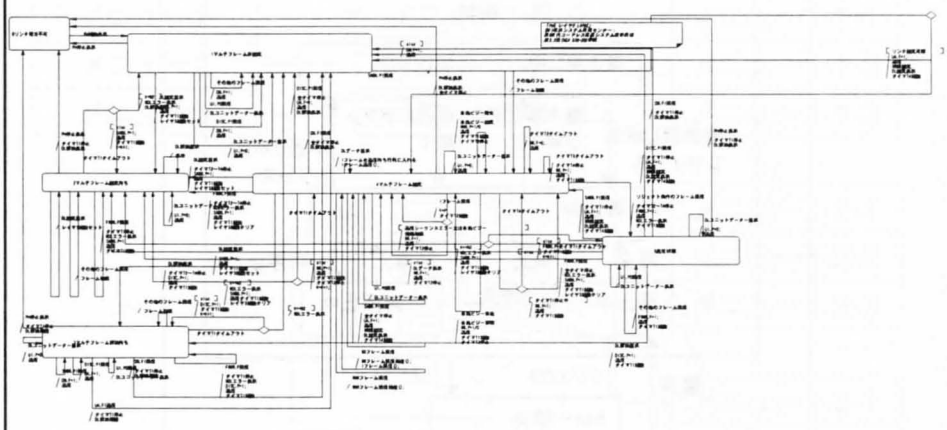

電源ON
トレー格納
巻き戻し中
早送り中
巻き戻しボタンリリース
巻き戻しボタン押下
早送りボタン押下
早送りボタンリリース
再生中
再生
停止
一時停止
一時停止
停止中
停止
一時停止中
電源
イジェクト
イジェクト
トレー排出
電源
電源OFF
翔泳社：Tucker!：憂鬱なプログラマのためのオブジェクト指向開発講座：ISBN4-88135-619-4
©CATS

■0 CD		電源ON ■電源ON	電源ＯＦＦ
	E \ S	0	1
電源	0	=>電源ＯＦＦ	=>電源ON

■0.1 電源ON		トレー格納 ■トレー格納	トレー排出
	E \ S	0	1
イジェクト	0	=>トレー排出	=>トレー格納

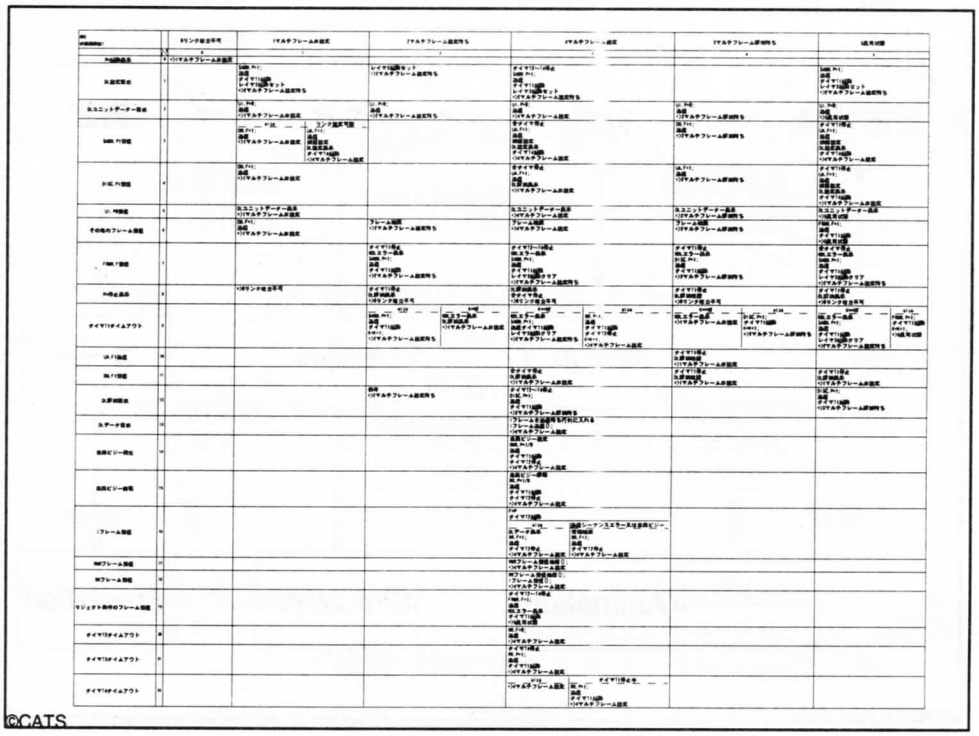
■0.1.1 トレー格納		停止中	再生中	一時停止中	巻き戻し中	早送り中
	E \ S	0	1	2	3	4
再生	0	=>再生中				
停止	1		=>停止中	=>停止中		
一時停止	2		=>一時停止中	=>再生中		
早送りボタン押下	3		=>早送り中			
早送りボタンリリース	4					=>再生中
巻き戻しボタン押下	5		=>巻き戻し中			
巻き戻しボタンリリース	6				=>再生中	
電源	7		=>停止中	=>停止中	=>停止中	=>停止中
イジェクト	8		=>停止中	=>停止中	=>停止中	=>停止中

©CATS

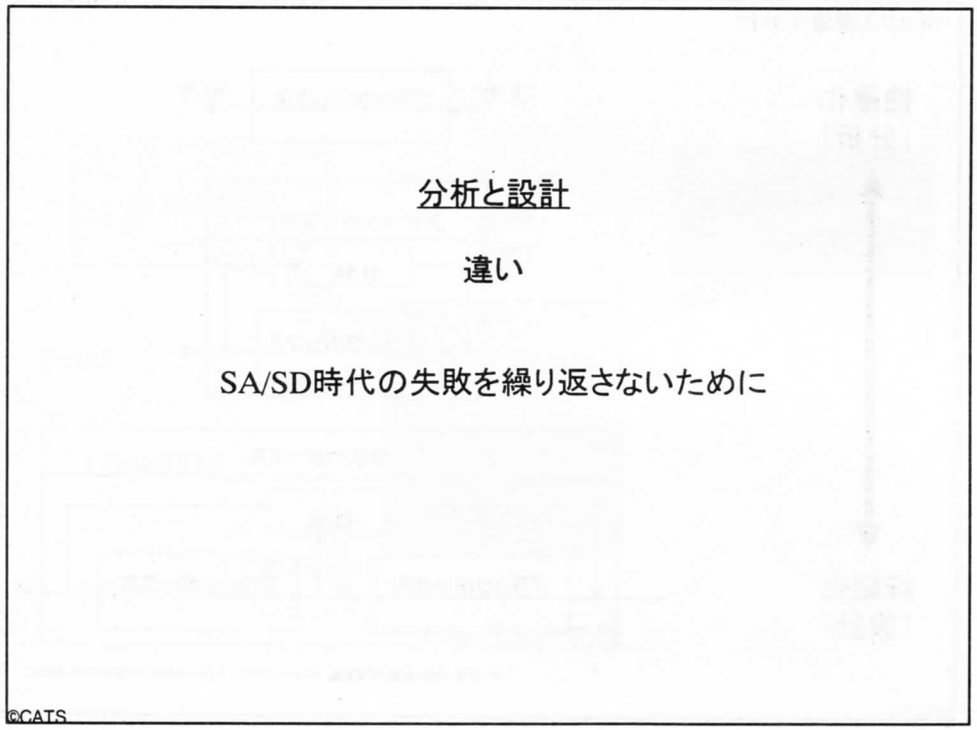


(財)電波システム開発センター:第2世代コードレス電話システム標準規格　第3.2版 (RCR STD-28)
「レイヤ２規格リンクアクセス手順 ＬＡＰＤＣ (Link Access Procedure for Digital Cordless)」
「レイヤ２ ＳＤＬ図」を元に作成したＳＴＤ

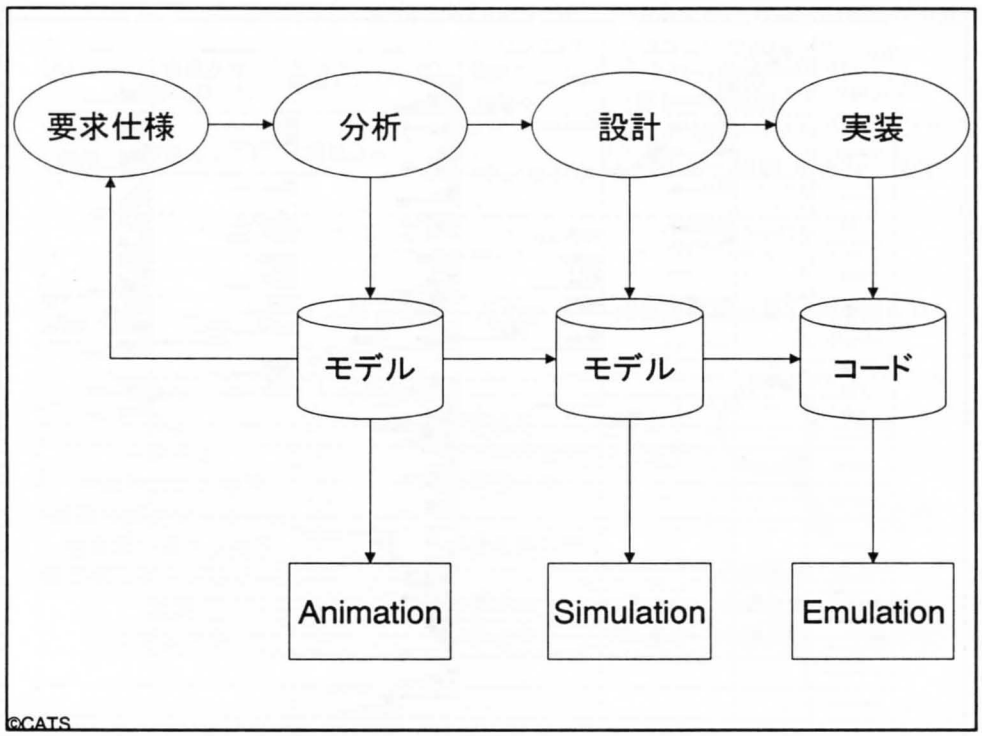
©CATS

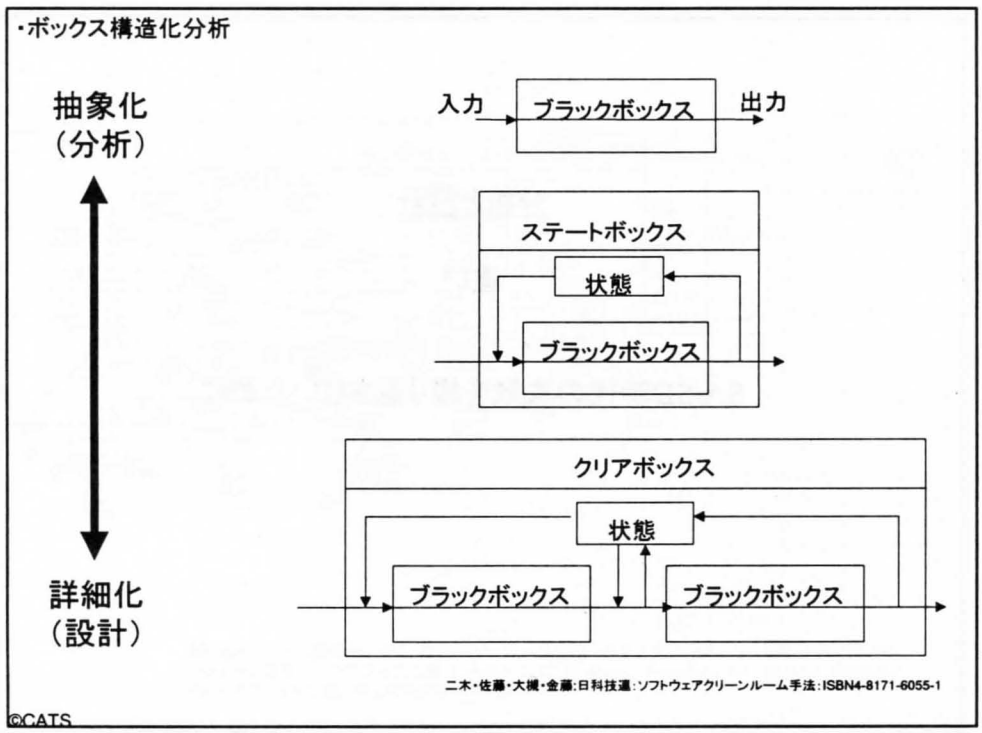


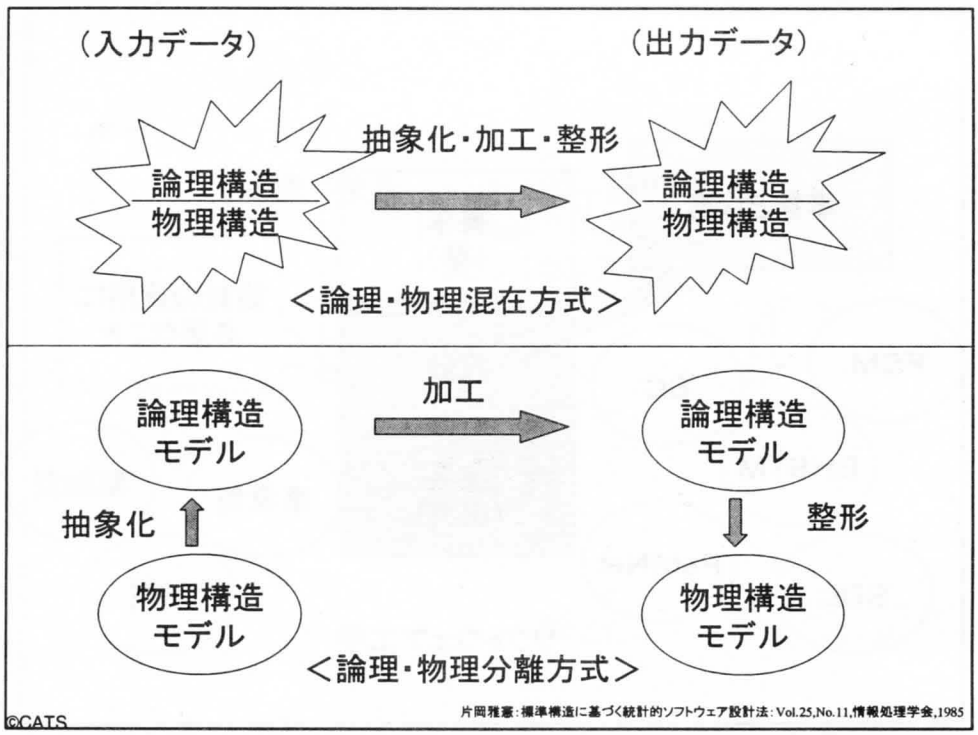
©CATS

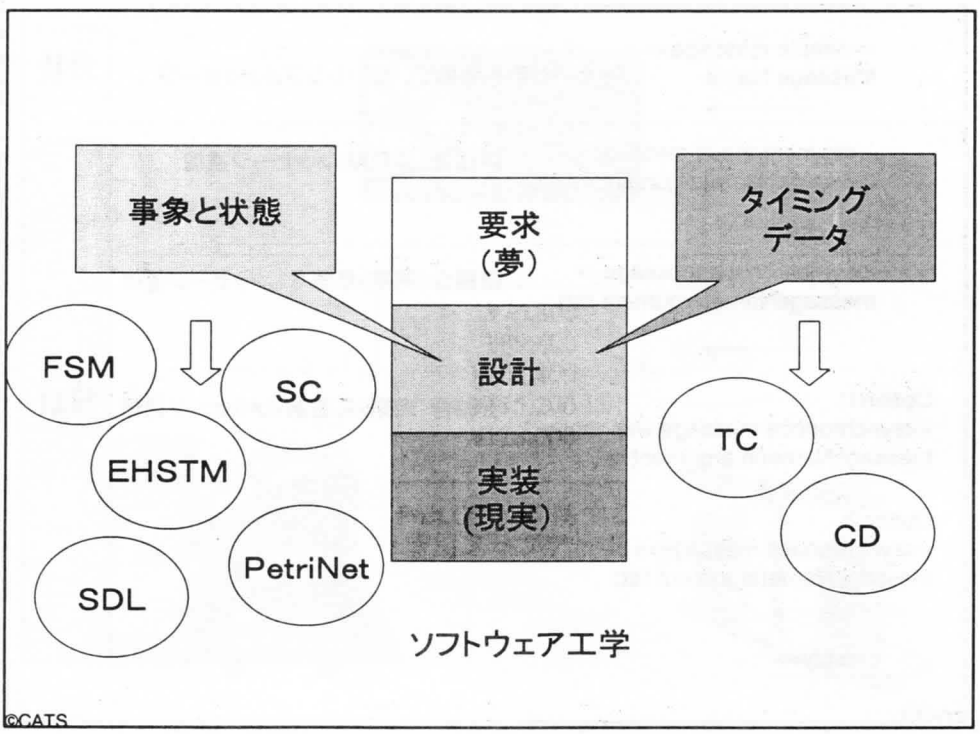


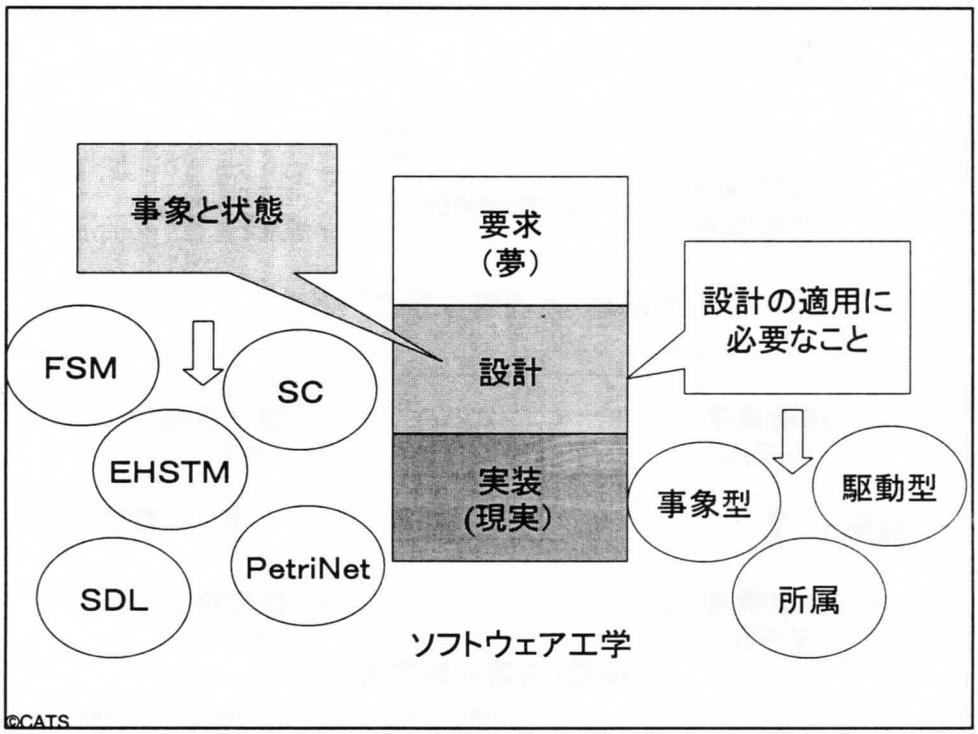
©CATS

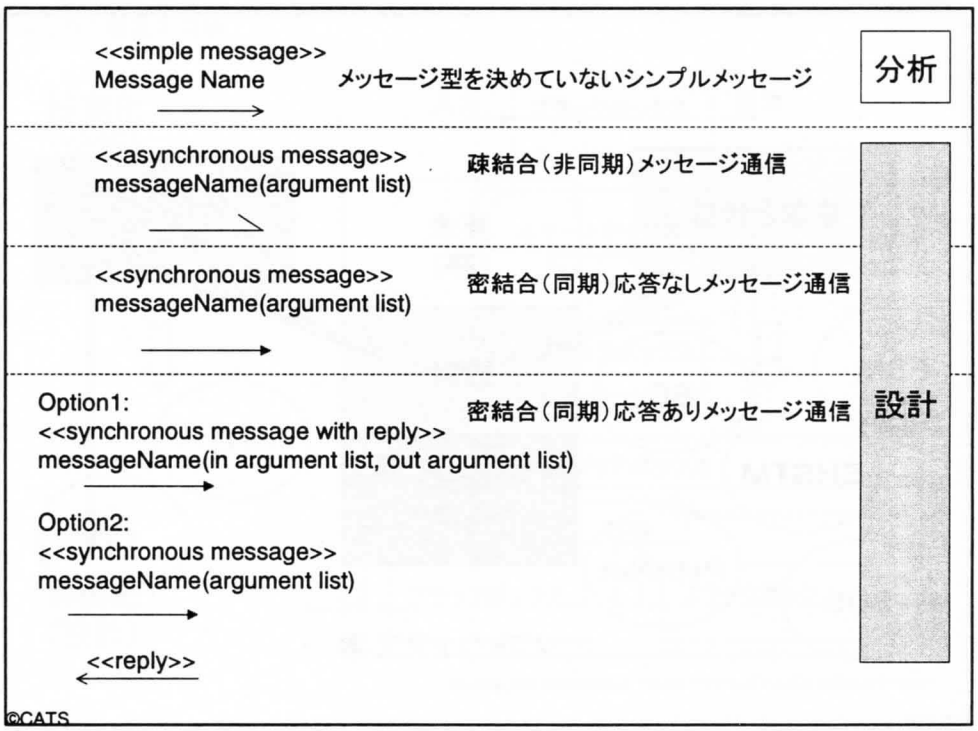

要求仕様
分析
設計
実装
モデル
モデル
コード
Animation
Simulation
Emulation
©CATS

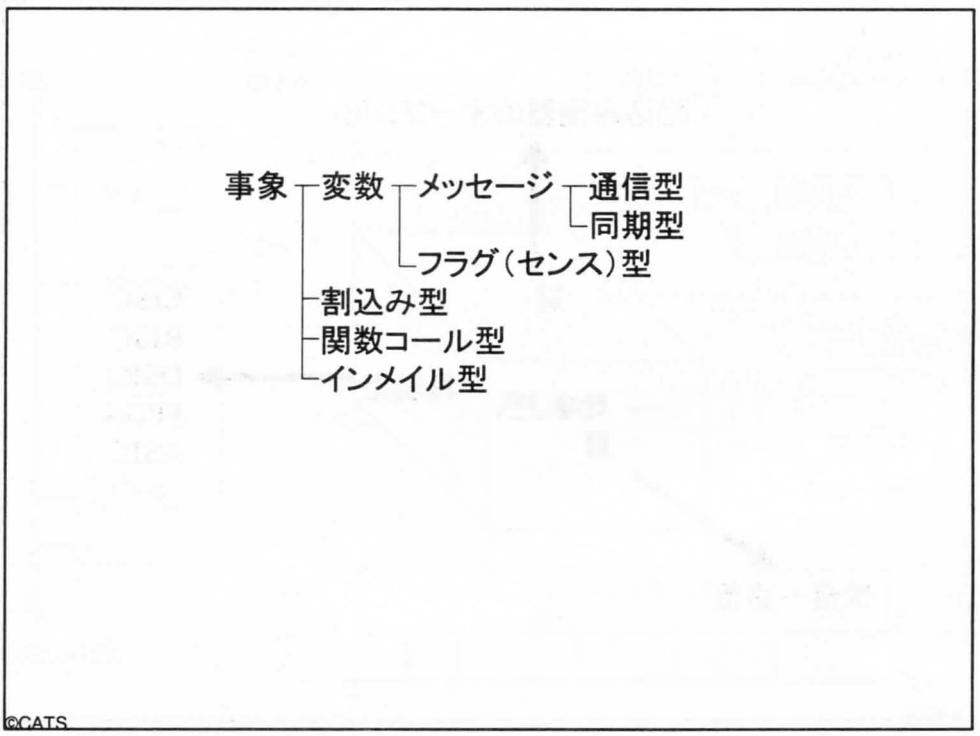

・ボックス構造化分析
抽象化
（分析）
詳細化
（設計）
入力
ブラックボックス
出力
ステートボックス
状態
ブラックボックス
クリアボックス
状態
ブラックボックス
ブラックボックス
二木・佐藤・大槻・金藤:日科技連:ソフトウェアクリーンルーム手法:ISBN4-8171-6055-1
©CATS

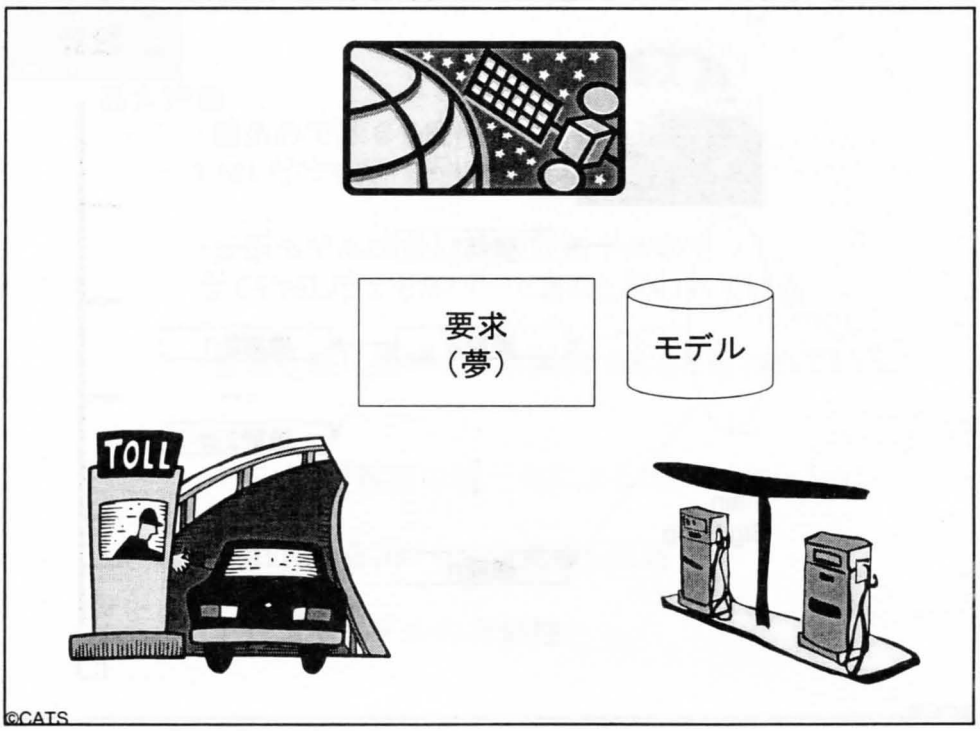

（入力データ）
（出力データ）
抽象化・加工・整形
論理構造
物理構造
論理構造
物理構造
＜論理・物理混在方式＞
加工
論理構造
モデル
論理構造
モデル
抽象化
整形
物理構造
モデル
物理構造
モデル
＜論理・物理分離方式＞
©CATS
片岡雅憲：標準構造に基づく統計的ソフトウェア設計法：Vol.25,No.11,情報処理学会,1985

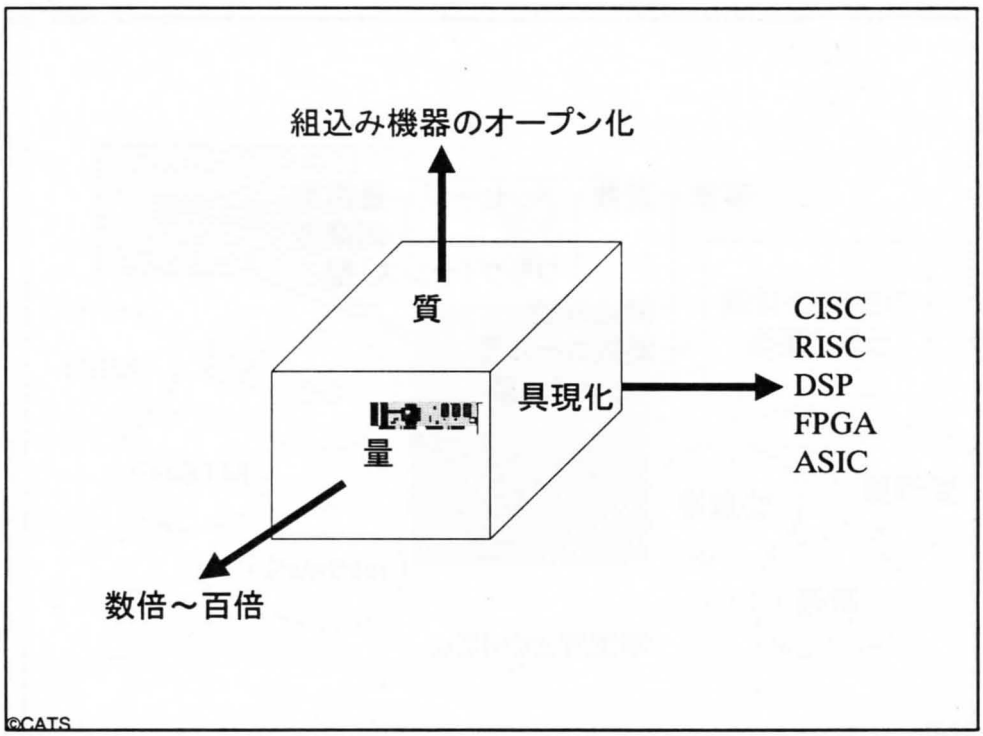

事象と状態
要求
（夢）
タイミング
データ
FSM
SC
設計
TC
EHSTM
実装
（現実）
CD
PetriNet
SDL
ソフトウェア工学
©CATS

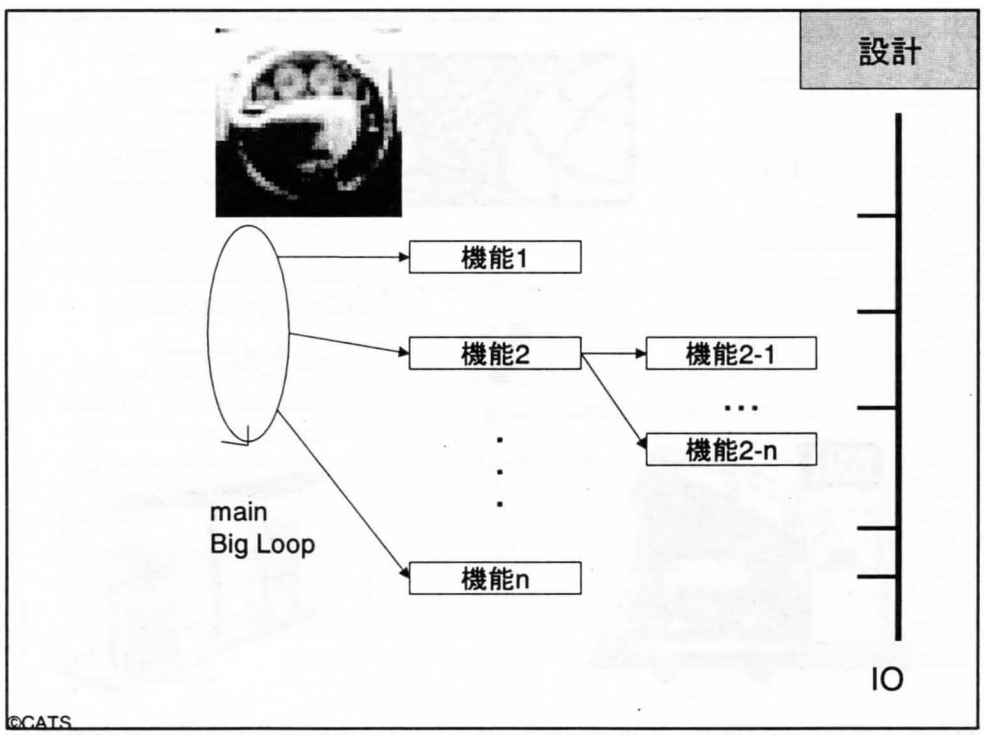

事象と状態
要求
(夢)
設計の適用に
必要なこと
FSM
SC
設計
EHSTM
実装
(現実)
事象型
駆動型
PetriNet
SDL
所属
ソフトウェア工学
©CATS

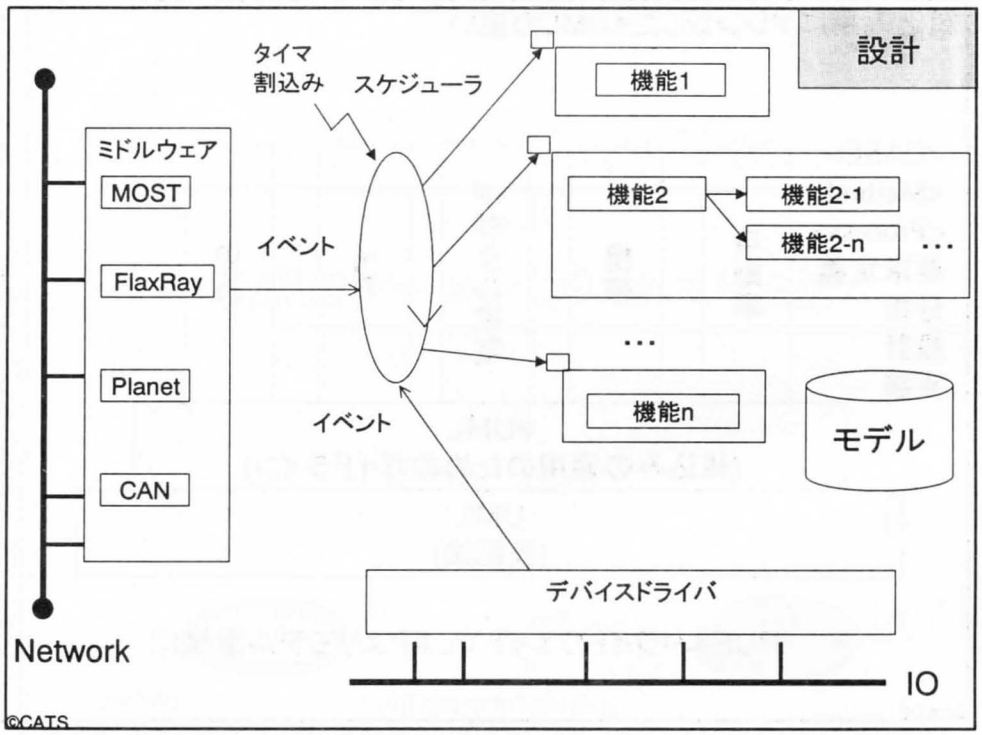

<<simple message>>
Message Name
メッセージ型を決めていないシンプルメッセージ
分析
<<asynchronous message>>
messageName(argument list)
疎結合(非同期)メッセージ通信
<<synchronous message>>
messageName(argument list)
密結合(同期)応答なしメッセージ通信
Option1:
<<synchronous message with reply>>
messageName(in argument list, out argument list)
密結合(同期)応答ありメッセージ通信
設計
Option2:
<<synchronous message>>
messageName(argument list)
<<reply>>
©CATS


事象
変数
メッセージ
通信型
同期型
フラグ(センス)型
割込み型
関数コール型
インメイル型
©CATS


要求
(夢)
モデル
TOLL
©CATS


組込み機器のオープン化
質
量
具現化
CISC
RISC
DSP
FPGA
ASIC
数倍～百倍
©CATS


設計
機能1
機能2
機能2-1
機能2-n
機能n
main
Big Loop
IO
©CATS



過大評価:

・組込みで数多く使用されている表記法を
UMLだけでカバーできると思われている

・分析モデルは良いが設計モデルまで
全てUMLだけでカバーできると思われている

・要求仕様は曖昧のまま開発できると思われている

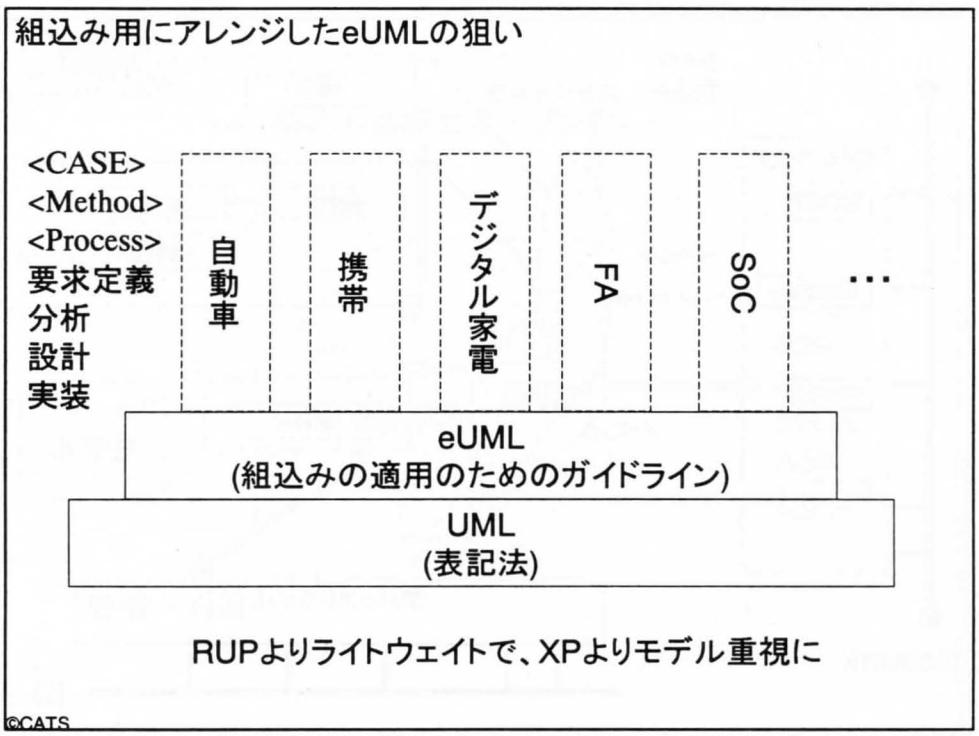
過小評価:

・モデリング言語の統一化によるコミュニケーション

・UMLによるプロセスの見直し気運

・実行可能モデルの有効性

©CATS


組込み用にアレンジしたeUMLの狙い
<CASE>
<Method>
<Process>
要求定義
分析
設計
実装
自動車
携帯
デジタル家電
FA
SoC
…
eUML
(組込みの適用のためのガイドライン)
UML
(表記法)
RUPよりライトウェイトで、XPよりモデル重視に
©CATS

開発ツール／プロセスとUML

Nil Software Corp.
Masao Ito

2002/7/12 (c) 2001 Nil Software Corp. 1

目次

- 「大きな物語」の終焉
- プロセスとツール
- ツール・プロセス・UML語られない小さな物語

2002/7/12 (c) 2001 Nil Software Corp. 2

「大きな物語(grand narrative)」の終焉 - JEAN-FRANCOIS LYOTARD -

- 新しい言語が古い言語に付け加わり，古都の郊外を形成する．...今日では，それに機械言語，ゲーム理論の行列式，新しい記譜法，．．．，遺伝子の言語，音韻構造のグラフなどを付け加えることもできるだろう．
- 誰一人としてこれら全ての言語を語るものはなく，これらの言語は普遍的なメタ言語をもたず，体系=主体の計画は挫折に終わり，人はそれぞれ個別的認識の実証主義のうちに沈み込み，学者は科学者となり，研究作業はもはや誰も統括することができないほど細分化された作業となってしまっている．（以上，「ポストモダンの条件」，リオタール，水声社）

2002/7/12 (c) 2001 Nil Software Corp. 3

ツールとプロセスにおけるかつての (そして今の?)「大きな物語」の終焉

- ツール(90年前後)
  - 生産性を数倍にするCASEツール
  - 品質が数倍になるツール
- プロセス(70年-今)
  - 使われない開発標準
  - ライフサイクルモデル
  - CMMなどの評価モデル
- UML(今?)
  - これさえ知っていれば...

2002/7/12 (c) 2001 Nil Software Corp. 4

何故プロセスでツールか

◈ 以下でモデル化してみる.

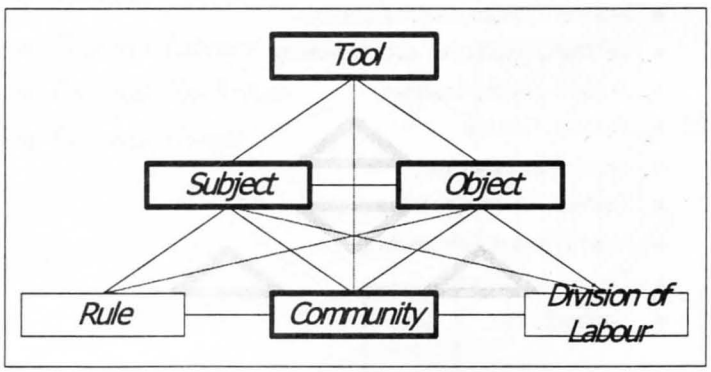
ブラックボックス　　プロダクト

Ⅰ. 活動理論
Ⅱ. 認知
Ⅲ. プロセス記述
・・・

2002/7/12 (c) 2001 Nil Software Corp. 5

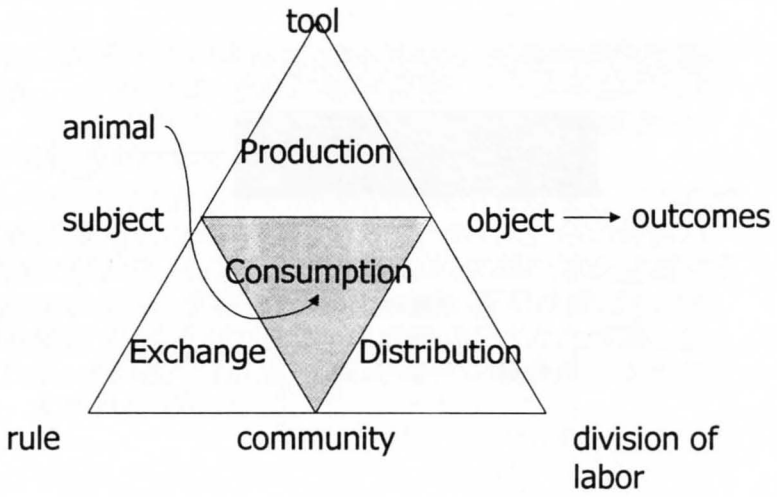
Ⅰ ツールの位置付け
Fundamental of our Environment

◈ Activity Theory (Engeström)



2002/7/12 (c) 2001 Nil Software Corp. 6

I ツールの位置付け
1+3 Sections (animal – human)



2002/7/12 (c) 2001 Nil Software Corp. 7

I どのようなツールが必要か:ECMA-TR69
垂直型, システム

- *System Engineering Services*
  - *System Requirements Engineering*
  - *System Design and Allocation*
  - ***System Simulation and Modeling***
  - *System Static Analysis*
  - *System Testing*
  - *System Integration*
  - *System Re-engineering*
  - *Host-Target Connection*
  - *Target Monitoring*
  - *Traceability*

2002/7/12 (c) 2001 Nil Software Corp. 8

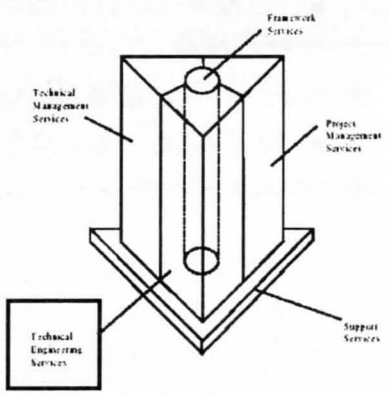
Iどのようなツールが必要か:ECMA-TR69 垂直型, ソフトウェア工学

- ・Software Engineering Services
  - **Software Requirements Engineering**
  - **Software Design**
  - **Software Simulation and Modeling**
  - Software Verification
  - **Software Generation**
  - Compilation
  - Software Static Analysis
  - Debugging
  - Software Testing
  - **Software Build**
  - **Software Reverse Engineering**
  - Software Re-engineering
  - Software Traceability

2002/7/12 (c) 2001 Nil Software Corp. 9

Iどのようなツールが必要か:ECMA-TR69 垂直型, プロセス工学

- ・Life-Cycle Process Engineering Services
  - Process Definition
  - Process Library
  - Process Exchange
  - Process Usage



2002/7/12 (c) 2001 Nil Software Corp. 10

TR-69のその他のサービス(水平型)

- Technical Management Services
  - Configuration Management Service
  - Change Management Service
  - Information Management Service
  - Reuse Management Service
  - Metrics Service
- Project Management Services
  - Planning Service
  - Estimation Service
  - Risk Analysis Service
  - Tracking Service

2002/7/12 (c) 2001 Nil Software Corp. 11

II. オブジェクト指向におけるプロセスの特徴

- 繰り返し型である.
  - 「仕様がなければ設計できない.」と「全ての仕様がなければ設計できない」の違い.
- 再利用の促進を意図している.
  - OPENにおけるプロセスモデル
- アーキテクチャへのFocus.

2002/7/12 (c) 2001 Nil Software Corp. 12

Catalysisにおけるプロセス例

	モデル	ふるまい	データ	アーキテクチャ	フレームワーク	コンポーネント
分析	ビジネスモデル 抽象モデル（分析モデル）	ユースケース システムコンテキスト	型モデル	テクニカルアーキテクチャ アプリケーションアーキテクチャ	モデルフレームワーク	
仕様定義	仕様モデル	アクション仕様	仕様型と型仕様			コンポーネント仕様
設計	設計モデル	操作	設計型		デザインパターン イディオムの利用	コンポーネント（実装）

2002/7/12 (c) 2001 Nil Software Corp. 13

UMLとツール／プロセス

語られない「小さな物語」集

個々人がどう解決するかを考えるため.

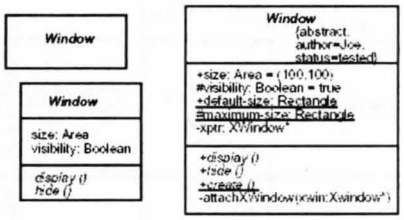
UMLの欠如を補うのではない.

既に，UMLは相手に出来ないほど大きい

2002/7/12 (c) 2001 Nil Software Corp. 14

UMLとツール #1

- 見せ方に関する配慮がある
  - Presentation Options
  - 1.4における仕様上の例
    - クラス名
    - 分析段階
    - 設計段階



- さて, リファクタリングは.

UMLとツール #2

- リポジトリ・交換
  - XMI = Yet Another CDIF ?
  - コミュニケーションメディアはどうあるべきか.
- FAQ 0.「コード生成はできますか?」
  - 何のために?
  - モデル=プログラムのモデル?

UMLとツール #3

◆ 正確な記述

- OCLのサポート
  - (公理的)集合論に基づく

```
context Guest invariant
    age ≧ 18
```

- 反応系の記述が(統一的に)できない.
  - modeling theory

$$AG[\forall z \in \mathbf{act}(\text{Guest } \mathbf{ref}) : z.age \geqq 18]$$

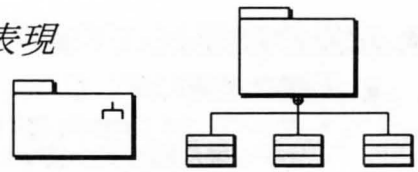
  - RTTL

$$\Theta \wedge (x_2 = X_{22}) \wedge (t = 0) \rightarrow \Diamond (x_1 = X_{12} \wedge t \leqq 4)$$

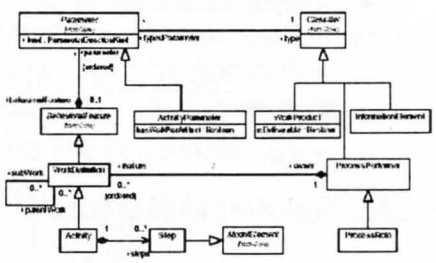
UMLとツール #4

◆ 大規模システムの表現

- パッケージ
- サブシステム

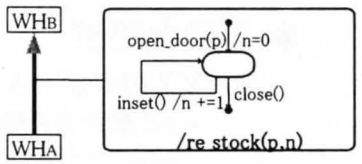


- モデル間の関係
  - SPEM 依存性



UMLとプロセス#1

◈ 洗練過程の表現



◈ 過程毎のモデルの保持(水平)
- 仕様, 分析, 設計, 実装の各モデル
- ラウンドトリップサポート

◈ コンポーネント記述(垂直)
- 提供するインターフェイス, 必要なインターフェイス
- 外部仕様
- 実行コード
- 確認コード
- 設計等

UMLとプロセス #2

◈ 成果物表現としての脆弱性
- 正確な表現に関して
  - 図とOCL (cf. Catalysis 型仕様)
  - モデル間の関係(クラス図中のOCLでの状態表現)
- 大規模・ある特性・あるドメインの表現が難しい.
  - Webアプリケーション開発プロセスに相応しいUML表現はいらない?
  - ステレオタイプ?
- 抽象表現のなさ(OOの呪縛)
  - e.g. ジョイントアクション, DOA

⇒ 図とドキュメントは異なるという当たり前のことが通用しない

まとめ

- よくあること(特にドキュメンテーションを考えると)
  - 見てよく分からない. 図だけでよしとする(単純な思考).
    オブジェクト指向 → 文書化 → UML
  - 文書が書かれないと云う古くて新しい問題
    - OOは多様な書き方を許容する. 従って, 特に意図を書くことの重要性は増している.
  - ただ一つのクラス図しか書かない.
    - 設計の洗練過程に対する希薄な意識.
    - 実装と同一レベル.
  - チェックする側の遠慮(どうせ分からない)
  - 逆生成で良しとする.
- 結局のところ,「大きな物語」はない.「小さい物語」を自分なりに紡いでいくしかない.

2002/7/12 (c) 2001 Nil Software Corp. 21

参考文献

- OMG, Software Process Engineering Metamodel Specification, Dec. 2001.
- OMG, Unified Modeling Language ver 1.4, 2001-09-67.
- ECMA, Reference Model for Project Support Environments, TR-069, Dec.1994.

2002/7/12 (c) 2001 Nil Software Corp. 22

システム分析設計とUML

コンサルタントの立場から
酒匂寛
sakoh@d2.to

UML現場への浸透度

- 「オブジェクト指向を使う」と標榜するプロジェクトでは多く使われている
  - クラス図が中心
    - クラスの仕様よりも構造に偏る傾向
    - スケルトン生成は一部盛ん
  - イベントトレース図を詳細設計代わりに使う傾向
    - 複雑なシナリオをプログラムレベルで記述しようとして破綻する傾向もあり
  - 状態遷移図は更に少なくなり、ともするとフローチャート風の利用
    - オブジェクトモデルとの対応関係が曖昧な傾向
- 今のところ Rational Rose が中心
  - 場合によっては Visio（単に図を描きたいだけ）
  - 安価な商用ツール（Structure Builder, Nirvana など）も登場
  - IIOS や Argo などのフリーソフト
  - UML Memo（大型ツールへのアンチテーゼ？）

UMLの使用上の課題

- オブジェクト指向アーキテクチャへの理解
  - 責務を負った実行主体の協調関係への理解が必要
    - OJT等を通した、感覚的な理解が不可欠
  - あたかもモジュール構造図のようなクラス図(static メソッドを呼び出していく)も多い
- 厳密なモデル作成への意思
  - 曖昧さは自殺行為あるいは二度手間
  - 正しいモデル記述の理解
    - 事前条件、事後条件
    - 限定子
    - 可視性
- 文書化との補完関係の認識
  - モデルが十分に語っていることは自然言語では書かない
  - ツールを活用、必要に応じて作成

UML は道具である

- 様々な用途に利用できる？
  - アーキテクチャ説明
  - デザインパターン説明
  - 設計レビュー資料作成
  - 分析レビュー資料作成
  - 文書化支援
  - コード生成
  - Etc
- しかしながら
  - 理解のための補助という役割を超えて、具体的な構築のための道具として考えた場合、必ずしも最適ではない
  - 複雑な構造を動的な振る舞いを含めて完全に記述すると、理解が難しくなる
  - 複雑な部分に関しては何らかの厳密な記述言語を併用したほうが良い
  - 記述要素、記述規則をプロジェクトの合意事項として確認する必要がある
- モデルの品質を上げるための Quick Hint
  - 状態(時間)推移に従って成立する条件を常に考慮
    - 「時間」をどのように定義するか
    - 「条件」をどのように取り出すか

事例1：UML vs Executable Model

- あるプロジェクトでフレームワークの設計を実施
  - 筆者は設計を担当、別の担当者が C++ で実装
  - 当初 UML で記述したが、細かいニュアンスが伝わり難かった
  - 擬似コードを書くという選択肢もあったが、Rose の中で記述しても厳密なチェックは難しい
- “仕様・設計記述”として Ruby を利用
  - 高生産性（フレームワーク記述に1日、C++ によるリライトに5日）… 個人的な感覚では java の5倍以上、C++ の10倍以上の生産性
  - もし UML（ここではRose）で記述していたら（モデリング、実装）それぞれに2倍以上の工数がかかり、手戻りも多く発生した筈である
  - 曖昧さの少ない仕様・設計伝達、テストケースの共有
- フレームワーク完成後、説明資料として UML クラス図を作成
  - スーパークラスとパブリックインターフェイスの説明だけではイメージがわきにくい？
  - 理解促進に貢献（ホットスポットのイメージ理解等）

事例2: UML and Reuse

- ある組み込みプロジェクトにてシリーズ展開のためのプラットフォームを設計
  - 記述には UML も用いているが、それを補助するための「パッケージ説明書」を用意
    - 単に「対象がどのようなものか」だけでは不足
  - パッケージ説明書の構成
    - 外部仕様
      - 自然言語ドキュメント、パッケージ API
    - 内部仕様
      - 自然言語ドキュメント、クラスモデル、状態遷移モデル
    - カスタマイズ仕様
      - カスタマイズ項目一覧
        - カスタマイズのためのホットスポットと、ホットスポットごとの解析法、設計トレードオフなどの情報
      - メトリクス情報
        - カスタマイズ項目に対応する製品系列ごとの定量データを蓄積
- こうした努力により
  - 見積もり精度、再利用効率、信頼性が向上

「オブジェクト的」であることの苦悩

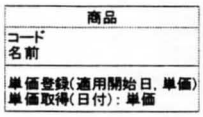
- 言わずもがなのことであるが、分析と設計の目的は違う筈である
  - 分析:システムが外界に提供するサービスを定義
    - Validate（妥当性の検証）にいかに注力できるか
  - 設計:上記で定義されたサービスの実現機構を定義
    - Verify（正当性の検証）をいかに注力できるか
- しかし分析～設計～実装に一貫してUMLを適用するという標語の下、設計に侵食された分析、分析に抑圧された設計を多く見かけるようになった
  - Boundary, Control, Entity ステレオタイプへの分類は便利だが、あくまでも設計に対する記述である
  - 分析モデルとして無理にオブジェクトへ属性と操作を配置すると結局設計時に歪が出て再考する必要が出る場合がある

例題:ある要求記述

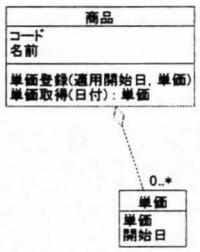
- 複数の種類の商品があり、それぞれの商品はコードと名称を持っている。また商品単価とも関連しているが、ときどき単価は改訂される。ある商品の単価を知りたいときは日付を指定して単価を得るものとする（ある商品に着目した場合、ある特定の日にはただ一つの単価しか存在しないものとする）。

例題:分析1

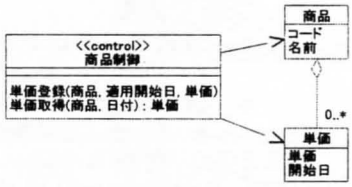
Plan A



Plan B



Plan C



どのモデルが優れているかという議論になりやすいが、仕様として考えた場合、本質はオブジェクト上への操作属性の配置の違いではない。

Plan B, C のようなモデルが教科書に載っている場合が多いが ...

例題:分析2

結局システムを外から観察するものの関心は？

観察される構造

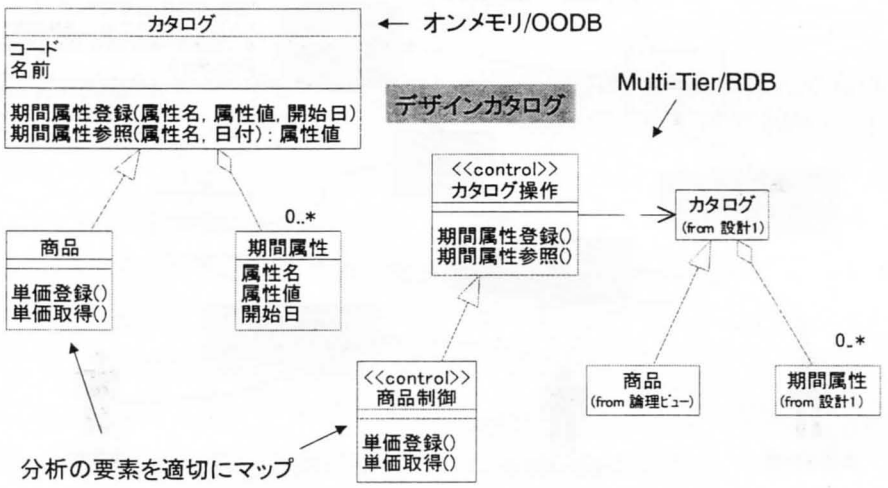
商品
コード 名前

提供されるサービス

単価登録(商品, 適用開始日, 単価)

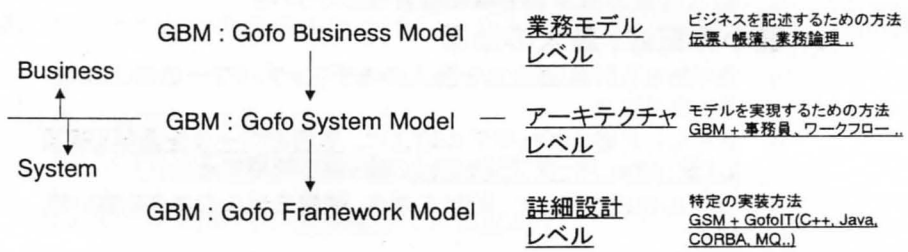
単価取得(商品, 日付) : 単価

例題:設計

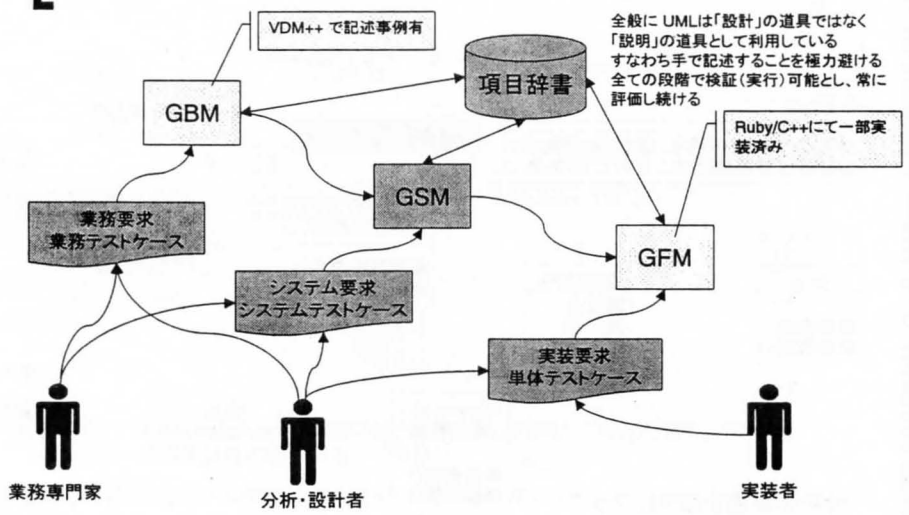


例: GSD (gofo system development)

事務処理に特化したモデリング・開発手法



参考:GSD の開発プロセス



番外:個人の「モデリング能力」の拡張

- 脱「紺屋の白袴」
  - Excel 万能主義への疑問
  - 個人作業の成果物も構成版管理を行うべき
- 最小の記述、最大の効果
  - ありあまる計算機能力を個人のモデリングパワーの向上に使うべきでは?
  - きちんと定義されたモデルの上に、専用のツールを素早く構築し(昔:pipe, 今:オブジェクト)、繰り返し利用する
  - ツール作成を通して、設計のネタ、情報モデルのネタを拾い続ける
- 折角構築したモデルは骨まで利用しつくすべき
  - 情報をコピーするものに災いあれ

SEA 特別ワークショップ「UML を徹底的に分析評価する」報告

熊谷章（PFU）

参加者募集で説明しているようにこのワークショップの狙いと効果を次のように設定した。

狙い：UML（Unified Modeling Language）が何者で、どこから来て、どこへ行こうとしているかを下記の観点から理解する

コンピュータサイエンス
　プログラミング言語、モデル、フォーマルメソッド
ソフトウェアエンジニアリング
　モデル、開発方法論、ツール、プロセス

効果：UML を理解し、仕事に有効に使用できるようになること
UML とはなにか
種々の方法論と UML の関係
どのような分野にどのように使えば有効か
システム開発のライフサイクル上での UML が果たす役割
UML の過大評価と過小評価
UML に欠けているもの（機能、表現、考え方など）

プログラムに従った発表者の説明とそれに関する議論と二つのパネル討論を通して、上記の狙いと効果に対して、どのような解が得られたかをコーディネータの立場から、まとめて報告することを試みる。

先ず、最初にプログラムの構成を眺めてみることにする。

UML の解説（羽生田栄一：豆蔵）
モデルと UML（玉井哲雄：東京大学）
UML の大学教育（中谷多哉子：S-lagoon）
UML の実践教育（桜井麻里：TIPS）
プログラマからみた UML（東田雅宏：PFU）
UML の意義と将来性（パネル討論：発表者全員）
UML とあなたはどうかかわるべきか（BOF）
OMG における UML の位置付け（伊藤友里恵：OMG Japan）

UMLを用いたシステム開発事例（石橋民男：JFITS）
組込み用にアレンジしたeUMLの狙い（渡辺政彦：CATS、eUML）
開発ツール／プロセスとUML（伊藤昌夫：ニルソフトウェア）
システム分析設計とUML（酒匂寛：Designer's DEN）
UMLを有効に使うために（パネル討論：発表者全員）

第一日目は技術を中心としたプログラム構成にした。羽生田さんには、このワークショップのキーノート的位置付けとしてUMLの肝と心を解説して貰うことにした。参加者全員がUMLに関する知識を共有し、誤解を無くすためには重要だと考えた。次いで、玉井さんからは、コンピュータサイエンスとソフトウェアエンジニアリングの視点で学から見た分析と評価をお願いした。次は、UMLに関する大学における教育と企業人の教育の実態を中谷さんと桜井さんから話して貰った。これで、UML教育を実践するための条件や法帆が分かると考えたのである。次は、オブジェクト指向プログラマから見たUMLの評価を東田さんに話して貰った。実際のプログラマにとってUMLは何の助けも役にも立たないという主張をよく聞くからだ。

パネル討論とBOFをプログラムに入れたのは、討論によらなければ明らかにならない物事がたくさんあると考えたからである。そして、より砕けた内容の議論を可能にするためにBOFセッションも設けた。

二日目は、UMLの応用をテーマにした積りである。伊藤さんには、OMGがなぜUMLを担いで普及させているのかをOMGの他の活動と連携させて説明して貰った。石橋さんからは、UMLを実際に適用したプロジェクトの結果とその評価を話して貰った。渡辺さんは、eUMLコンソーシアムの設立者であり、かつ組み込みようのソフトウェア開発の実際とUMLに対する期待などを話して貰った。伊藤さんには、ソフトウェアエンジニアの専門家として開発ツールとプロセスという視点からのUMLの評価をお願いした。酒匂さんには、多くのコンサルテーションの経験からシステムの分析／設計をする時にUMLがどんな役割を果たせるかを話して貰った。

最後のパネル討論は、このワークショップのまとめとして計画した。

各セッションの感想と寸評を記憶に従って述べると次のようになる。

1. UMLの解説 －その肝と心－（羽生田栄一：豆蔵）

羽生田さんの「UML解説」は出色の出来だった。UMLもキモとココロを分かり易く説明して戴きたいというのが主旨であったが、期待に違わずピタリとその筋道を示してくれた。

世の中にたくさんのUML解説があるだろうが、UMLのセマンテックスに言及し、あそこまで踏み入って解説を聞いたことがない。そして、UMLのセマンテックスをOCLなどのフォーマル言語では記述不可能なので、自然言語（English）でしかよく記述できないのが現状である、という辺りがUMLの現状を正確に表現している、と思われた。以下に、彼の発表資料に沿って、記憶に残っている情報とコメントや質問をいくつか列挙する。

始めに、主なUMLユーザを紹介してくれた。ソフトウェア業界以外の業界、例えば地図やビジネスの業界で大雑把なシステム記述が必要とされるところとJavaを使用している分野でプログラムより冗長度の高い記述が必要なところで多く使用されているという。このことは、UMLはシステム分析とシステム設計に用いられている訳ではなく、システムを表現し、その意味を人に伝達する手段として多く利用されていることを証明しているようにみえる。

なぜ、UMLはデファクトスタンダードになれたかという説明の中で、注意を引いた点は「厳密さを犠牲にしたセマンテックス規定」がある。厳密さの最低の基準として、Javaでシステム開発するときに必要とされる最低限のものを採用したらしい。ここらが、UMLのキモとココロである。UMLの各モデルが、マシンリーダブルでなく、テキストに変換不可能なことは、モデルとしてはある基準から考えれば致命的である。つまり、UMLで作成したモデルのシンタックスとセマンテックスをコンピュータで自動的にチェック出来ないからである。書かれたモデルは、人が解読して始めて正当性や他のモデルとの関係などが理解できるのである。

また、逆にビジュアルなダイアグラムでモデル表記することが、人の視覚と直感に訴え分かり易いので、これをベースに異なった職業の人々が理解し合え、かつ議論し合えるということなのか。オブジェクト指向技術が現れ、プログラミングレベルでオブジェクトモデルが話題になったときに、同じようなことが言われた。異なった職業をまたがって共通に表現できるオブジェクトモデルが発明されたので、様々な分野の人々があるものを議論するとき始めてオブジェクトモデルを使用してまともな儀論が出来るようになったのである。しかし、オブジェクト指向プログラミングは厳密さ保たれていたが、UMLダイアグラムでは厳密さが保たれていない。従って、UMLの狙いは明らかに厳密さを欠いた冗長度が高く、大雑把な表現を狙ったものと言わざるを得ない。果たして、そんなもモデルに意味があるのかと真剣に問うてみたくなる。

次は各UMLダイアグラムの説明である。UMLでの新規発案はなく、従来の開発技法からイイとこ取りをしたのであまり特徴はない。その中で一番揉めたのがアクティビティ図だという。これは、データフローや業務フローに酷似しているのですんなりと決まらなかっ

たらしい。良く見れば、アクティビティ図はステートチャートのサブクラスであることが分かる。

UML の段階的理解はとても分かりやすい説明であった。モデル要素、基礎語彙、関係、意味規則、共通メカニズム、構造図、振舞い図、実装図、アーキテクチャと４＋１ビューは、UML の全体を理解するための必要でかつ充分な素養を与えてくれる。

次の目次、UML セマンテックス入門が珠玉である。先にビジュアルなダイアグラムで冗長度が高い大雑把な表現が UML のキモとココロかと述べたが、このセマンテックスの説明を聞いて、理解し、UML の本当のキモとココロがここに置かれているだろうと感じた。その理由を、標準拡張機能、OCL、UML メタモデル、UML 抽象構文、UML 適性規格と追加操作、UML 意味論記述例などにみることができる。現時点では、UML は言語になっていないが、将来キチンとしたセマンテックスを持ったマシンリーダブルなモデルを目指していることが、UML セマンテックス入門の説明から窺い知ることができる。それが、初期の目的を達成し成功するかは、当然のことながら別の儀論であるのだが。

羽入田氏のまとめが的を射ている。今回の説明では、UML 記法ガイドだけでなく、UML 意味論の本文が読める最低限の知識を紹介した。UML 自体は言語として未成熟である。現状の UML1.4 から UML2.0 で変化するだろう。UML 利用は、目的に応じて必要な部分だけを利用すればよい。キメの細かい仕様や設計の指定には、OCL も利用すればよい、と言う。

2. モデルと UML（玉井哲雄：東京大学）

玉井さんの発表は、モデルとはなにか、という基本的な定義から始まり、図式を用いたモデルによる記述を解説した。この手の説明は、お目に掛かることが少ない。表現に共通に使用されているグラフ構造を説明し、そのようなグラフ表現はソフトウェアに限らず一般的に使用されていると説明する。それアプローチは、人間の知覚能力や記号論を連想させるもので、興味深い。更に、論を進め、グラフによる静的モデルと動的モデル表現、グラフによるモデルの解釈に至り、例として Kirchhoff's Law を挙げ説明する。この下りは、UML だけを念頭に置いていると発想できない。

グラフ表現の一般的性質を分析し説明した後で、UML の位置付けを次のように説明している。それは、オブジェクト指向モデルの記述言語で、これまでの分析と設計ノウハウを図中心に集大成したカタログである。このように UML をまとめてみた後で、次の三視点からさらなる分析と説明を加える。

分析設計ノウハウとモデル表現のカタログ

グラフとしての特性に注意
モデル図間の差異の明確化

なるほど、上で挙げた三視点からＵＭＬ図を分析した例は見たことが無い。面白いし新たな気付きを体験できる。例えば､クラス図が、ＥＲ図や意味ネットワークに似ているということを普通は気付かないだろう。また、クラス図の辺が表している静的関係がオブジェクト指向のもの作りを難しくしているのではないかという指摘は今後検討に値するだろう。その静的関係とは、汎化（generalization）、関連（association）、集約（aggregation）、依存（dependency）を指している。

UML についての補足的なコメントに含蓄がある。UML は、実世界の分析モデルよりシステム設計モデルにシフトしていること、テキスト表現の扱い、例えば OCL であるが、これを今後どのように活用するかが UML のキーポイントになると予測している。

最後に UML の各図の差異を明示的に意識し使用することが大切であるとしめている。

3. UML の大学教育（中谷多哉子：S-lagoon）

中谷さんは国内の４大学でオブジェクト指向とUMLを学生に教えている経験から大学教育と UML の関係を話してくれた。

大学で使用しているシラバスは次に内容である。

オブジェクト指向概説
オブジェクト指向ソフトウェア開発プロセス概説
ユースケース、オブジェクトモデル
設計とプログラミング規約
プログラム構築
デザインパターン
再構築

この内容で講義は、一回が 1.5 時間で 12 回行っているそうだ。学生の理解に関する報告が面白い。情報隠蔽の概念を Java で理解することは早くできるが、UML とプログラミングを結びつけて理解することが困難だという。ただし、UML を理解し、モデルを書くことはできるという。このことは、何を意味しているのだろうか。

翻って、オブジェクト指向と UML を企業内教育で行うとどうなるかという説明があった。内容は、オブジェクト指向概要、UML、開発方法論、演習とレビューで、7 時間を三日間

やるのだそうだ。習得の鍵は、プログラミング経験に基づく裏付けにあるらしい。

4．UMLの実践教育（桜井麻里：TIPS）

桜井さんは企業の人々がどのようなUML教育を受けているかを説明してくれた。彼女自身豆蔵でUMLセミナーの講師をしているので、その経験紹介してくれた。

セミナーは、UMLによるモデリング入門が一日で、UMLを用いたオブジェクト指向分析と設計が三日間である。教えている内容は、UMLの表記法、開発プロセスと開発方法論で、RUPやXPについても言及する。ただし、UMLのメタモデルとOCLは教えない。なぜなら、その必要性を感じていないことと時間が取れないからだという。

企業内教育の課題は、三日間位のセミナーで全部分かるわけがないこと、現場に戻ってもUMLを使用できるような環境と文化が必要なことである。演習には、Rational Roseを使用しているという。

5．プログラマからみたUML（東田雅宏：PFU）

東田さんは、Smalltalkerプログラマとしてユニークな発表をしてくれた。

プログラマは、プログラミング言語で思考し、ツールで動き、作成したプログラムを動かすことに意義を感じて、所期の目的を達成するまでプログラミングを続ける。従って、この活動の中で、UMLが現れることはないという。分析とか設計のために、文書を書くことは二度手間であり、中途半端な活動だと断言する。

この観点から、UMLには言語の意味が無く、ツールが無く、UMLで書いたモデルは動かない。また、UMLで書いたモデルが動いたとしてもそれが嬉しいことか。これだけ、何もないとUMLの意味は、文書としてしかないだろう。しかし、UMLは盛り沢山過ぎて、難しい上に体得する手段があまりない。だから、出来上がったプログラムの一部を自動的にUMLにマッピングして、仕様書や設計書とすることが良いかも知れない。

オブジェクト指向プログラミングでは、創造活動はクラス設計であり、デザインパターンであり、アプリケーションフレームワークを作り出すことである。UMLがもの作りの道具として役立つようになることを期待したい。

6．UMLの意義と将来性（パネル討論：発表者全員）

パネル討論は、発表者への質疑応答で終始した。特筆すべき内容はなかった。

7．UML とあなたはどうかかわるべきか（BOF）

山崎さんから次のような主張と提案があったように覚えている。

UML は、Booch 法、OMT、OOSE と OOAD から生まれた。それは、対象システムの仕様とインプリメントの両方を表わす。これらの考え方の、基本は「Objects + Message Passing」である。この方法の問題に、Subtyping が挙げられる。これは、実行中にタイプが変化するもので、Late Binding で対応されている。

いま、あるシステムを UML で書いたものを Xmとする。このXmを構成しているダイアグラムをD1、D2、、、、D10 とする。各 Di に対して任意なインプリメンテーションがある。Di に対する 2 つのインプリメンテーションを、I1、I2、、、、I10 と、I'1、I'2、、、、I'10 とする。一つの UML 記述を元に二つのシステムが作られたことになる。この二つのシステムはどういう関係にあるのか。これを解明するために、Precise Semantics が必要である。

対象システムXを仕様記述したものをXsとし、Xsをインプリメントしたものを Xi とする。X—Xs 間で Validation が、Xs－Xi 間で Verify が可能になる。従来の UML ユーザからは隠した形でこの仕掛けを作ることに意味があると考えている。一般的に、フォーマルメソッドは、動的な記述ができないという観点から忌み嫌われているが、X から Xs への仕様化において、下記の問題を解決できるはずである。

Safety、Liveness、Failure、Dead lock free、Divergence

会場から同意の意見が出され、このようなアプローチを実施している方がいるのかという質問があった。フィンランドの Back が Usecase からステップワイズ方式でシステムをチャント作ろうと提案している、とい説明があった。

8．OMG における UML の位置付け（伊藤友里恵：OMG Japan）

伊藤さんには、OMG と UML の関係と将来の展望をお聞きしたかった。他の発表者が技術屋と科学屋だったので、政治的で政策的な話が新鮮に思えたのが不思議に思えた。OMG の考え方、やり方、狙い、日本の状況などをよく知ることができた。

伊藤さんの発表は洗練されており、過不足のない表現と発表時間を厳格に守ることなどからプレゼンテーションのプロフェッショナルだと感心した覚えがある。彼女の働きかけでソフトウェア技術者協会は OMG のリエゾンの一つとなった。

9．UML を用いたシステム開発事例（石橋民男：JFITS）

石橋さんは、新しい証券機関システム（Tradeone）の開発に、UMLとフォーマルメソッドを使ったという興味深い話を提供してくれた。実体験に基いた話で、かつプロジェクトメンバーのほとんどがオブジェクト指向技術とフォーマルメソッド技術の使用経験がなかったという前提条件を明らかにしてから、話し始めたので聴衆は話題に乗って来た。

設計／開発方法にRUP、設計表記法にUML,開発言語にC++とJavaなど、ソースコードサイズは約1.9Mega、開発期間は26ヶ月、ロジックが明確でない部分にフォーマルメソッドVDMを適用した、という。Toy Projectでない本物のシステム開発である事が分かる。

UMLに関連する部分をピックアップすれば、UML教育に一人当たり20時間費やしたこと、UMLを用いて設計の有功度合いと実装~テストの実現可能性を検証したこと、オブジェクト指向とUML採用によりソフトウェアの生産性が1／2に減少したこと、などがあった。

ＵＭＬをこのプロジェクトに適用した結果の評価が意味深い。いくつかの問題が発生したものの、結果的にはプロジェクトの成功に大きく寄与できたと考えているのだ。つまり、仕様記述が統一でき、サブシステム間の不整合などによる手戻りが抑えられ、実装以降のフェーズが非常に効率的に実施できたらしい。一方、UML技術者の育成には問題が起きた。汎用マシンの経験者は、設計を実装レベルまでにブレークダウンすることは困難なので、仕様確認レベルまでの体得に止まったことである。

この発表に対しては、質問が相次いで出された。汎用マシンのCOBOL技術者が、新しいシステム開発にオブジェクト指向を基本技術として採用し、かつ分析／設計にUMLを使い、プロトタイプを使い、部分的にフォーマルメソッドVDMも使ったのだから皆さんの注目を引いたのは頷ける。石橋さんのレジュメも発表の分かり安くとても良かった。

１０．組込み用にアレンジしたeUMLの狙い（渡辺政彦：CATS、eUML）

ソフトウェア技術者は全国で53万人いるらしいが、その中で組み込み関係が20万人いる。この人々はいままで下請け的色彩が強い仕事を黙々とやってきて脚光を浴びたことが少なかった。しかし、いまや、組込みソフトウェアが社会的に認知され、その重要性が叫ばれるようになった。この状況を喜ぶとともにeUMLというコンソーシアムを作り新しい挑戦を始めた。このような熱い思いをこのワークショップで語ってみたい。開口一番、渡辺さんの主張は面白く頭に心地よかった。

UMLが何者で、何処から来て、何処へ行こうとしているかは次の視点から理解できるという主張が的を得ている。

項目	eUML
コンピュータサイエンス：プログラミング言語	アクションシンタックス
モデル	システムモデル
フォーマルメソッド	詳細化証明
ソフトウェアエンジニアリング：モデル	問題発見
開発方法論	ガイドライン
ツール	検証
プロセス	コミュニケーション

この表の中で、フォーマルメソッドの対する説明である詳細化証明だけが考慮中であるらしい。筆者には、ソフトウェアエンジニアリングのモデルが問題発見であることやツールが検証であるという認識が頭を揺さぶってくれた。

要求仕様の説明で、分析した結果のモデルをアニメーションで、設計した結果のモデルをシミュレーションで、実装した結果のプログラムをエミュレーションで、試験するという。オブジェクト指向のシステム開発に酷似していて納得が出来た。

このワークショップの効果を次のようにまとめてくれた。

項目	eUML
UML とはなにか	表記法
種々の方法論と UML の関係	コミュニケーション
どのような分野に有効か	演繹的アプローチ
システムライフサイクルでの UML の果たす役割	コミュニケーション
UML の過大評価と過小評価	コミュニケーション
UML に欠けているもの	テスト、STM、タイミングチャート

最後に eUML の構想と今後の見通しについて話してくれた。RUP よりライトウェイトで、XP よりモデルを重視した組込み用にアレンジした eUML を作り上げたいという。成功を祈らずにおられない。渡辺さんの熱演に改めて感謝する次第である。

１１．開発ツール／プロセスと UML（伊藤昌夫：ニルソフトウェア）

伊藤さんは、彼独特の語り調で、「大きな物語の終焉」について話してくれた。リオタールの言葉を引用し、淡々と語るその容姿は伊藤さんの世界である。

「誰一人としてこれらすべての言語を語るものはなく、これらの言語は普遍的なメタ言語をもたず、体系＝主体の計画は挫折に終わり、人々はそれぞれ個別的認識の実証主義のうちに沈み込み、学者は科学者になり、研究作業はもはや誰も統轄することができないほど細分化された作業となってしまっている」

ツールとプロセスという観点から、かつてのそして今でも生きている「大きな物語」が終焉していることを次の例から主張する。

ツール（1990 年前後）：生産性と効率を数倍あげる CASE ツール
プロセス（1970 からいま）：使われない開発標準, ライフサイクルモデル、
CMM などの評価モデル
UML（いま）：これさえ知っていれば、、、、

伊藤さんの指摘は直感的で分かり易い。しかし、彼の説明は会場の人々からは難しすぎてよく分からないという。このギャップが彼の魅力であり, 我々聞く側の問題であろう。伊藤さんの警鐘は社会全体に対するものとして聞くに値すると考えた人は多いはずだ。

１２．システム分析設計と UML（酒匂寛：Designer's DEN）

酒匂さんは, システム開発のコンサルタントの視点から UML に関する考察を述べてくれた。例題を多く示した説明は、わかりやすく示唆的であった。特にシステム開発の全体的視野から UML を評価しており、UML の相対的な位置付けがわかりよく理解できた。

最後に酒匂さんが提案した GSD（gofo sysytem development）は、事務処理に特化したモデリング・開発手法で今後が期待される。それは、業務モデルレベル, アーキテクチャレベル、詳細設計レベルの三層で構成される開発方法である。

１３．UML を有効に使うために（パネル討論：発表者全員）

最後のパネルは皆さん少々くたびれ気味であったが、各参加者から感想と意見を聞いた。多くの質問も出て中々面白かった。その詳細は触れないことにする。

終わりに

最後にコーディネータがこのワークショップを通して得た事をワークショップをやる前に設定した狙いと効果に照らし合わせて考察してみたい。

（1）UML は何者か

狙い：UML（Unified Modeling Language）が何者で、どこから来て、どこへ行こうとし

ているかを下記の観点から理解する

コンピュータサイエンス
プログラミング言語、モデル、フォーマルメソッド
ソフトウェアエンジニアリング
モデル、開発方法論、ツール、プロセス

この狙いはほぼ達成されたと考えている。少し、観てみよう。

UML はコンピュータサイエンスとソフトウェアエンジニアリングの知識領域から考えれば、プログラミング言語ではないし、ある特定のモデルでもないし、フォーマルメソッドでもないし、開発方法論でもなく、ましてツールやプロセスをカバーするものでもない。では、UML は何なのかといえば、グラフとダイアグラムであるものを表記するカタログを示しているに過ぎない。

怪我の功名かも知れないがRUP とBooch 法とかいう開発方法をUML の範疇から放棄したことがUML の普及に大きく貢献したと思われる。単なる表記法があるだけで、それをどう使うかは使用者の自由である、といういい加減差が、いい加減な人類には向いているのだろう。つまり、いま仕事をやっている人々にはある種の決まりややり方が定着しており、それに基づいて毎日の生活が運んでいる。これを新しい方法に変革することは至難の業である。ところが、従来の考え方とやり方を踏襲しながら、表現法だけをUML に変えることは簡単である。セマンチックスを変えないで、シンタックスだけを合わせるのである。

このような観点から、UML はいままでソフトウェア技術を利用していなかった分野で重宝がられているようだ。システム LSI 開発分野、組込みソフトウェア分野、ビジネスモデル分野などがその具体例として挙げる事ができる。彼らは従来の表現方法、日本語、英吾、その分野に特化した表やグラフなど、数式、プログラムなどを捨てて、共通表現形式であるUML を採用し始めた。そのメリットは、変化がパワーの源になることと、様々な専門家たちが共通の表現を使って意思疎通が可能になることのように思える。そういう意味ではなんでも良かったのかも知れない。そこで、便宜的に皆さんが慣れ親しんでいるダイアグラムとグラフが選ばれたと見える。

UML の源は、一応オブジェクト指向とメタモデルと思われるが、これはあまり真剣に考えられているとは思われない。その証左として筆者が考えるたことを記す。もしメタモデルがUML の肝と心ならば、メタモデルの仕掛けを実装したものをOMG なりUML グループが社会に出すのが当然である。メタモデルとは、それを使用するユーザがカスタマイズで

きる機能をメタモデルに注入すれば、所期のものが得られるというメカニズムを指す。こえさえあれば、モデルエディターという観点からすれば、必要十分条件を満たす。高額なモデルエディターソフトに対価を払う必要がなく、皆幸せになれる。

UMLは、今後UML2.0と変化し続けて行くことだろう。そして、その行き先は、しっかりしたプログラミング言語の一つにしたいという意志が感じられる。羽生田さんが力を入れて説明してくれたUMLセマンテックスはそのための下敷きであるに違いない。また、OCLでは不十分なので、もっと厳格なAction Semanticsなどの導入が検討されているようだ。これで動的な記述や厳格な記述を可能にし、マシンリーダブルにしたいという意図が窺われる。しかし、そこに到達することは至難の技にみえる。そういう意味で、UMLが何処へ行こうとしているかは、予断を許さない状況にあると思われる。

（2）どうすればUMLを有効に使用できるか

効果：UMLを理解し、仕事に有効に使用できるようになること

UMLとはなにか
種々の方法論とUMLの関係
どのような分野にどのように使えば有効か
システム開発のライフサイクル上でのUMLが果たす役割
UMLの過大評価と過小評価
UMLに欠けているもの（機能、表現、考え方など）

（1）でUMLとは何かが分かったので、ここではどうすればUMLを仕事に有効に使えるかという視点から考えてみたい。

まず、種々の開発方法論とUMLの関係であるが、それらの間には論理的な関係はない。都合の良いところで、都合のよいように使えばよいだけである。この縁が極めて薄くないところがUMLの最大の特長であり、評価できる点であるらしい。

どのような分野にどのように使用すれば有効か、に対する答えは羽生田さん、渡辺さんと酒匂さんお発表にありました。それは、厳格でない大雑把なシステム記述を必要とするところやプログラムより冗長度が高い記述が必要な分野である。もっと一般的に言えば、ソフトウェアエンジニアというよりシステムズエンジニアが対象システムを分析し、設計し、それらの結果を他の人々に説明するときに、対象システムを表現する手段としてUMLを使えばよいのではないか、という程度のことを表明していると考えられえる。

そう考えれば、システムのライフサイクルモデルとは直接関係はないと思われる。

UML の過大評価の典型を OMG の MDA（Model Driven Architecture）に見ることができる。MDA の考えによれば、UML で対象システムの全部を記述するという。これは、PIM（Platform Independent Model）である。つまり、書かれたモデルが最終的に動作するコンピュータや環境から独立している。そこで、ある種のトランスレータを作成し、この PIM を、対象コンピュータと環境で動作する PSM（Platform Specific Model）に変換するという。こうすれば、開発すべき対象システムをモデル記述をするだけで、その後は自動翻訳によりソースプログラムまで出来上がることになる。

過小評価の典型は、UML は開発方法もなく、言語でもなく、単なるお絵かきのカタログでしかないので、ソフトウェア開発には何の役にも立たない。却って、そんなものを使えば余分な情報や余分な労力が掛かるだけであるという主張である。これはこれで尤もらしい。

最後に、UML に欠けているものはなにかという設問であるが、この特集号をご覧になられた方には既にご理解されているはずなので、ここでは省略することにしたい。

ワークショップに参加してくれた皆さんに改めて御礼を申し上げる。また、どこかでお会いしましょう。

ソフトウェア技術者協会
〒160-0004 東京都 新宿区 四谷 3 - 12 丸正ビル 5F
Tel: 03 - 3356 - 1077 Fax: 03 - 3356 - 1072
E-mail: sea@sea.or.jp
URL: http://www.sea.jp/