



SEAMAIL

Newsletter from Software Engineers Association

Vol. 14, Number **11-12** August, 2006

目 次

第56回 SEA-SPIN Meeting	1
開催案内	1
スライド(ソフトウェア会社のトヨタ生産方式に学ぶ)	高木徹 2
スライド(NPS伝導師活動報告)	山崎勲 10
参加者アンケートのまとめ	田中一夫 15
SEA Forum May	21
開催案内	21
SEA Forum アспект指向は実用に使えるか	小林修 22
スライド(AOP入門)	増原英彦 30
スライド(あなたの知らないAOP)	山根英次 40
Forum 参加者の声	51
画像問題のアспект	玉井哲雄 53
SEA Forum Special	60
開催案内	60
SEA Forum Special を終えて	野中哲 61
スライド(形式手法：開発現場での使用経験)	佐原伸 62
スライド(フォーマルアプローチの限界はどこにあるか)	飯島淳一 69
スライド(システムリスクとフォーマルアプローチ)	山本修一郎 79
Forum Report	中村英二 91
ソフトウェア開発を支える"ディシプリン"	新谷勝利 95
編集後記	101

ソフトウェア技術者協会

Software Engineers Association

ソフトウェア技術者協会 (SEA) は、ソフトウェアハウス、コンピュータメーカ、計算センタ、エンドユーザ、大学、研究所など、それぞれ異なった環境に置かれているソフトウェア技術者または研究者が、そうした社会組織の壁を越えて、各自の経験や技術を自由に交流しあうための「場」として、1985年12月に設立されました。

その主な活動は、機関誌 SEAMAIL の発行、支部および研究分科会の運営、セミナー／ワークショップ／シンポジウムなどのイベントの開催、および内外の関係諸団体との交流です。発足当初約 200 人にすぎなかった会員数もその後増加し、現在、北は北海道から南は沖縄まで、350 余名を越えるメンバーを擁するにいたりました。法人賛助会員も 17 社を数えます。支部は、東京以外に、関西、横浜、名古屋、九州、広島、東北の各地区で設立されており、その他の地域でも設立準備をしています。分科会は、東京、関西、名古屋で、それぞれいくつかが活動しており、その他の支部でも、月例会やフォーラムが定期的に開催されています。

「現在のソフトウェア界における最大の課題は、技術移転の促進である」といわれています。これまでわが国には、そのための適切な社会的メカニズムが欠けていたように思われます。SEA は、そうした欠落を補うべく、これからますます活発な活動を展開して行きたいと考えています。いまで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために、ぜひとも、あなたのお力を貸してください。

代表幹事： 田中一夫

常任幹事： 荒木啓二郎 熊谷章 高橋光裕 玉井哲雄 中野秀男

幹事： 石川雅彦 落水浩一郎 窪田芳夫 蔵川圭 小林修 小林允 近藤康二
桜井麻里 酒匂寛 塩谷和範 篠崎直二郎 新谷勝利 新森昭宏 杉田義明
鈴木裕信 中來田秀樹 奈良隆正 野中哲 野村行憲 野呂昌満 端山毅
平尾一浩 藤野誠治 松原友夫 渡邊雄一

事務局長： 岸田孝一

会計監事： 吉村成弘 橋本勝

分科会世話人 環境分科会(SIGENV)：塩谷和範 田中慎一郎 渡邊雄一
教育分科会(SIGEDU)：君島浩 篠崎直二郎 杉田義明 米島博司 森泉清
ネットワーク分科会(SIGNET)：人見庸 松本理恵
プロセス分科会(SEA-SPIN)：伊藤昌夫 塩谷和範 新谷勝利 高橋光裕 田中一夫 端山毅 藤野誠治
フォーマルメソッド分科会(SIGFM)：荒木啓二郎 伊藤昌夫 熊谷章 佐原伸 張漢明 山崎利治
オープンソース分科会(SIGOSS)：石川雅彦 岸田孝一 杉田義明 鈴木裕信 中野秀男

支部世話人 関西支部：小林修 中野秀男 横山博司
横浜支部：野中哲 藤野見延 北條正顕
名古屋支部：石川雅彦 角谷裕司 野呂昌満
九州支部：荒木啓二郎 武田淳男 平尾一浩
広島支部：佐藤康臣 谷純一郎
東北支部：布川博士 野村行憲

賛助会員会社：ジェーエムエーシステムズ SRA PFU
オムロンソフトウェア キヤノン 新日鉄ソリューションズ
ダイキン工業 オムロン 富士電機リテイルシステムズ
リコー NTTデータ ヤマハ オープンテクノロジーズ
SRA西日本 SRA東北 エフビクス 電盛社
(以上17社)

SEAMAIL Vol. 14, No. 11-12 2006年8月25日発行 編集人 岸田孝一
発行人 ソフトウェア技術者協会 (SEA)

〒160-0004 東京都新宿区四谷3-12 丸正ビル5F

T: 03-3356-1077 F: 03-3356-1072 E-mail: sea@sea.or.jp URL: <http://www.sea.jp/>

印刷所 市田印刷株式会社 〒114-0014 東京都北区田端2-3-25

定価 1,000円 (禁無断転載)

ソフトウェア会社のトヨタ生産方式導入から学ぶ

主催

SEA プロセス分科会 (SEA-SPIN)

今回の SPIN Meeting では、「実践！！IT屋のトヨタ生産方式」という話題の本の著者を迎え、その内容をお聞きするとともに、ソフトウェア開発の現場におけるトヨタ方式の適用の実例を中心に、皆さんと今後のシステム開発の品質向上に向けた意見交換を行いたいと考えます。

昨年、「実践！！IT屋のトヨタ生産方式 あるソフトウェア会社の挑戦」という本が風媒社から出版されました。本の紹介に次のようにあります。

- ・ ソフトウェア開発に『トヨタ生産方式(TPS)』の考え方を適用し、5年後に生産性を3倍にする」という野心的な目標を掲げ、全社規模の業務改善に取り組んだ。
- ・ ここでは、中堅社員を「カイゼン塾」と呼ぶ社内トレーニングに参加させ、TPSのリーダとして育てていった。各リーダは現場の抵抗に遭い、悩みながらも創意工夫を凝らして独自のTPSを生み出した。
- ・ 事務作業の効率を上げるといった「ムダ取り」や、アジャイル開発の応用までさまざまな取り組みがなされた。

今回は、その実践の中心におられる講師の方から直接、実践の中の本質の話をベースにお聞きするとともに、今後のシステム開発の品質向上に向けた意見交換を行います。

なお、上記の本をお読みいただいた上で参加いただくことをお勧めします。また、下記のURLに関連する記事が出されていますので、これも一読をお勧めします。

「トヨタ生産方式の導入によるソフトウェア開発プロセスの革新 (PDF)」

<http://magazine.fujitsu.com/vol56-6/paper16.pdf>

開催要領

1. 日 時: 2006 年 5 月 8 日(月) 13:00 ~ 17:00
2. 会 場: (株)日立システムアンドサービス本社 20F コンファレンスホールD
3. プログラム:

コーディネータ 藤野 誠治、田中 一夫

13:00 - 13:30 受付及びネットワーキング

13:30 - 15:00 ソフトウェア会社のトヨタ生産方式導入から学ぶ
(富士通ソフトウェアテクノロジーズ 高木 徹)

15:00 - 15:15 休憩

15:15 - 15:45 伝道師の活動
(富士通ソフトウェアテクノロジーズ 山崎 勲)

15:45 - 16:45 質疑・意見交換

16:45 - 17:00 クロージング

『ソフトウェア会社のトヨタ生産方式導入から学ぶ』 ～FST全社改善活動事例に見る伝道師育成による「人づくり」～

2006年5月8日
御富士通ソフトウェアテクノロジー
経営企画室 NPS推進部
KAIZEN塾塾長 高木 徹

はじめるにあたって

- 失礼ですが、ご興味についてお聞きさせてください。(少しでも双方向にしたい。)
- トヨタ生産方式について？
- 手法論について？
- 現場の意識改革について？
- 数値化できるものは数値化する(原理・原則)

05/05/06SPIN

3

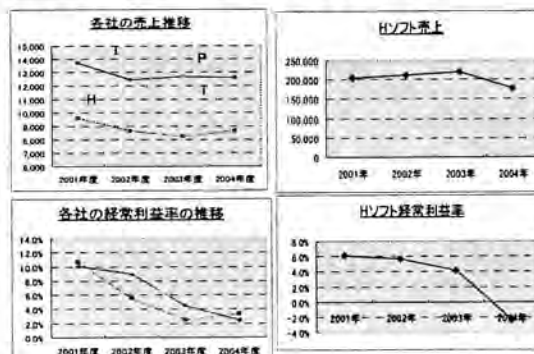
皆さんにお伝えできるか？

- 現場の活動を伝えるためにどうするか？
 - 現地・現物の原則からすると現場で説明すべき。
- 1%程度の伝達量。
 - KAIZEN塾(24回×7H=168H)
 - 今回の講演は2H
- 今回の発表向けに改善してみました。
 - 改善発表会の映像(現地・現物、視覚による情報提供)
 - 塾生からの発表(伝道師の生の声)
 - 改善とは何か？体験ツールのご紹介。

06/05/06SPIN

4

なぜ？トヨタ生産方式を取り入れたか？



環境、市場変化への対応遅れ

- 海面上
 - 価格下落(ITインフレ) ⇒ 利益率の恒常的低下
- 海面下
 - 協力会社依存による空洞化 ⇒ 協力会社管理
 - 原価差額で利益を確保する体質
 - ノウハウが貯まらない(人が育たない)
 - 加工費の増加
 - 時間外増加
 - 間接費増加

06/05/06SPIN

5

2003年末にトップの決断

従前の方式に変わる生産性を革命的に向上させる新たな方式を独自開発し、
生産性を5年で2.5倍へ！



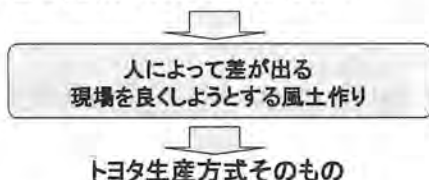
- 従来の考え方を変える(現状否定)
- あるべき姿を描き
- 環境変化に対応できる人づくりを行う

06/05/06SPIN

6

会社の文化・風土の再構築

- ツールの良し悪しだけでもない。
- 開発プロセスの良し悪しだけでもない。
- 当たり前のことができているか？



06/20/06SPIN

1

企業文化について考えてみましょう。

- 富士通の企業文化とは？
 - ある？ ない？
- 文化の必要性？
 - 会社として組織(チーム)として力を発揮するため。
 - 世の中の流れに対して「変動対応力」を持つため。
 - 楽に仕事をするため。
- 文化とは？
 - 社員一人一人の実践活動からしか構築できない。

06/20/06SPIN

2

文化・風土の例として

- 朝、ある会社に行くとき誰もおはようございます。と挨拶してくる姿を見たときに。
- 会社の近くの地下鉄の駅を出ると、10人近いサラリーマンがホウキと塵取りを持って掃除をしている姿を見た時に。

皆さんはどう思われますか？

06/20/06SPIN

3

社員の行動から企業価値、社会貢献へ

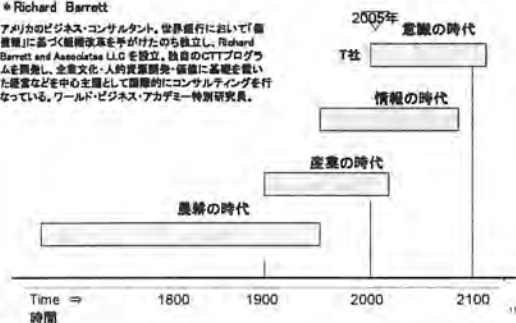


10

競争優位性を築いた分野

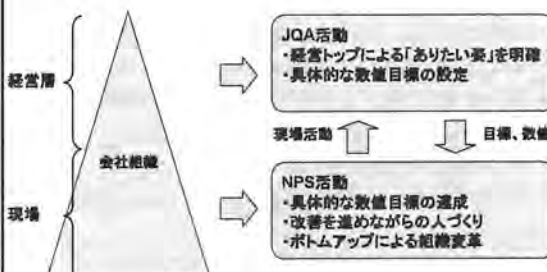
* Richard Barrett

アメリカのビジネス・コンサルタント。世界銀行において「価値観」に基づく組織改革を手がけたのを機に、Richard Barrett and Associates LLC を設立。独自のCTIプログラムを開発し、企業文化・人的資源開発・価値に基礎を置いた経営などを中心として国際的にコンサルティングを行っている。ワールド・ビジネス・アカデミー特別研究員。



11

FSTにおける企業文化の再構築



06/20/06SPIN

12

文化・風土を作るための施策

- JQA(日本経営品質賞)の実施
 - 経営層(部長以上)の意識統一
 - 会社の方針・ビジョンの明確化、目標の設定
- NPS(Nagare kaizen Profit System)の導入
 - 現場(課長以下)の意識統一
 - トヨタ生産方式を参考にして仕事の流れを整流化することによって利益を生む方式を確立する。
 - 組織体制の整備
 - NPS推進組織の新設(生産技術部⇒NPS推進部)
 - KAIZEN塾の新設
 - 全社教育の方針変更(NPSに追従)

DMS/CS/SPIN

12

では、どうやって現場の意識を変える？

- KAIZENを実践する核となる人材の育成
 - KAIZEN塾(現状認識)
 - 自分達の仕事のやり方、価値感、方向性のバラツキなどから生ずる問題についての気づき
 - KAIZEN塾(他の異文化の理解)
 - TPS(Toyota Production System:トヨタ生産方式)のDNAを注入。
 - 実践主義、3現(現地・現物・現認)主義、人間尊重などの原理・原則に沿った実践教育
 - 伝道師として他の部署にKAIZENを如何に浸透させることができるかの命題の実践。
 - 伝道先の部署には、KAIZENリーダーが育成される。

DMS/CS/SPIN

KAIZEN活動への反発

- 人間誰も今までの仕事のやり方、考え方を
変えたくない。他人に指図されたくない。
 - 現状否定に対する抵抗
- 余分な仕事が増えるだけ。
 - 効果を経験してない、知らないための抵抗。食わず嫌い。
- トヨタそのものに対する抵抗
 - 富士通の文化で良いじゃないか。なぜ、トヨタなのか？俺はトヨタより、日産、ホンダが好きなんだ。

DMS/CS/SPIN

13

KAIZEN活動を理解してもらうためには

- 現状の正しい認識
 - 会社の業績、一緒に考えてもらう(他社と比較しながら)
- 仕組みはトップダウン
 - 何でも稽古事(ピアノ、茶道、柔道など)はまずは言われた事をやってみることから始まる。
 - 5Sも仕組み、制度としては入れるが、現場には強制的にやらせるのではなくやってみようから。
- 少しでも現場が変わって良くなることを経験
 - 現場が少しでも変わってくる(変化)を味わう
 - チームのモチベーションがあがる。
 - 気づく機会が増えてくる。

DMS/CS/SPIN

14

資本主義である以上働かなくてはならない

働いている人達には色々な思いがある。

- 仕事をやる以上楽しく効率的に(楽に)仕事をしたい。
- お金だけでなく、充実感、達成感などを味わいたい。
- いい会社ですね。人から言われる会社に勤めたい。
- 彼女や奥さん子供に何か買ってあげたい。
- 新しい車が欲しい。
- 地域の人達に必要とされるそんな会社にいたい。

会社が儲かって利益を出さないといけない

少しでも儲かっていれば良いじゃないか。(危機感の薄さ)

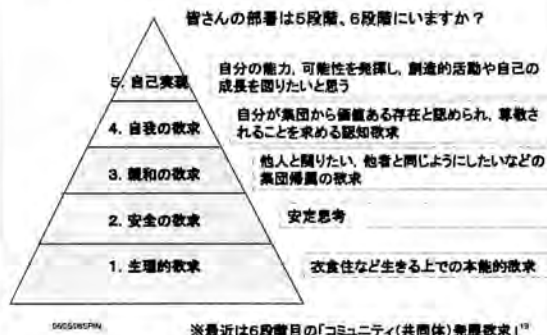
従業員が豊かで幸せな生活を送るには？

- 利益を生み出す会社でなければならない。
- 人間尊重の考え方にに基づいた仕事の与え方
- 社会にとって必要な会社でなくてはならない。
 - あるアメリカの大手企業の社長にあなたの会社はどういう会社でいたいですか？
 - 世界で一番の会社になりたい。
 - 世界にとって一番ではなく？
- 地域貢献ができ奉仕の精神を持っている。

DMS/CS/SPIN

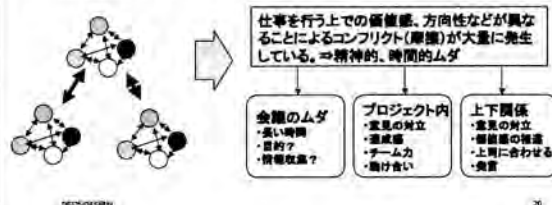
15

マズローの5段階欲求説



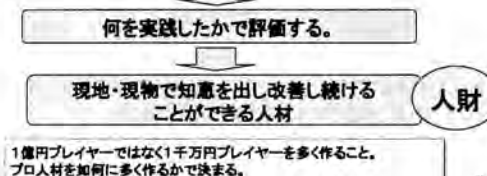
原理・原則論の重要性

- 自分の中で判断するとき、いつも不安で上司にお伺いを立てていませんか？
- いつも押しの強い人に言い負けていませんか？
- 発言したら攻(口)撃される不安はありませんか？



真の実力とは？

- 口の立つ人でもなく。
- 技術論を語れる人でもなく。
- 自分なりの価値感を持っている人でもなく。
- プライド(自信)だけでもなく。



トヨタ生産方式(TPS)について

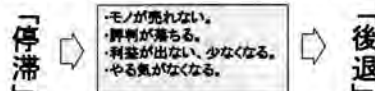
TPS: Toyota Production System

TPSの二本柱

- JIT(ジャスト・イン・タイム)
 - 必要なときに必要なものを必要なだけ
 - 在庫を持たない(普通は欠品をしてしまう)
 - 例: 開発が始まる当日の8時40分に要求仕様書、開発者、環境などがすべて段取りされ作業が開始できるイメージ。
- 自動化
 - 異常検知の仕組み。品質を後工程に送らない。
 - 例: 結合テストを行ったらバグが出た。全員が開発を止めて根本問題の解決と再発防止。
 - 異常を上にあげることができる人は信頼できる人。

なぜ改善をしなければならないのか

- 世の中は常に進歩している。
- 非常に短い周期で「変化」している。
- 会社、組織、本人が何も現状を変えようとしないと、「停滞」しているように見えても、実は「後退」している。



- 本人は進歩しているつもり(ノウハウや設備の自己満足)では意味が無い。
- その進歩が他社と比較して負けても意味が無い。

- お客様から見た進歩でなければならない。



常に改善し続けなければならない。



会社は生き残れない。

06/05/08SPIN

25

改善は原価低減へのあくなき追及

- 売値＝原価＋利益
- 売値－原価＝利益 どちらが正しい？
- 原価低減を追求する真の意味を理解するには、資本主義社会における市場競争の原理を「再認識」すること。
- 売値はあくまでもお客様が決めることでありこちらの言い値では成立しない。

06/05/08SPIN

26

NPSにおける人材育成プログラム

06/05/08SPIN

27

トヨタにおける人材育成の仕組み

- QC活動（小集団活動における人材育成）
 - 原理・原則に基づく実践活動
 - 魅力品質の考え方（顧客から見た価値創造活動）
 - PDCAの実践
- 自主研活動
 - 生産調査部のメンバーがTPSの考えに基づいてボトルネックとなっている部署に行き改善活動を推進しながら人材育成を行う活動。

06/05/08SPIN

28

トヨタにおける人づくりとは？

改善し続けることができる人材の育成

⇒ こんな人材が1000人いたら会社は良くなる？



改善能力の高い人が増える。

- ムダを見つける能力の醸成
- 今までの常識を非常識と思う
- あたりまえの事を徹底的に実践
- リーダーシップ（率先垂範して実践）
- 頭であれこれ考える前にやってみる
- 原理・原則の徹底

06/05/08SPIN

29

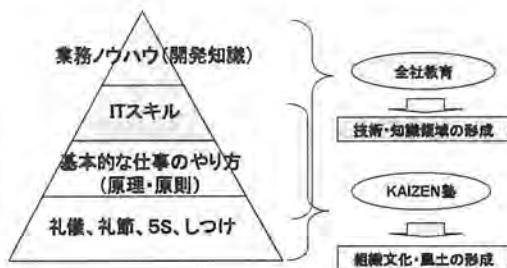
NPSで求められる人材像

- 基本は、トヨタ生産方式（TPS）を参考にしているため下記の行動原則が求められる。
 - 小集団活動
 - 現地・現物主義
 - 実践主義など
- NPSとしては、上記のTPSの原理・原則に基づき、ソフトウェア開発現場でKAIZENを実践し続ける人材を育成する。

06/05/08SPIN

30

NPSにおける人財の4段階モデル



06/05/08SPIN

31

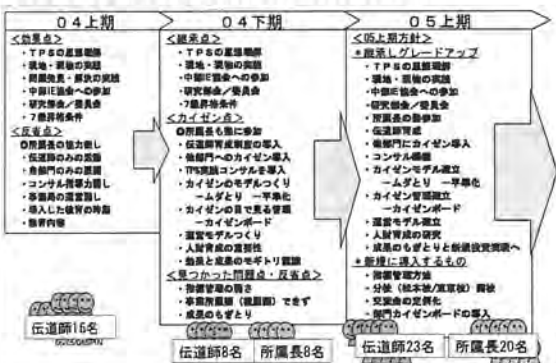
KAIZEN塾のご紹介

2004年～2005年活動状況

06/05/08SPIN

32

KAIZEN塾の変遷(継承とカイゼン)



伝道師15名

伝道師8名 所属長8名

伝道師23名 所属長20名

KAIZEN塾変遷(継承とカイゼン)



06/05/08SPIN

伝道師24名 所属長24名

34

体制

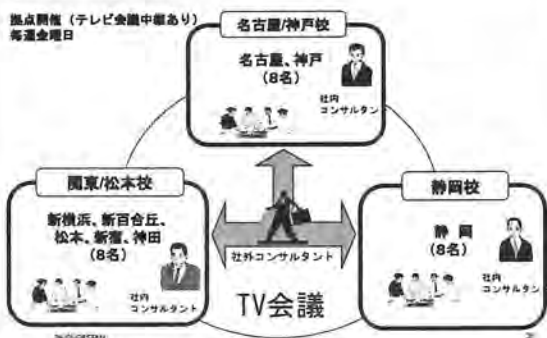
- 担当役員：平木社長
- 全体
 - 経営改革室長：吉田茂
 - 塾長：NPS推進部部長 高木
 - 事務局：高森
- 名古屋分校
 - 分枝長：高木部長
 - 事務局：高森
- 松本分校
 - 分枝長：藤澤課長
 - 事務局：高森
- 新百合丘分校
 - 分枝長：上田部長
 - 事務局：未定

06/05/08SPIN

05下期から新設

- 新横浜分校
 - 分枝長：田中部長
 - 事務局：未定
- 静岡分校
 - 分枝長：松澤部長
 - 事務局：未定
- 神戸分校
 - 分枝長：村瀬専任部長
 - 事務局：未定
- 新宿/神田分校
 - 分枝長：小野課長代理
 - 事務局：未定

KAIZEN塾の開催



06/05/08SPIN

35

KAIZEN塾導入手法 ストア管理のご紹介

ストア管理手法とは

■ ストアとは

TPSの“後工程引取り”を実現するカンバン方式で用いられる“カンバンストア”に由来

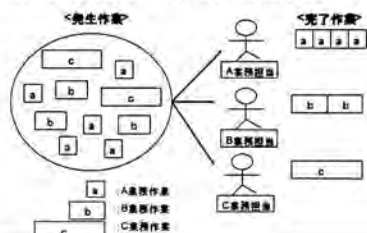
■ ストア管理手法とは

以下の4つのTPSコンセプトを抽象化し、一般的な業務プロセスのKAIZENに適合させた手法

- ①平準化
- ②ストア引取り(一個流し)
- ③一列待ち
- ④目で見る管理(見える化)

一般的な業務プロセスモデル

ランダムに発生する3種類の業務作業を各担当が順次処理



問題点

担当間で処理能力と業務負荷のバラツキが発生し、全体の処理能力が低下する。

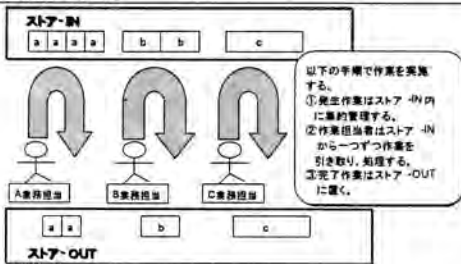
①ストア管理手法による平準化KAIZEN

指示管理を集約し、担当間の能力と負荷のバラツキを最小化。作業滞留のムダを除き、納期が短縮され、全体の処理能力向上。



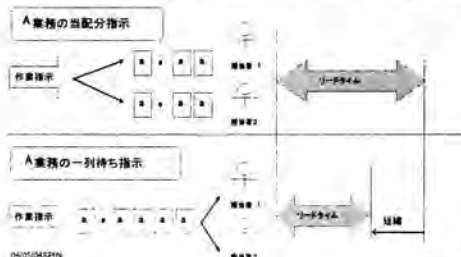
②ストア引き取り(一個流し)による作業

極力、2つ以上の作業を並行して実施しない仕組み。作業者の頭の切替えによるムダや、依頼者の待ち時間のバラツキがなくなり、納期短縮につながる。



③一列待ちの作業管理

作業をストア-IN内に一列待ちキューとして管理する仕組み。作業がランダムに発生する場合、担当に等配分して処理するより、一列にして先頭から処理する方が納期が短い(待ち行列理論より)



10

34

DECLARATION OF INTEREST

References

Date						Time		Location		Weather		Wind		Sea		Visibility		Temperature		Humidity		Pressure		Other	
Day	Month	Year	Start	End	Area	Temp	Wind	Dir	Speed	Height	Direction	Temp	Humidity	Barom	Other	Temp	Humidity	Barom	Other	Temp	Humidity	Barom	Other	Temp	Humidity
1900	1	1	0800	1200	Off Cape Cod	65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
1900	1	1	1200	1800	Off Cape Cod	68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
1900	1	1	1800	0100	Off Cape Cod	62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72
						65	10	SE	10	2	SE	65	75	30.00		65	75	30.00		65	75	30.00		65	75
						68	15	SE	15	3	SE	68	78	30.05		68	78	30.05		68	78	30.05		68	78
						62	10	SE	10	2	SE	62	72	30.00		62	72	30.00		62	72	30.00		62	72

- 2014-2015

第56回 SEA-SPIN Meeting

NPS伝道師活動報告

(株)富士通ソフトウェアテクノロジー
 ユビキタス事業部
 ユビキタス統括部第三開発部
 04上期KAIZEN塾生 山崎 勲

2006年5月8日

自身のKAIZEN活動経緯

- 2003年度下期
 - 旧富士通プライムソフトテクノロジー時代に、「生産性向上活動」開始
 - モデルプロジェクトとして当PJが選出され、活動を実施
 - ⇒ トップダウンのため、実質リーダー1人での活動となる
 - ⇒ 成果が見えない、PJ員に浸透しない、などの問題発生
 - ⇒ 活動自体が形式化してしまった
- 2004年度上期
 - KAIZEN塾発足、一期生として伝道師活動を開始
 - リーダーはKAIZEN塾生として、PJ員主体による生産性向上活動を実施
 - ⇒ ボトムアップによる活動となり、メンバー全員で取り組む
 - ⇒ リーダーとして行うこと、メンバー全体で行うことが噛み合い、風土として浸透
- 2004年度下期～現在
 - KAIZEN塾が定着、塾生も各PJに1人はいる状況となり、手法もツール化されたことで会社全体での活動となった

Copyright © Fujitsu Software Technologies Limited. 2006. http://spinfusion.com/PDF

2004年度上期KAIZEN塾活動

- 第一期KAIZEN塾生15名
- TPSの理解と、ソフトウェア業界への展開のために何をすればよいか、について検討
 - 風土として根付かせるため、日報の定着化や5S活動を実施
- 問題解決技法、BPIE(ビジネスプロセス革新エンジニア)技法などの習得
- コンサルタントによる指導
 - 自PJが抱える問題点を把握
 - 問題点は「なぜ、なぜ」の繰り返しにより根本原因を探り出す
 - 問題解決のための取り組みを決定し、スケジュールを組む
 - 毎週のKAIZEN塾で活動内容を報告し、コンサル指導を受ける
 - 指導内容をPJ員にフィードバックし、活動に取り入れる

Copyright © Fujitsu Software Technologies Limited. 2006. http://spinfusion.com/PDF

KAIZENプロジェクト業務内容

- プロジェクトの業務内容
 - メンバ14名による富士通製パソコンのLANドライバ製品化作業

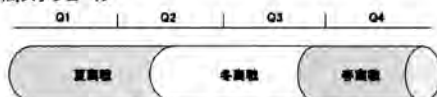


世界のLANベンダーから富士通製品に組込むLANドライバの調査
 選定・検証・開発・サポートを実施—「ネットワーク主担当部署」

Copyright © Fujitsu Software Technologies Limited. 2006. http://spinfusion.com/PDF

KAIZENプロジェクト業務内容

➢ 年間スケジュール



各商戦毎に定量業務が発生

- β版評価(OEM元から提供されるデバイスドライバの評価)
- 製品化評価(マスタースケジュールに合わせて入試)
- サポート業務(出荷後製品のサポートセンターからの問い合わせ)

※年間100機十機種、1000モデル以上が対象

Copyright © Fujitsu Software Technologies Limited. 2006. http://spinfusion.com/PDF

テーマ選定経緯

➢ 選定理由

- LANドライバ製品化作業は定量化作業+繰り返し作業であること
- 2004年度上期KAIZEN塾導入以前、2003年度下期に生産性向上活動を実施、モデルプロジェクトとして作業効率化を実施
 - ⇒ 数値効果が表れず、中途半端
- 生産性向上活動における活動指針がトップダウン
 - ⇒ ボトムアップを中心とし、グループ員全体の意識向上を促す
- KAIZEN塾への入塾
 - ⇒ プロセス改善のノウハウを習得

➢ 何より。。。.

LAN製品化業務における利益率UP!!

Copyright © Fujitsu Software Technologies Limited. 2006. http://spinfusion.com/PDF

現状把握・目標(当時)

問題意識

- LANドライバの製品化作業において
 - 環境不備によるリードタイム遅延や評価品質のバラつき
 - メンバのスキル(経験)不足やドキュメント不足による評価結果のバラつき
 - マネジメント不足により発生するムダ(あいまいな作業管理、人員配置の偏り)
- ⇒ 不要な工数増加を生み出している



設備・能力・マネジメントの観点から工数増加の原因を分析し、解決する

Copyright © Fujitsu Software Technologies Limited 2005. All rights reserved. group/sea

活動計画・実施方法

設備

- 評価環境の5Sと運用のルール化
- 新規評価環境の設置

能力

- メンバの多能工化
- テスタのスキルアップ

マネジメント

- 業務日報による作業管理
- 毎日朝礼で各メンバの進捗確認および、見える化の有効利用
- テスト手順書/テスト項目書の標準化による迅速な人員配置「誰がどんな作業でも」

Copyright © Fujitsu Software Technologies Limited 2005. All rights reserved. group/sea

苦労した点

- 開始当初からメンバ(間)に疑問視の声
 - なぜ朝礼?なぜ作業実績入力?
 - ⇒ 「KAIZENにはまず風土作り」の意識をメンバに浸透、理解を得る
- できることから
 - 実験室の5Sの取り組み
 - ⇒ メンバ自身も積極的に
 - ⇒ 他に求められることで変化を実感
- 見える化で何を見えるようにするか
 - 「見える化」にしないための工夫
 - 作業経費、ToDo、作業指示、チェックシート etc...
 - NPS目標、スキルマップ、KAIZEN提案数(呈取表)
- 既存のテスト項目書/手順書
 - 「あるけれど、実は熟練者に聞かないと判らないことが多い」
 - ⇒ 誰が見ても判るものを作らなければ意味がない

Copyright © Fujitsu Software Technologies Limited 2005. All rights reserved. group/sea

設備KAIZEN ー実験室5S活動ー

➢ 考えてみれば「当たり前」のことに実施

- 「(A)当り前のことを、(B)馬鹿みたいに、(C)ちゃんとやる」
- 実験室のレイアウト図を作成「あるべき姿」
- スーパーのような、「陳列形式」
- CD、DVDには、「ラベル貼り」
- 小物収納のダンボールにも、「ラベル貼り」
- 足元スペースは基本的に「何も置かない」

⇒ これだけで、リードタイムを1/5にすることができた

➢ 5S ⇒ 取り掛かりやすい ⇒ 成果が見えやすい

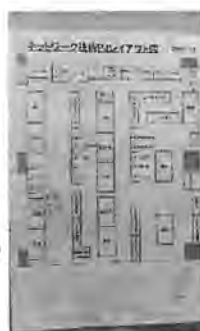
- 大変なのは、「継続」
- 大切なのは、「PDCA」

Copyright © Fujitsu Software Technologies Limited 2005. All rights reserved. group/sea

設備KAIZEN ー実験室5S活動(1)ー



- <ポイント>
- どの棚に何が置かれているか
 - ロッカーには何が入っているか
 - 机は誰が使用しているか
- ⇒ 実験室各所に設置、見える化



Copyright © Fujitsu Software Technologies Limited 2005. All rights reserved. group/sea

設備KAIZEN ー実験室5S活動(2)ー



- <ポイント>
- 使用していないケーブルを、100円ショップで購入したハンガーフックを利用して1本ずつ束ね、1箇所で保管
- ⇒ 取り出し、収納が容易に



- <ポイント>
- ハンガーフックを棚ごとに配置
- ⇒ 取り出しが更に容易に

Copyright © Fujitsu Software Technologies Limited 2005. All rights reserved. group/sea

設備KAIZEN ー実験室5S活動(3)ー



<ポイント>
 ・ダンボール製プレートに、保管すべき材料名(例えばペンダ単位)を記載し、見える化。
 ⇒どのP/Oに何のチップが搭載されているかが一目瞭然となり、リードタイムが短縮
 ・棚等に台帳を設置、持ち出しの頻度、使用者の名前、持ち出し日、返却日を記録
 ⇒所在の見える化



Copyright © Fujitsu Software Technologies Limited. 2006. All rights reserved. (japan/fst)

設備KAIZEN ー実験室5S活動(4)ー



<ポイント>
 ・実験室机は作業する場所であり、2段机との間の空間を利用して、必要な資料をフックで吊るして整理
 ⇒必要なものを必要ときに



<ポイント>
 ・実験室机にもスケジュールを貼るなどの工夫
 ⇒事務所に限らず、次の仕事を取り出せる

Copyright © Fujitsu Software Technologies Limited. 2006. All rights reserved. (japan/fst)

設備KAIZEN ー得られた成果ー

- 実験室の5S
 - ー 整理・整頓に1ヶ月、ルールに則った運用開始して3ヶ月
 - ⇒ 継続した5S活動(単なる大掃除に留まらなかった)
 - ⇒ 機材準備時間(リードタイム)短縮
 - ー 実験室スペースに清潔感
 - ⇒ 作業メンバから不快感が無くなる
- SystemCastWizardサーバ設置
 - ー 評価OSリカバリ時間20分短縮
 - ⇒ 繰り返し作業のため、トータル効果は1人月/商戦
- 遮断カーテン設置
 - ー 性能測定回数の削減(1回で済むようになった)
 - ⇒ 測定品質の向上(バラつきがなくなる)
 - ー クリーンルーム使用のための出張が削減
 - ⇒ 手待ち時間の削減

Copyright © Fujitsu Software Technologies Limited. 2006. All rights reserved. (japan/fst)

能力KAIZEN ースキルマップ(1)ー



Copyright © Fujitsu Software Technologies Limited. 2006. All rights reserved. (japan/fst)

能力KAIZEN ースキルマップ(2)ー



Copyright © Fujitsu Software Technologies Limited. 2006. All rights reserved. (japan/fst)

能力KAIZEN ー得られた成果ー

- スキルアップ
 - ー スキルマップ作成によりスキルアップ重点項目は明確化
 - ー 勉強会だけではメンバがスキルアップを実感できない
 - ⇒ 実践の必要性
 - ー スキルアップカリキュラム作成、サポート部門教育コンテンツとしても利用できる
 - ⇒ 作成したコンテンツにてサポートセンタ向けに無線LAN教育を実施
- マルチタレント化
 - ー テスト環境構築手順書、テスト仕様書の標準化により、テストのローテーションが可能となったことから業務内で実施、スキルの底上げに成功

Copyright © Fujitsu Software Technologies Limited. 2006. All rights reserved. (japan/fst)

マネジメントKAIZEN ー得られた成果ー

- テスト項目書/手順書
 - ー 商戦対応終了後に反省会を実施
 - ➡ テスト観点の見直し、不要項目の削除、次期商戦にフィードバック
 - ー トラブル情報とのリンク
 - ー テストシート毎に作業時間測定
 - ➡ テスト作業が標準化、正確なスケジューリングができるようになった
- 見える化・朝礼・作業日報
 - ー 貼り出す情報について試行錯誤
 - ➡ 必要な情報が整理されたことで全体作業を把握
 - ー 作業実績の分析により、作業比率が明確に
 - ➡ テスト項目書の標準化によりベンダ毎の偏りが減少
 - ー 朝礼実施でタイムリーな状況報告
 - ➡ グループミーティングを廃止

Copyright © Fujitsu Software Technologies Limited, 2006. All rights reserved.

振り返り

- KAIZENを開始してから約2年・・・
 - ー 朝礼、日報、SS、挨拶など、仕事の基本となりうることを習慣化することで風土作り
 - ➡ 地道なことであるが、個々の認識として「腹に落とす」ためには重要
 - ➡ メンバの意識が変わる、例えば、「1日は12H⇒8H」
 - ➡ 最初は「やっている人が目立つ」⇒「やっていない人が目立ってしまう」
- ー 継続することの難しさ
 - ➡ 形骸化への危機
 - ➡ 幹部社員参加による定期的な意見交換会で、会社全体の取り組みであることを再認識
 - ➡ '05年度塾生の取り組みがよい刺激に！
- ー 常に取り組むことが重要、一時的な盛り上がりでは成果は出ない

Copyright © Fujitsu Software Technologies Limited, 2006. All rights reserved.

FUJITSU SOFTWARE TECHNOLOGIES LIMITED.

SEA SPIN Meeting

ソフトウェア会社のトヨタ生産方式導入から学ぶ

参加者アンケートのまとめ

田中 一夫

CSK システムズ

(SEA SPIN 世話人グループ・メンバー)

ゴールデンウィーク明けの 5 月 8 日に、第 56 回の SEA-SPIN Meeting が開催されました。今回のテーマは、「ソフトウェア会社のトヨタ生産方式導入から学ぶ」で富士通ソフトウェアテクノロジーズの高木さんと山崎さんのお話を聞きました。

昨年、「実践！！IT屋のトヨタ生産方式 あるソフトウェア会社の挑戦」という本が風媒社から出版されました。この本を昨年暮れに、SEA-SPIN 世話人の一人が読み、高木さんの会社まで押しかけて、活動内容をお聴きし、見せていただきました。今回は、そこで見聞した内容を、SEA-SPIN の場でお話していただきました。

参加申し込みは、SEA-SPIN の例会始まって以来最高の 71 名でした。会場の関係から数名の方の申込みをお断りしています。このレポートは、参加された方々にお願いした事後アンケートの報告です。

参加者数:63 名 内訳(SEA 会員:24 名、賛助会員:3 名、一般:36 名)

アンケート回収率 57%(有効回答数 36 名)

1. 今回の Meeting 案内を、どこから知ったか？

SEA からの案内 50%

IPA SEC からの案内 22%

知人からの紹介 19%

その他 8% (企画担当、上司からの参加指示など)

2. 今回の Meeting 内容は有意義でしたか？

有意義 94%

その他 6%

内訳:①モチベーションの重要性や作業時間管理など、有意義な部分もあったが、期待していたものとは違った。

② 回答の選択肢がちょっと粗すぎる。「どちらかと言えば」みたいな選択肢あった方が付けやすいです。分析も楽なのではないかと思います。

③ 対象業務が、発表にあったように繰り返し作業の場合には導入可能だと思うが、受託作業の場合はむずかしいのでは？

3. 感想

あ(会員): 何をやるにも人づくりが大切で、参加ではなく参画するマインドをどう育てていくのか、たゆまない努力の大切さを再認識しました。そのために、伝道師、熱狂者を育てる、そういっ

た思いを持っている人を引っ張ってくる、こういったことも必要なのだと思います。

い(会員): 製造業の品質に対する考え方がソフトウェアに適用できるかということが、ずっと気になっていましたが、「ムダを取る」という発想はかなりしっくりくる言葉のように感じます。自分の活動に対するヒントを得られたような気がします。

う(一般): 最後の質問にもありましたが、プロセス定義も重要ですが、そのプロセスの中で、いかに業務の充実、生産性向上を実現するか。まさに高木さんのお話は、そのことに一つの道を切り開いていると感じました。その意味でたいへん有効でした。実は、私も開発プロセス定義は、ある程度のものがあればよく、その中での指導を如何に可視化するかに注力すべきと思います、PMBOKを雛型にして管理職の開発チームに対する働きかけの方針と、その実績を聞く(場合によっては指示)したりする活動を始めようとしています。

え(一般): 改善が目的ではなく、改善をできる人を創ることが目的だということがわかり、有意義でした。

お(会員): トヨタ生産方式のソフトウェア開発への適用ということで期待していたが、実際の事例は、テストおよび組み立て工程への適用であり、「ソフトウェア開発」という点からは、典型的なものではなかった。また、改善効果について定量的な結果を明確に述べていただけなかったのが残念である。

か(賛助会員): 非常に興味深い内容でしたが、紹介されたような方法のみで生産性を5年間で2.5倍にすることが可能なのか、疑問に感じました。その点に関する目論見等があればアンケート結果にでも記載していただきたいと思います。

き(会員): 今回のFSTさんのお話は、非常に分かりやすく整理されていたように感じた、私の頭の中で過去の経験を整理するのに非常に役に立ちました。

く(一般): トヨタ方式を適応するのは大変なんだと、あらためて思いました。大変参考になりました。すぐさま適用することは難しいと思いますが、講演の中で教えていただいた言葉は、これからの会社生活で有効だなと感じました。以下、心に残った言葉を列記します。

- ・本を読んだということを開きたいわけではない。「何をやり」「その結果どうだった」を開きたい。(トヨタの人の言葉)
- ・5S(整理、整頓、清潔、清掃、躰)が必要。
- ・トヨタ方式を説明するときは、オーナー社長のところしか、説明しようとは思わない。もし、オーナー社長でもなく、それなりの役職にもない君がトヨタ方式をやりたいのなら、継続的に改善を行う組織を作れ。それくらいの覚悟を見せろ。(トヨタの人の言葉)
- ・KAIZEN塾は6カ月。毎週金曜日、計168時間かかる。それくらいでないと人は育たない。
- ・部下の不平不満を受け止めろ／それを受け止めずして、それより上位の意見が出てくるわけがない。
- ・経営者層に月1回危機感をあおる。「会社はつぶれるんだぞ」と。
- ・教えよう／話そうという人が一番勉強する。
- ・世界で一番になるのではなく、世界にとって一番になることを目指せ。
- ・トヨタはマズローの5段階説で人を育成している。
- ・自立的な行動ができる人間を育てたい。
- ・1億円プレーヤを作るのではない。1千万円プレーヤをどれだけ多く作るかが課題。
- ・改善しないこと、すなわち「停滞は後退」である。
- ・改善には、改善しようとする「思い」が重要

け(会員): 高木さんの講演では、企業文化／組織風土の話がありましたが、改善の出発点は

そこからだと、前にいた会社での自らの取組みを振り返って実感しています。講演を聞いて、会社のトップを巻き込んで進めている改善活動の成果がよく出ていると思いました。ソフトウェア開発現場にトヨタ生産方式を取り入れて改善活動を行うには、まだまだ多くの課題を解決していく必要があるとも感じました。

こ(会員)：改善運動は人材に依存する部分が多いので、年功序列型の日本企業で定着し成果が出るように思います。しかし、海外の環境などに見られるような、外部要員や退職率の多い開発の現場において、ノウハウをどのように積み上げて利用するか、組織的な改善運動は機能するのか疑問を感じます。

さ(一般)：トヨタ生産方式については従来から興味があり、大野耐一さんの本も読んでいました。スタッフとして会社の品質管理の改善活動を担当しているので、どうすれば改善を組織的な活動として展開できるかについて、いつも考えています。今回の発表は、当たり前のことをちゃんとやるのが組織的に行われていて、感心しました。大事なものは手法ではなく人づくりであることが少し分かりましたが、まだ実感が伴わず、頭で理解したというレベルです。

し(賛助)：内容はよかったと思いますが、具体的な事例も聞きたかったです。成果をどのようにとらえているのかなど。

す(一般)：昨年初までプロセス改善推進の立場におりましたが、現在は現場のプロジェクトで、サブリーダー兼QA責任者になっています。立場が変わっての勝手な感想ですが、推進する側の眼が、現場に向かっていない(?)ような雰囲気を感じました。べきこと、べき姿も大事ですが、現場側も好んで参加していけるような会合となると良いかと思っています。実際の現場は、社外のセミナー参加するのが難しいのが実態ですが、今後ともこの種の集まりに何らかの形で参加していきたいと思います。

せ(一般)：方法論や手法から入るプロセス改善は一過性のものにすぎず、長続きしないということを改めて認識しました。各個人の意識改革と全員の価値観の統一によってプロセスを改善していくことが、長続きする秘訣であると感じました。当社ではウォーターフォール型の開発が主となっているのですが、このような開発プロセスでの改善事例を多数紹介していただけると今後の参考になると思います。

そ(会員)：プロセス改善についての講演には手法の話が多いが、その基礎作りの話であったので、参考になった。

た(一般)：トヨタのカイゼンについては、「ザ・トヨタウェイ」を読み、参考になることが多くありました。しかし、実際に適用するためには難しいと感じています。トヨタでは文化として定着しているため、新しい人にも浸透しているのではないのでしょうか。ソフトウェア・テストの工程にカイゼンを適用し浸透していったら事例を聞くことができ大変参考になりました。ソフト開発の部門への適用例の紹介を次回に期待します。作業者が自立してカイゼンを回していくようになるまでの工夫について苦労があったものと思いますので、そのあたりを詳しく聞ければと思います。

ち(一般)：久しぶりの sea-spin ですが、実践者のナマの声は一言一言が納得できる、重みのある発表でした。直接 TPS の導入に関わる機会はゼロですが、考え方や行動には即活かせるものが多々ありました。

つ(一般)：現場を見ないと、、、とありましたが、高木さんの会社でフォーラムやるべきだったかなどちょっと思いました。開発に関わるところの事例も見なかったな。誰かも質問していましたが、日々のタスクの割振りが一番難しそう。知識・教育のジャストインタイムということが、「おっとそうか」と気付かされたところでした。同じ形では出来ないでしょうが、形を変えて適用したいと強く思いました。

て(会員)： こういう開発方式があることを知らなかったのが、聴講した意義はあったと感じています。ただし、特に前半の講演において、精神論、抽象論に多くの時間を割きすぎて、具体的な話に触れる時間が短くなったのは残念です。SEA-SPIN の WebPage に、あらかじめ「実践!!! IT 屋のトヨタ生産方式」を読んでおくと書いてあったのですから、あの本に書いてあることを時間をかけて説明する必要はなかったと思います。同じ Web で紹介されている論文(PDF)の内容はより具体的でしたが、講演ではこの論文にはほとんど触れられませんでした。せっかく実績値まで挙げて説明しておられるのですから、この内容に関連した説明が聞きたかったです。開発組織に「お客様プロキシ」というメンバーを参加させるというアイデア は面白いと思いました。

と(会員)： 「KAIZEN」に代表される日常の活動は言葉でいうのは簡単ですが、それを風土、文化として定着させるには明示的な強い意志と、長いめげない努力の継続がもっとも重要だと感じました。

は(一般)： 今回の SPIN-Meeting はとても参考になりました。コーディネーターの方の進め方がうまく、Q&Aでも多くの情報が得られました。また機会があれば参加させていただきたいと思います。

ひ(一般)： トヨタ生産方式(TPS)を用いた開発手順の紹介かと思って参加しましたが、「改善」活動に終始した内容で、とても参考になりました。特に、「今日は改善したか」「停滞は後退」「改善できる人材の育成」「ABC」という言葉が印象に残りました。これらの内容をもとに今後に活かしたいと思います。

ふ(一般)： ソフトウェア開発現場へのTPSの適用を試みられたことと、実績をあげられるまでの過程とそのポイントが拝聴でき、今後の改善活動において非常に参考になりました。弊社でも、効果が現れそうなどころから適用していきたいと強く感じました。

へ(会員)： 「プロセスを作ることよりも、プロセスを改善出来る人を育てることが大事」という言葉には、はっと させられました。現場で実践している方々は、相当なパワーを求められているのだなあと、改めて感じました。

ほ(会員)： SPIN HP へのアクセスを増やす方策を考えねば。。。コーディネーション有難うございました。

ま(会員)： コミュニティの維持というのは非常に難しいですね。継続していただいていること、非常に感謝しております。JASPIC には参加しておりませんので、SEA のような場はとても私としては有益なコミュニティです。個人で所属できるのがとても助かります。

み(一般)： 業界を超えた情報交換の場として有意義だと感じました。

む(会員)： やはり、カイゼンをソフトウェア開発作業そのものに適用した例を、是非披露していただきたかった。それでも、取り組みについて聞けて、大変有意義でした。

め(一般)： 実体験を交えた話で説得力があり、面白かったです。

も(会員)： 改善という異文化を導入する苦労は、さぞかし大変なことだろうと思いました。企業文化として根付き、定着することを祈るばかりです。

や(会員)： SEA/SPINの多角的な活動、尊敬します。また、これからもいろいろと相談にのってください。

ゆ(会員)： 久しぶりに参加しました。興味深い話を聴けて良かった。

よ(賛助)： 現場寄りのお話だったので、大変参考になりました。会社の規模や風土が異なっても、みなさん同じような問題を抱えていらっしゃるのだなあとということを感じました。「技法」「手法」よりも、最終的にはTOPのコミットメントと伝道者の熱意 なんだと思うようになりました。また、お話の中で何度も出てきた「腹に落とす」というフレーズ、とても印象に残り、また心に響くものが

ありました。

4. 今後 SEA/SPIN で取り上げて欲しいテーマは？

- a: 以前 CMU のメアリーショウさんが、他の分野とのアナロジーでソフトウェアの世界を考えておられました。特に化学がフィットしているといわれていたように記憶していますが、他の分野に学ぶ、といった企画も面白いのではないのでしょうか？
- b: 組込み系のソフトウェア開発における品質を如何に確保するか。業務系ソフトウェアとの違いについて。
- c: 今回の話では、プロセス改善ではなく、作業の無駄の排除による 作業効率の向上という話がメインであったように感じます。プロセス改善という部分に焦点を当てた事例等がありましたら、紹介していただきたいと思います。
- d: 他社の改善事例
- e: 「デスマーチ」の改定版も出たことですし、プロジェクトはなぜ失敗するのか というのはどうでしょうか。何度も取り上げられているテーマかもしれませんが。
- f: 難しいですね。JASPIC との「競合」というか、SEPG Japan 等の場でいろいろな話を聞けるようになったというのも大きいかと思います。経験談でないとすると理論の話になるかと思うんですが、人が集まらないでしょうかね。
- g: プロセス改善 vs 改革、特効薬を探る リスクの定量化。
- h: テスト駆動開発の効果。
- i: 改善、意識改革の事例紹介を希望します。
- j: プロセスの定式化、定義の話が少しありました。プロセス定義を先行させることは、実態とのギャップを生むだけで、昨今、あまりよいやり方とは思えなくなっているように感じます。昔は、プロセスが混沌とした企業が多かったようですが 最近では、最低限のプロセスは整っている状況ではないでしょうか プロセス改善のやり方が、変ってきているように思います。プロセスを記述したとか、計測したとかではなく、本当に改善できた事例があれば聞きたいと思っています。
- k: プロセス改善を推進する側でなく、改善を受け入れる(受動的な段階で無理です が)側の意識改革のきっかけ等を知りたいです。
- l: アジャイルプロセスの概要(XP, SCRUM, ASD 等)と適用事例事例 (どのような分野のソフトウェア開発に有効か。)
- m: モデル派 vs 課題改善派。
- n: 小規模の組込ソフト開発における、各種手法の適用例の紹介。または、小規模の組込ソフトに対して有効な手法。
- o: 派生開発の話
- p: SEA SPIN だからプロセスの話。アカデミックにならず、事例に即したものがよいかと。
- q: 私は SEA-SPIN のメンバーではないので SPIN 内部で既に発表があったかどうかは判りませんが、日本での PSP の実践報告(伝道の苦勞も含めて)が聞けると有難いです。
- r: SPIN との関係は離れてしまうかも知れませんが、現場レベルの取り組み を中心にお願いいたします。NECグループでは、今回のテーマと同じトヨタ生産方式の岩城先生にご支援いただいて、ソフトウェアにおける生産革新に取り組んでいます。
- s: 「ISO9001を120%活用し、プロセス改善をどのように実践されているか」に関して具体的な事例をご紹介いただきたい。

- t: 既に開催済みかもしれませんが、アジャイル開発の導入に関するテーマに興味があります。
- u: 失敗学。
- v: モチベーション維持に関するテーマ。
- w: ソフトウェア開発におけるソフトウェア技術者の責任 ソフトウェア開発プロセスと初期・中間・最終成果物の関係。
- x: 今回のテーマとも多少かぶるかもしれませんが、トレーニングとか人材育成という分野について興味があります。が、もやもやしたままで書くので、まとまりがなく、申し訳ありません。会社と大学とのよい関係とか、稚内北星学園大学なんかのサテライト校の活動とか(IPA とか他でやっていますね)、TATA コンサルティングのトレーニングだとか、そんなことを皆さん(大学、産業ともに)どう思っているのかを話す機会があるといいなと。トレーニング機会はそれなりに増えているようにも思いますが、一方 正直なところその活動があまり高く評価されていない(知らないだけかも知れませんが失礼承知で書いています)ようにも思います。何かかみ合っていないような感じです。もちろん知識提供だけでよいわけでもないですが、一方で MOT のようなことで解決する話題でもないとおもいます。が、そんなことを考えていて、なんとかしようとしている人(なんとかなん かする必要がないという人でもいいですが)がいたら、お知り合いになりたいですね。書いていて思い出したのですが、だいぶ前に新谷さんの「テクニカルバイタリティー」のお話を聞きました。育成された結果はその状態が望ましいように思えますが、それを受け入れる土壌、支える環境(大学を含む)も重要かと思っているしだいです。やっぱりまとまりがなく申し訳ありません。伝わったでしょうか? でも、他の方には興味がない話題かもしれませんね。
- y: 組み込みシステムに関連した内容など。
- z: CMMI Ver1.2 の改訂目的と改訂概。
- aa: ソフトウェア調達・委託先管理に関係した品質管理・品質保証の話などあれば助かります。
- bb: 改善の効果測定が難しい印象を受けましたた、そのような事例発表はできないでしょうか?
- cc: 改善取り組みの実態を交換し課題を討議できるとよい。
- dd: 派生開発プロセスの紹介 ⇒JISX0161 との位置関係や観点の違い/共通点などを踏まえた解説を聞きたいです。

次回の SPIN 例会のテーマは、まだ決まってませんが、上記4に寄せられたご意見を参考に世話人で検討して、決めることにしたいと考えています。

アスペクト指向技術は実用に見えるか

主催： ソフトウェア技術者協会 (SEA)

アスペクト指向プログラミング (AOP) が提唱されて約 10 年が経ちました。とくにこの数年、その注目度がにわかには高まっています。アスペクトとは、1 つの関心事に対応するソフトウェア構成単位です。従来のシステムは、たとえば手続きあるいはオブジェクトという単位をモジュールとして構成されてきましたが、セキュリティ、性能、実行時データ分析などの関心事は、それらのモジュールを横断して存在します。それらの記述をモジュールごとに分散させていたのでは、可読性、保守性、再利用性がきわめて悪くなります。そこでそのような横断的関心事をアスペクトとして独立に記述し、それらと元のシステムを合成することにより、高い生産性と信頼性をもったソフトウェア開発を行う、というのがアスペクト指向ソフトウェア開発の基本的な考え方です。

AOP が普及してきた要因の一つに、AspectJ という Java ベースの言語とその処理系が公開され、広く使われるようになったことを挙げることができます。さらにアスペクト技術はプログラミングレベルだけでなく、分析・設計時のモデルとして用いられるようになってきました。

このフォーラムでは AOP 研究の第一人者の一人である増原英彦さん (東京大学) に、まず AOP の全体的な見取り図を話していただきます。増原さんはこの 3 月にボンで開かれたアスペクト指向ソフトウェア開発の旗艦国際会議 AOSD 2006 のプログラム委員長を勤められ、また AOP に関して日本語で読める優れたチュートリアルを書かれています (「コンピュータソフトウェア」誌 2006 年 4 月号)。また、実践の場で先駆的に AOP を取り入れた仕事をされている山根英次さんに、その経験に基づいた応用の可能性についてお話いただきます。

***** 開催要領 *****

1. 日時: 5 月 17 日 (水) 13:30~18:00

2. プログラム

13:00 - 13:30 受付

13:30 - 14:30 講演: “AOP 入門” 増原英彦 (東京大学)

AOP は、Java の拡張である AspectJ 言語や、AspectWerkz, JBoss AOP, Spring AOP のようなフレームワークによって容易に利用できるようになり、大きな注目を集めています。本講演では、AOP がどのようなものであり、どのようなメリットがあるのかを AspectJ 言語を中心に解説します。

14:30 - 15:30 講演: “あなたの知らない AOP” 山根英次 (スーパーグラフィックス)

近年、AOP という言葉は特に目新しくなくなってきた感はありますが、その適用範囲はまだ局所的で AOP の魅力を十分に表現しきれていないのではないのでしょうか? 本講演では、ある意味邪道(?)ともいえるような AOP の適用方法を AspectJ を用いた実装例を踏まえて解説します。

15:30 - 16:00 Coffee Break

16:00 - 18:00 討論: アスペクト指向技術の役立ち方

コーディネータ: 玉井哲雄 (東京大学, SEA 常任幹事)

パネリスト: 上記講師のお2人 + 小林修 (SRA, SEA 幹事)

3. 会場: 全国情報サービス産業厚生年金基金 (JJJ) 会館 7F B会議室

4. 定員: 30名 (申込み順, 定員になり次第, 受付を締め切ります)。

SEA Forum

「アスペクト指向は実用に見えるか」

小林 修

SEA 関西支部

1. アスペクト指向フォーラム開催の経緯

今年（2006 年）5 月 17 日と 6 月 17 日にそれぞれ東京と大阪において、「アスペクト指向プログラミング」をテーマとした SEA Forum が開催された。この企画はもともと SEA 幹事会のメーリングリスト上で今年のフォーラムの開催計画についての議論が行われた際に、玉井先生から「アスペクト指向」が提案されたのがきっかけだ。

実は、私が世話人を務める SEA 関西プロセス分科会でも一昨年の 2004 年に一度「アスペクト指向」をテーマにした分科会を開いたことがあった。（講師は私（前座）と今回も講師をお願いした山根英次氏であった。）その頃から、アスペクト指向には深い関心があり、今回の玉井先生のご提案に対しても即座に参加したい旨をメールで申し入れた。また、講師として SEA 関西プロセス分科会において講師をお願いした山根氏に再度、講師をお願いすることとなった。

玉井先生からは、東京大学の増原先生を講師として推薦していただいた。増原先生は、今年度のアスペクト指向の国際会議（AOSD）のプログラム委員長を務められるなど、アスペクト指向では日本を代表する研究者と呼べる方である。また、山根氏は既に 2 年前から AspectJ を実業務で応用されるなど、アスペクト指向の現場導入に関して先駆的な仕事をされている。

アスペクト指向プログラミングや AspectJ の実務応用への関心が高まる中、このお二人による講演をまとめて聞ける今回の SEA Forum は、非常にタイムリーな好企画であったと思う。

2. アスペクト指向との出会い

私自身が「アスペクト指向」の言葉を知ったのは、およそ 10 年くらい前のことだ。

「アスペクト指向プログラミング」の名前が広がるきっかけとなったパロアルト研究所からの論文の発表が 1996 年のことなので、ちょうどその頃である。はっきりと覚えてはいないが、SEA 関係のイベントで誰かに聞いたのだと思う。ただ、その時には、心にひかかるものは感じるものの、深く追求はせずに忘れてしまった。当時は一般のプログラマに

手に届く情報（特に日本語）がほとんど無かったのも一因だが、私自身のソフトウェア開発に対する理解とアプローチがオブジェクト指向の消化で手一杯で、アスペクト指向を受け入れる余裕も下地も無かったこともあると思う。

再び「アスペクト指向」が視野に入ってきたのは、4年くらい前のことだ。

当時、Java でのサーバー・アプリケーションの開発を生業としていた私は、Java フレームワークに関する情報を海外の WEB サイトから拾い集めるようになっていたのだが、その中に Aspect や AspectJ といった名前をちらほら見かけるようになってきたのだ。

最初に目についたのは確か、javaworld.com という Java に関する技術情報を集めた WEB サイトの記事だったと思う。その後、theserverside.com や IBM の DeveloperWorks にも記事が出るようになった。

これらの記事は、当時私が関わっていたような Java によるサーバー・アプリケーションにおけるアスペクト指向プログラミングの応用に焦点をあてたものが多かったため、今回はアスペクト指向に対する関心を大いにかきたてられた。

さらに、AspectJ に関する解説書がいくつか（海外で）出版されるようになった。その中の 1 冊「AspectJ in Action」（参考文献[2]）（上記の javaworld.com の解説記事の著者による）を読んでみたが、これは AspectJ のプログラミングについて基本的な概念から実応用まで丁寧に書かれた好著で、これにより、Java アプリケーションへのアスペクト指向の応用について、はっきりとしたイメージを持つことができるようになった。

以前より交流のあった山根氏とアスペクト指向に対する関心で意気投合し、SEA 関西でのアスペクト指向をテーマにした分科会を企画したのも、この頃である。

同じ頃、Java 以外の経路でのアスペクト指向の再発見もあった。

その頃、「ソフトウェア・プロダクトライン・エンジニアリング」にも強い関心を持ち、特にプロダクトライン開発で重要なドメイン分析の手法である「フィーチャー・モデリング」に関する情報を探していたのだが、たまたま東京の書店「Generative Programming」（参考文献[3]）と言う本を見つけて手にとってみると、この本がドメイン分析とフィーチャー・モデリングにかなりのページ数を割いて解説していることがわかった。さっそく購入してあらためてよく中身を読んでみると、アスペクト指向もプロダクトラインの変動部の実現における有望な技術として、約 80 ページに渡って解説されていた。

この本が出版されたのは 2000 年で、AspectJ や Java アプリケーション・フレームワークでの応用が一般的な注目を集める以前のことである。そのため、アスペクト指向の解説も、より基礎的な内容の紹介となっており、私にとっては、より基本的なレベルでの理解を深める（アスペクト指向の源流にさかのぼるような）意味があった。

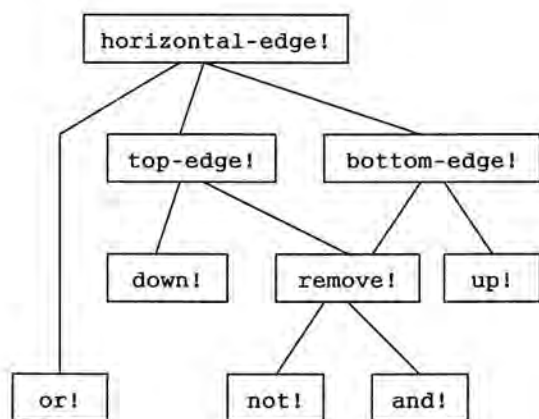
3. アスペクト指向とは何か

SEA 関西のプロセス分科会が始まった 4 年前、岸田さんから「プロセスを議論するので

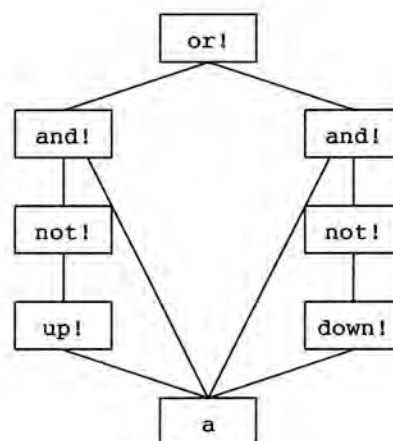
あれば、すくなくともこの論文は読んでおくべきだ」と何本かの論文のコピーを送っていただいたことがある。この中には、Royce のウォーターフォール・モデルの原論文や、Osterweil 先生のプロセス・プログラミングの論文が含まれており、大いに参考になった。特に、「ウォーターフォール開発」に関してはそれまでの見方が大きく転換したのである。この時の経験から、SEA 関西でアスペクト指向の分科会を企画した時、私もこれにならって「原論文」に当たるべきだと思い”Aspect Oriented Programming”と題されたパロアルト研究所の論文（参考文献[1]）と、関係する何本かの論文にあたってみた。（源流を知るという意味では、前記の”Generative Programming”の解説も参考になったが、現在の AspectJ をはじめとするアスペクト指向に関しては、やはりパロアルトの研究成果が参考になると思う。）

最初の論文では、アスペクト指向プログラミングの最初の応用といえるシステム開発の紹介から始まり、アスペクト指向プログラミングの基本的な概念と用語を定義している。また、アスペクト指向が単独で成立するプログラミング・パラダイムではなく、構造化プログラミングやオブジェクト指向プログラミングのパラダイムと相補うものだとしている。（この論文では、アスペクト指向の観点からは「構造化」と「オブジェクト指向」の区別を意味があるものとはしていない）

この論文では、画像処理ソフトウェアを例にして、アスペクト指向における重要な概念である「クロスカッティング」を解説しているのだが、ソフトウェアを2つの異なる観点（「機能分割」と「実行時のデータフロー」）から見た場合に、まったく異なる構造が現れて相互に対応しないこと、それぞれの構造は、システムの持つ特定の品質属性（「保守性」と「性能」）を実現するのにそれぞれ重要であることが述べられている。



機能分割の構造



データフロー構造

このように「抽象的なレベルの構造」同士の間での「クロスカッティング」を問題にすることは、私には非常に重要だと思われた。現在の AspectJ や Java のフレームワークにお

けるアスペクト指向の応用目的が、ソースコードレベルでのクロスカッティングを中心に議論されているように思われるが、これがもともとはアプリケーションのドメインモデルの構造と、トランザクション実行時の処理フローのパターンとの間の構造的なクロスカッティングが、ソースコードのレベルで現れていると考えることができるからである。より抽象度の高いレベルで問題を理解することは、開発のプロセスやソフトウェアのライフサイクルとアスペクト指向をうまく絡める上で重要だと思う。

さらにアスペクト指向の源流には、David Parnas の 1972 の論文 "On the criteria to be used in decomposing systems into modules" があるとも言われている。これをアスペクト指向の原典と言うのは少し言いすぎだろうが、アスペクト指向が解決しようとしている問題が表面的なものではなく、ソフトウェア工学上の伝統ある問題（「関心の分離 (Separation of Concerns)」）の解決を目指すものだということを認識することは意味があると思う。

4. フォーラムの内容について

さて、フォーラムでの講演については、増原先生、山根氏の秀逸な発表のプレゼンテーションが参照できると思うので、以下は、討論会の内容（傾向）と、私自身のアスペクト指向とのかかわりと思いを述べたいと思う。

パネル討論では、講師を務められた増原先生、山根氏の他、玉井先生と私の 4 人がパネラーとして前に座った。（正直に言うと、この 3 人のような優れた実績を持つ方々の間に私のような単に興味本位の者が座るのは、少々居心地の悪い気もしたのだが。）

東京と大阪とも時間いっぱいの議論が交わされたが、議論された内容については東京と大阪とで傾向の違いが見られたように思う。

東京での討論においては、どちらかと言うと、AspectJ の言語仕様や実装など技術的な内容に議論が集中していたように思われた。それだけ東京の参加者の方々の言語や方法論に関する具体的な知識や見識を持った方々の参加が多く、最初から高度な問題意識に基づいた議論になっていたように思う。一方で技術論に集中することでやや「アカデミック」寄りの印象を受けた参加者もいたようである。

それに対し、大阪では東京に比べると導入の戦略や、組み込み系など AspectJ とは別の分野への応用の可能性が話題になっていた。これは大阪という土地柄もあろうかと思われるが、組み込み開発の関係者の参加が多かったことにもよると思う。実際、討論会の場で組み込み関係の方々がどのくらい参加されているか挙手を求めたところ、1/4 から 1/3 の方々の手が上がっていた。

私自身は、どちらかというとはり大阪タイプで、AspectJ の技術的な詳細よりも、手法としてのアスペクト指向の可能性や課題に関心があったため、討論では以下のような問題提起を行ってもみた。

(1) アスペクトの複合について

新しい手法や方法論の適用でいつも気になるのは、現実の課題への適用力、特にスケーラビリティである。アスペクト指向で以前から気になっていることのひとつに、多数のアスペクトが同じプログラムに適用された場合のアスペクト間の相互作用が、正しく扱えるかどうかということがある。

これに関してはオブジェクト指向の「多重継承」をいつも連想するのだが、当初オブジェクト指向の優れた特徴と考えられていた「多重継承」が、最近システムの複雑化に伴ってクラスの継承構造が複雑化した場合の弊害のため、排除される傾向にある。(Java における多重継承の廃止などが例)

アスペクト指向でも、数少ないアスペクトをシステム全体に均一に適用できる場合はよいが、プログラムの同じ箇所に複数の性格の異なるアスペクトを編み込む場合や、アスペクトの実装に対し別のアスペクトを適用するような場合をうまくコントロールできるだろうか？

(2) 高レベルな記述によるジョインポイントの明示化

現在の AspectJ などの処理系は、Java などの汎用のプログラミング言語の文法構造のレベルでジョインポイントを指定する形をとっている。これは非常に柔軟性が高いやり方で、ウィーバーも応用分野に関わらずに一本化できるので、効率のいいやり方なのだが、実際には汎用プログラミング言語よりも高い抽象レベルにおいて初めて見出されるようなジョインポイントもあるのではないかと思う。

例えば、パロアルト研究所の最初の論文で紹介されていた画像処理システムの例では、ジョインポイントを明確にするためのシステム固有の高レベルな構文を定義して使用している。

最近、ソフトウェア・アーキテクチャやプロダクトライン・エンジニアリングの分野でドメイン固有言語 (Domain Specific Language あるいは DSL) を重要視する動きもあるが、個人的には、これらとアスペクト指向との関わりにも興味があるところである。

この問題については、増川先生から応用分野毎にウィーバーを自作することのコストを指摘され、汎用のウィーバー上で分野向けのアスペクトのライブラリを構築するのが現実的ではないかとのご意見をいただいた。

(3) 組み込みにおけるアスペクト指向について

前に述べたように大阪のフォーラムでは組み込みソフトウェア開発関係者の参加が目立った。現在アスペクト指向の応用は処理系の開発が先行していることもあって、Java によ

る（業務系）サーバー・アプリケーションに集中している感があるが、組み込みソフトウェアは以下のような特徴を持っていることから、アスペクト指向の応用の潜在的価値は高いと思う。

製品「系列」を持つ

同一系列内に多くの類似した製品があり、それぞれ少しずつ異なる機能や属性を持つ。

系列内での再利用による効果が高い、しかし機能要件と非機能要件のコードの混在により再利用が難しい。（それぞれの要件の「関心」を分離する必要がある。）

テスト環境と開発環境の分離

製品機能とは別に「テスト用」の機能を実装する必要がある。

結果だけでなく処理の「シーケンス」「タイミング」の検証（モニタリング）が必要。

信頼性・安全性

機能要件とは別の「品質要件」の重要性の高さ（品質要件が開発コストに占める割合の高さ）

これらの要素が、組み込みソフトウェア開発においてソースコード上での「関心の混在」を招き、再利用性と生産性を低下させていると思われる。これに対しては、アスペクト指向による関心の分離が、実装面でサポートされれば非常に有効ではないだろうか？

5. システム開発における意味

アスペクト指向を単にプログラミング上のテクニックや便利なツールという以上に、ソフトウェア開発における方法論として捉えた場合、どのような意味があるのだろうか？ 私個人として最も大きいと感じるのは、プログラム開発上で頻繁に発生するコードの「クロスカッティング」が、個別の設計の悪さ・下手くそさに起因するのではなく、ある意味必然的な「構造的」な問題であることを明らかにした点にあると思っている。同じ対象を異なる関心を持って異なる観点から眺めると、別々の構造が見えてくる。しかもそれらの間には明確な対応関係が無い。構造的に対応しないのだから、いくらがんばってもうまくモジュール分割できる訳が無くどうしたってクロスカッティングしてしまう！これが見えたと、今までクロスカッティングは自分の修行が足りないのが原因だと思って苦しんできたことが、何か嘘のように飛んでいく。（飛んでいって欲しい！）

このようにソフトウェアを複数の観点から見た異なる構造の集合としてうまく理解しようというのは、最近のソフトウェア・アーキテクチャ論では必ず出てくる話である。カーネギーメロン大学の SEI におけるソフトウェア・アーキテクチャ研究（参考文献[6]）でも、Philippe Kruchten の「4 + 1 ビュー・モデル」（参考文献[9]）でも、複数のビューを用い

てソフトウェア・アーキテクチャを表現することは共通している。そしてそれぞれのビューの構造は見事にクロスカットする。

ソフトウェアを「大局的」に捉えるアーキテクチャ論とプログラミングから始まったアスペクト指向とは、実は同じ原理に動かされているようだ。その意味で、SEI のソフトウェア・アーキテクチャ研究の主導者の一人である Paul Clements が、上流フェーズからのアスペクト指向を推進する研究者団体 Early Aspects (参考文献[5]) のメンバーであることには注目される。

もうひとつ、意外なところに「クロスカッティング」の存在を指摘した人がいる。あのユースケース駆動開発の Ivar Jacobson である。Jacobson の最近の著書に”Aspect-Oriented Software Development with Use Cases”というのがあるが、この中で彼はユースケース駆動開発とオブジェクト指向開発の意外な(?) ミスマッチについて述べている。あるひとつのユースケースを実現するためには複数のクラスのコラボレーションが必要である一方、あるひとつのクラスは複数のユースケースの実現に貢献している。え？ユースケースとオブジェクトがミスマッチ？ユースケース駆動開発というのはオブジェクト指向開発の一部ではなかったの？

Jacobson によれば、ユースケースという技法はアスペクト指向を目指す「関心の分離」を要求分析段階で実現する手法であり、「アスペクト」は原則的にすべてユースケースとして表現可能である。しかし従来はオブジェクトモデルを設計する時点でせっかく分離した関心を混ぜ合わせなければならず、これが以降の工程をスムーズに進めることができない原因になっている。だがアスペクト指向プログラミングがあれば大丈夫。ユースケースのそれぞれはアスペクトとして独立して開発し、最後にウィーバーによって自動的に編み合わせれば全体システムが出来上がります。アスペクト指向プログラミングによってユースケース駆動開発は完成する！

Jacobson は「ユースケース・スライス」という新しい概念を持ち出してこのあたりを詳細に論じており、詳しくは本を読んで欲しいが、主張はなかなか明快で説得力があり、目から鱗が落ちるような感じである。(今までが騙されていたような気がしなくもないが。) 気がついてみると、世の中のソフトウェアは「クロスカッティング」だらけではないか！どうして今まで気がつかなかったのだろう？そう思うくらい、実はクロスカッティングというのはソフトウェア開発では当たり前のことだったのだ。今まで見えなかったことが、名前が付くことで改めて見えてくることがある。

「人は語ることでできない事については沈黙しなければならない。」と言った人がいたが、アスペクト指向プログラミングによって定義された幾つかの概念と用語によって、ソフトウェアについて今まで語ることでできなかったいくばくかのことが改めて明晰に語るができるようになったと言うと、少し大げさだろうか？

6. 最後に

今回、玉井先生のご提案と増原先生、山根氏のご協力を得て、東京・大阪合わせて2日間、わくわくするような楽しい時間を過ごすことができました。深く感謝いたします。また、大阪での開催を薦めてくださった岸田事務局長。ありがとうございました。

参考文献

- [1] Aspect-Oriented Programming
by Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda,
Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin
<http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-ECOOP97/>
- [2] AspectJ in Action: Practical Aspect-Oriented Programming
by Ramnivas Laddad
Manning Publications (2003)
- [3] Generative Programming: Methods, Tools, and Applications
by Krzysztof Czarnecki, Ulrich Eisenecker, Krzysztof Czarnecki
Addison-Wesley Professional (2000)
- [4] Aspect-Oriented Software Development with Use Cases
by Ivar Jacobson, Pan-Wei Ng
Addison-Wesley Professional (2004)
- [5] The Main Page of Early Aspects.net
<http://www.early-aspects.net/>
- [6] Software Architecture in Practice, Second Edition
by Len Bass, Paul Clements, Rick Kazman
Addison-Wesley Professional (2003)
- [7] Architectural Blueprints — The "4+1" View Model of Software Architecture
by Philippe Kruchten
IEEE Software 12 (6) November 1995, pp. 42-50
- [8] On the criteria to be used in decomposing systems. into modules
by David.L. Parnas
Communications of the ACM, Vol. 15, No. 12, December 1972 pp. 1053 - 1058
<http://www.acm.org/classics/may96/>

AOP入門

増原英彦 (masuhara@acm.org)
東京大学 大学院 総合文化研究科

SEAフォーラム 2006年6月

1

概要

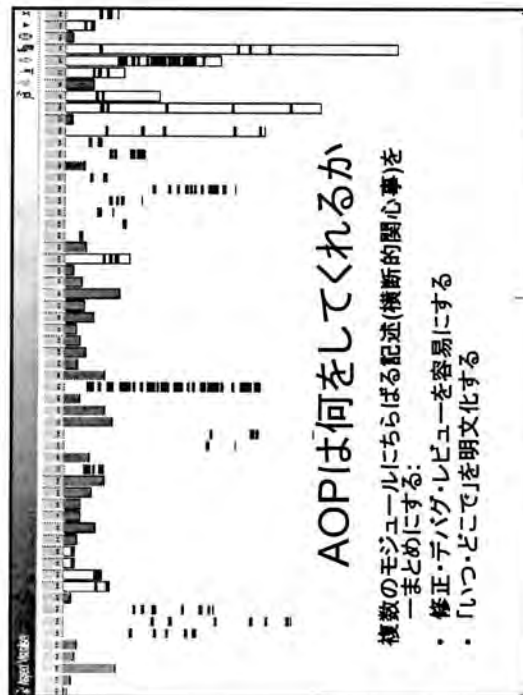
- AOPでやりたいこと
- AOPでできること
 - 現在の状況
 - AspectJの紹介
 - 応用例
- 他のAOP言語

2

AOPは何をしてくれるか

複数のモジュールにちらばる記述(横断的関心事)を一まとめにする:

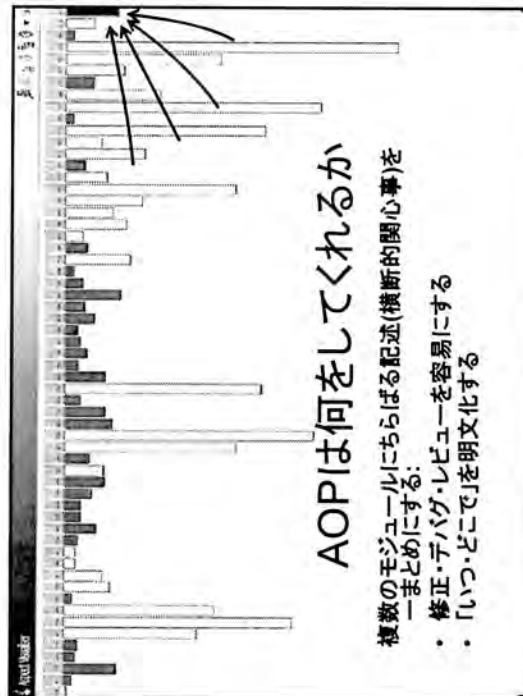
- 修正・デバッグ・レビューを容易にする
- 「いつ・どこで」を明文化する



AOPは何をしてくれるか

複数のモジュールにちらばる記述(横断的関心事)を一まとめにする:

- 修正・デバッグ・レビューを容易にする
- 「いつ・どこで」を明文化する



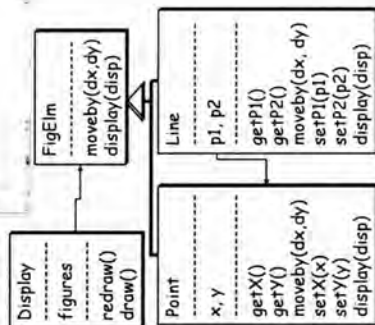
AOPとは: 具体例

- 例: 図形エディタの内部のモジュール化
 - モジュール: 図形要素(点・線・円等)、画面、命令(生成・移動・削除)など
 - with OO: モジュール=クラス
 - 効用
 - 拡張・修正・デバッグ範囲の限定 (例: 図形要素の追加)
 - ポリシーの明文化
- ※ アスペクトもモジュール

5

具体例: 図形エディタ

- ディスプレイの定義は図形要素の詳細を知らずにできる
- 図形要素の種類を増やすときの手間が小さい

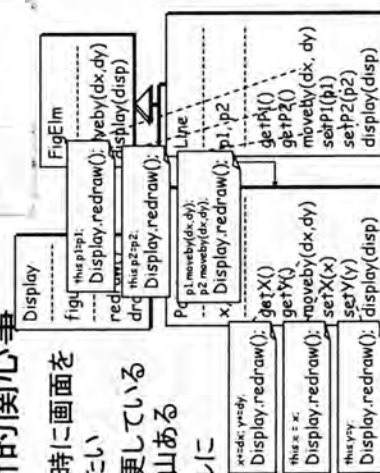


6

問題: 散らばる記述 aka 横断的関心事

- 図形変更時に画面を再描画したい
- 図形を変更している場所は沢山ある
- モジュールにまとめられない!

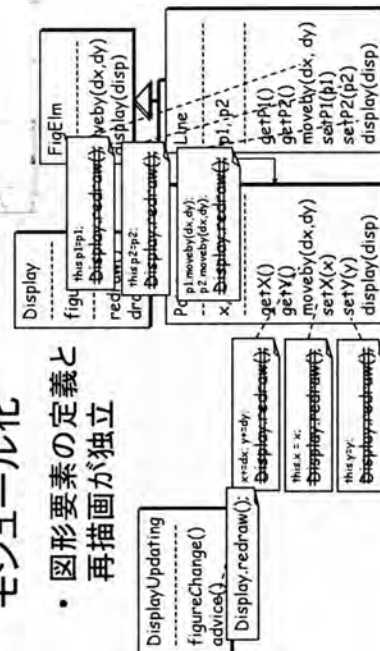
※ほんの一例



7

アスペクトによる モジュール化

- 図形要素の定義と再描画が独立



8

AOPの現状: 道具

- 実用的な処理系が揃いつつある
 - AspectJ, AspectC++
 - AspectWerkz, JBoss AOP, Spring AOP
- 開発環境もある
 - AJDT plug-in for Eclipse
- ライブラリはこれから
for more:
 - community wiki @ aosd.net
 - AOP@Work (IBM developerWorks)

9

AOPの現状: 使われ方

- 3種の使われ方 [Kiczales]
 - exploration / enforcement
 - auxiliary / infrastructure
 - core / business
- 状況: 実験的使用 ~
実システムでの部分的使用
– 商用システムでも (e.g., IBM SWG)

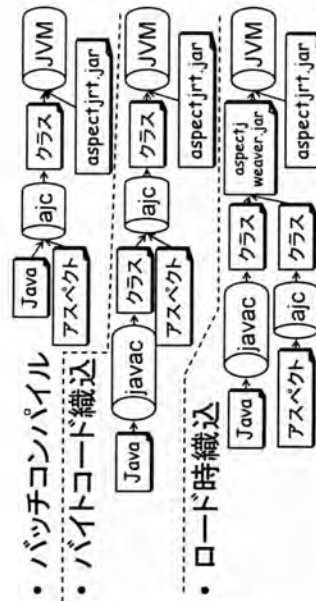
10

AspectJ言語

- 最も知られている汎用AOP言語
 - 他のAOP言語の手本
- Java言語と上位互換
- 開発体勢:
 - Xerox PARCで開発がスタート('90後)
 - Eclipse オープンソース開発 (現在)
- 開発環境: Eclipseプラグイン等

11

織込と実行



12

AspectJの主要概念

- アスペクト (cf. クラス)
横断的関心事をまとめる単位
- アドバイス (cf. メソッド)
追加的な操作
- ポイントカット
「どんなとき」を決める
- 型間宣言 — 追加的宣言 — 構造

動作
結合点

13

AspectJの主要概念

- アスペクト
- アドバイス
- ポイントカット
- 型間宣言

```
aspect DisplayUpdating {
    pointcut move() :
        call(int FigElm.moveby(int,int)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point));
    after() : move() { Display.redraw(); }
    void FigElm.draw(Display d):
    void Point.draw(Display d) { ... }
    ...
}
```

14

AspectJの主要概念: アドバイス

追加・代替の動作を記述

- どんな動作の(ポイントカット)
- 前/後/かわりに(修飾子)
- 何をするか(本体)
— Javaの文

```
aspect DisplayUpdating {
    pointcut move() :
        call(int FigElm.moveby(int,int)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point));
    after() : move() {
        Display.redraw();
    }
    void FigElm.draw(Display d):
    void Point.draw(Display d) { ... }
}
```

moveした後は
redraw()を呼べ

15

AspectJの主要概念: ポイントカット

何かが起きたときを
指定

- 動作の種類
(メンソッド呼出etc.)
- シグネチャ
- 合成

```
aspect DisplayUpdating {
    pointcut move() :
        call(int FigElm.moveby(int,int)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point));
    after() : move() { Display.redraw(); }
    void FigElm.draw(Display d):
    void Point.draw(Display d) { ... }
}
```

FigElm.movebyまたはPoint.setXまたは...を呼出すとき

16

AspectJの主要概念: 型間宣言 (inter-type declarations)

既存の型に外から
宣言を追加

- 既存の型
 - クラス・インタフェース
- 追加される宣言
 - メソッド・フィールド
 - extends・implements節

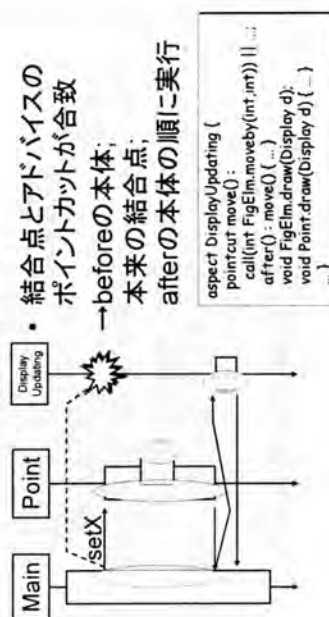
```
aspect DisplayUpdating {
    pointcut move() :
    call(* FigElem.moveby(int,int)) || ...;
    void FigElem.draw(Display d);
    void Point.draw(Display d) { ... }
}
```

declare parent MyTask;
implements Runnable;
public void MyTask.run() { init(); }

MyTaskクラスに
Runnableを実装

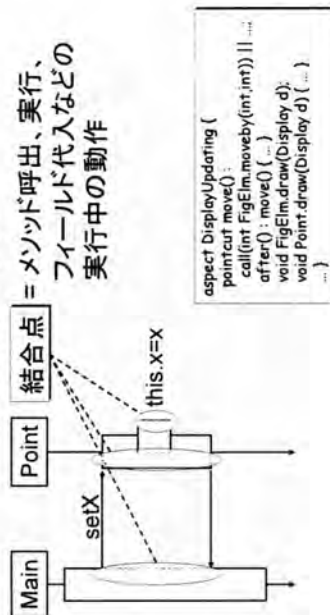
17

AspectJの実行モデル: 結合点モデル



19

AspectJの実行モデル: 結合点モデル



18

AspectJの実行モデル: aroundアドバイス



20

ポイントカットの能力: 横断的な指定

```

aspect DisplayUpdating {
  pointcut move() :
    call(int FigElm.moveby(int,int)) ||
    call(void Point.setX(int)) ||
    call(void Point.setY(int)) ||
    call(void Line.setP1(Point)) ||
    call(void Line.setP2(Point));
  after() : move() { Display.redraw(); }
}

```

FigElm・Point・Lineに
対する動作を
まとめて指定
↓
モジュール化!

21

ポイントカットの能力: ワールドカード + 命令規則

```

aspect DisplayUpdating {
  pointcut move() :
    call(int FigElm.moveby(int,int)) ||
    call(* FigElm+.set*(..));
  after() : move() { Display.redraw(); }
}

```

- ワールドカードによって簡潔に定義できる
- 例: FigElmとその子クラスにある "set" で始まる名前のメソッドが呼び出されたとき

22

ポイントカットの能力: 様々な条件

- 色々な種類の条件
 - within(myapp.db.*): DBパッケージ内のみ
 - set(int Point.x): Point.xへの代入
 - withincode, execution, get, handler, initialization, static initialization
- 組み合わせる
 - call(* javax.swing.*(..))
 - && !within(myapp.ui.*): UIパッケージ以外からのSwing呼出し

23

ポイントカットの能力: 文脈情報(self)の取得

```

aspect DisplayUpdating {
  pointcut move(FigElm fig) :
    (call(int FigElm.moveby(int,int)) ||
     call(void FigElm+.set*(..)))
    && target(fig);
  after(FigElm fig) : move(fig) {
    Display d = fig.getDisp();
    d.redraw(fig);
  }
}

```

画面更新の例

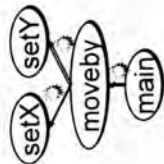
- 各図形に表示画面が対応
- アドバイスは「変更される図形」から画面を取得して再描画

呼出先のオブジェクト: fig

24

ポイントカットの能力: 制御の流れ(cflow) 1

- p.moveby(2,3)は
3回redrawを
呼んでしまう



```

aspect DisplayUpdating {
  pointcut move() :
    call(int FigElem.moveby(int,int)) ||
    call(* FigElem.set*(..));
  after() : move() { Display.redraw(); }
}

```

```

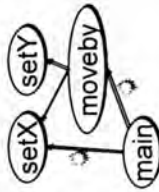
class Point ... {
  void moveby(int dx, int dy) {
    setX(getX()+dx);
    setY(getY()+dy);
  }
}

```

25

ポイントカットの能力: 制御の流れ(cflow) 2

- 呼出スタックを
調べられる →
「setX, setYが
movebyなどから呼出
されていないとき」



```

aspect DisplayUpdating {
  pointcut move() :
    call(int FigElem.moveby(int,int)) ||
    call(* FigElem.set*(..));
  after() : move() &&
    !cflowbelow(move()) { ... }
}

```

```

class Point ... {
  void moveby(int dx, int dy) {
    setX(getX()+dx);
    setY(getY()+dy);
  }
}

```

26

再利用: 抽象アスペクト

- 抽象アスペクト
 - 「どこ」= 抽象
ポイントカットとして
定義を遅らせる
 - 動作 = アドバイス
に定義する
- 具体アスペクト
 - ポイントカットを
具体化するだけ

```

abstract aspect AbstractLogging {
  abstract pointcut logPoint();
  after() : logPoint() {
    ... ログ取り動作 ...
  }
}

```



```

aspect DBLogging
  extends AbstractLogging {
    pointcut logPoint():
      call(* myapp.db.*.*(..));
  }
}

```

27

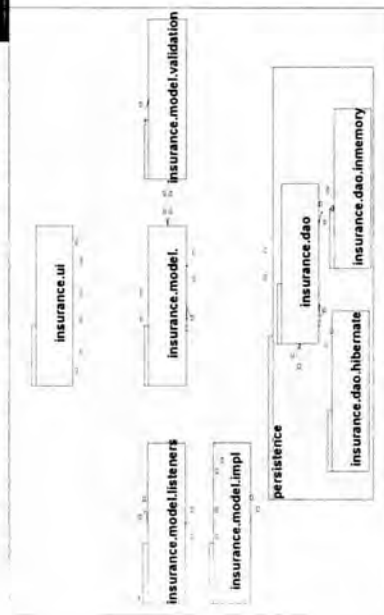
AOPの応用例

- Architecture enforcement
 - API Scanner @ IBM SWG
- 例外処理

slide courtesy of Gregor Kiczales

28

Architecture Enforcement



29

Architecture Enforcement – Modules

```
public aspect ArchitectureEnforcement {
    public pointcut uiCall():
        (call(* insurance.ui..*(..))
         || call(insurance.ui..new(..))
         && !call(* java.lang.Object..*(..)));
    public pointcut modelCall():
        <similar>;
    public pointcut modelImplCall():
        <similar>;
    ... one per module ...
}
```

30

Architecture Enforcement – Modules

```
...
public pointcut inUI():
    within(insurance.ui..*);

public pointcut inModel():
    within(insurance.model.*);

public pointcut inModelImpl():
    within(insurance.model.impl..*);

... one per module ...
```

31

Architecture Enforcement – Rules

```
...
declare warning: uiCall() && !inUI():
    "No calls into the user interface";
declare warning: modelImplCall() && !inModelImpl():
    "Please use interfaces in insurance.model instead";
declare warning: daoCall() && !inModelImpl() ||
    inAnyDAO():
    "Only model and DAO implementers should use DAO interface";
```

- プログラマブル・柔軟性が高い
- 1つのツールであらゆるアーキテクチャを扱うのは難しい

32

Architecture Enforcement At Work



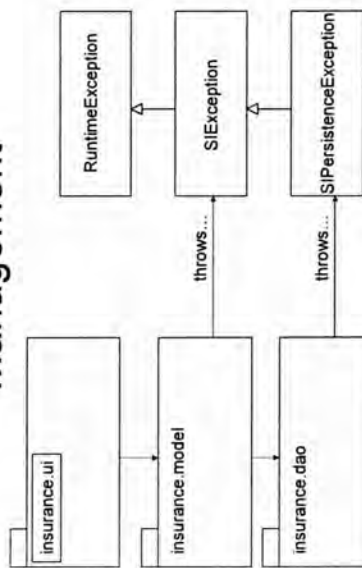
33

Example: API Scanner

- IBM SWGで利用されているアスペクト群
 - 30のプロジェクト/製品で利用
 - 良くない場合をdeclare warningで検知
 - 1つのプロジェクト/製品は他のを使用する環境
 - 例: publicだけ外部/IFではないメンソッドの呼出
- 配布しやすいスクリプトにまとめられている
- 50,000以上の問題点を発見
 - 設計のモジュール性を尊重すれば自由度が上がる
 - 将来の製品開発の時間短縮の可能性

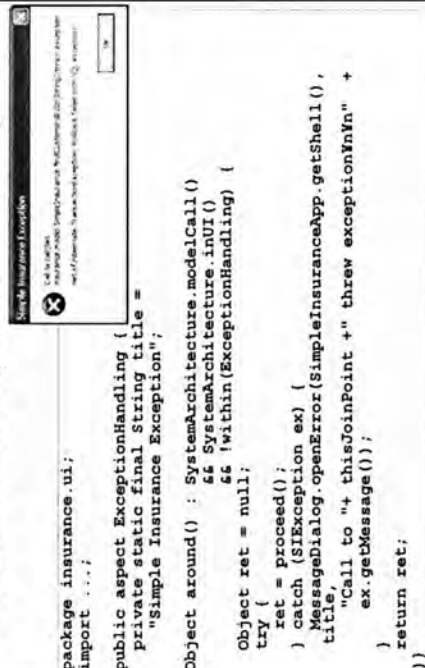
34

Example: Exception Management



35

Example: Preliminary Exception Management



36

AOPはどんな場合に使えるのか?

- 「いつそれをやるか」が単純でない場合
 - 特定の動作だけログをとる
 - このモジュールだけ例外を別扱い
 - かつ規則化したい場合
 - 将来の拡張・変更のため
 - カズくで100箇所コードを書くと後で泣く
- そうでなければOOでrefactoringすべき

37

他のAOP言語

(実用に耐えられそうなもののみ)

- Javaアプリケーションフレームワーク
 - 特徴: pure Javaの記述 (メタファイル・アノテーション)
 - ライブラリ: 永続化・キャッシュ等
 - 処理系:
 - AspectWerkz (BEA) — AspectJと共通化
 - JBoss AOP
 - Spring AOP — AspectJと共通化?
- Java以外: AspectC++

38

まとめ

- AOPとはモジュール化の手法
- 使えるところから使われている
- AspectJの紹介
 - 結合点・ポイントカット・アドバイス
 - 型間宣言
 - 応用例
- 他のAOP言語: 共通化が進んでいる

39

あなたの知らないAOP

有限会社スーパーグラフィックス
山根 英次
yamane@super-gs.jp
<http://d.hatena.ne.jp/eyamane/>

AOPと聞いて...

- Spring Frameworkや、Seasar2のようなDIコンテナとの併用で十分
- トランザクション制御やロギングを楽にするだけのもの
- 所詮流行もの

上記問いに1つでも「Yes」が1つ以上ある人
あなたをAOPの未知の世界へ誘いましょう。。。

Agenda

- AOP With DIContainerの限界
- Past AOP
- Present AOP
- P(rogramming)からD(esign)へ
- future AOP
- まとめ

Sampleの動作環境

OS	Windows XP SP2
JDK	JDK 1.5.0_06
Eclipse	3.1.2
AJDT	1.3.1(Aspect J 1.5.1)
Ant	1.6

ソースコード&実行環境は以下のURLより入手可能です。

ソースコード: <http://super-gs.jp/~yamane/archive/seminar2006AOP-src.zip>

実行環境: <http://super-gs.jp/~yamane/archive/seminar2006AOP-rt.zip>

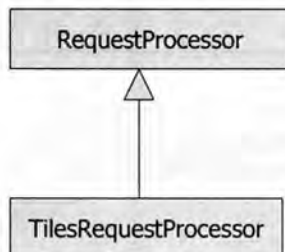
AOP With DI Containerの限界

- ジョインポイントが少ない
 - publicなオブジェクトメソッドのみ
 - TemplateMethod/パターンへのWeavingは実質不可能
- DI Container外で生成されたオブジェクトには(当然)適用されない
 - TomcatやStruts、DI Containerと言った3rdパーティコンテナに対するConcern編み込みが難しい
- InterType定義が行えない(*1)

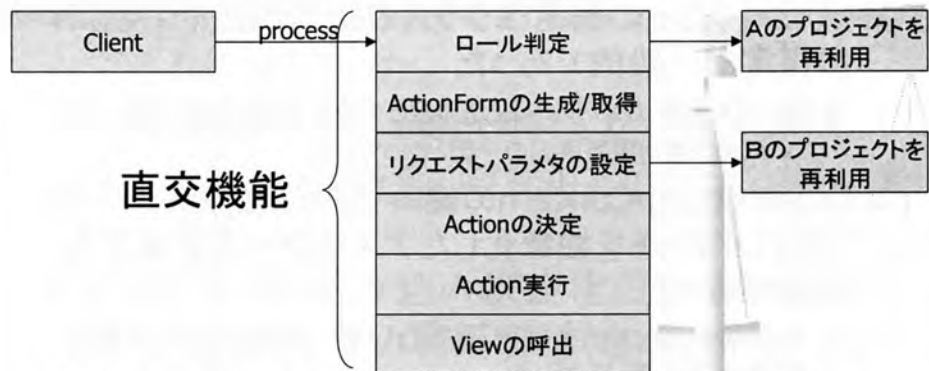
*1:Seaser2 2.4からは出来る

困った事例その1. StrutsのRequestProcessorの拡張

- 拡張性が低いことで有名(?)なクラス
- でもFrontControllerであるがゆえに拡張することが多いクラス
- しかも。。。
 - 最初から継承したクラスが用意されている!!!
 - TemplateMethodパターンで実装されているのでDIコンテナ経由でConcernをWeavingしても効果が薄い
 - 機能が豊富であるゆえ機能の取捨選択が求められる。



RequestProcessorの振る舞い



- プロジェクト横断的にモジュールを利用するために各振る舞い毎にAspectを定義する

サンプルコード

```
package jp.gr.java_conf.e_yamane.aop2006.struts;

import org.apache.struts.action.RequestProcessor;

public aspect ProcessRoleAspect {
    pointcut processRoles() :
        execution(protected boolean
            RequestProcessor+.processRoles(..));

    Object around() : processRoles() {
        System.out.println(">> processRoles");
        try {
            return proceed();
        } finally {
            System.out.println("<< processRoles");
        }
    }
}
```

Past AOP(AspectJ)

- ポイントカットの指定はクラスの継承関係や名前での指定が一般的であった
 - 編み込まれていないことを検証することが困難であった。
 - AOPの適用阻害要因の一因となっていた
- Crosscutting-Concernの編み込みは抽象アスペクト(ポイントカットを抽象化したアスペクト)を定義する必要があった
 - 利用者(Concernを編み込みたい側)がAspectの存在を意識する必要があった。
 - AOPの適用阻害要因の一因となっていた

サンプルコード

```
package jp.gr.java_conf.e_yamane.aop2006.performance;
abstract public aspect PerformanceMeasureClassicAspect {
    abstract pointcut measurePoint();
    Object around() : measurePoint() {
        long start = System.currentTimeMillis();
        try {
            return proceed();
        } finally {
            long finish = System.currentTimeMillis();
            System.out.println(finish - start);
        }
    }
}
```

```
public aspect PerformanceMeasureClassicAspectImpl
    extends PerformanceMeasureClassicAspect {
    pointcut measurePoint() : execution(* PerformanceMeasureClassic.foo());
}
```

Present AOP AspectJ 5(+tiger)では。。。

- アノテーションに対してのポイントカットが適用できるようになった。
 - 編み込まれている箇所の特定が容易になった
 - 利用者側への技術的障壁が下がった
- アスペクトの編み込み決定権限をAspectJの表記法を知らない利用者に(も)委ねることが出来る！！

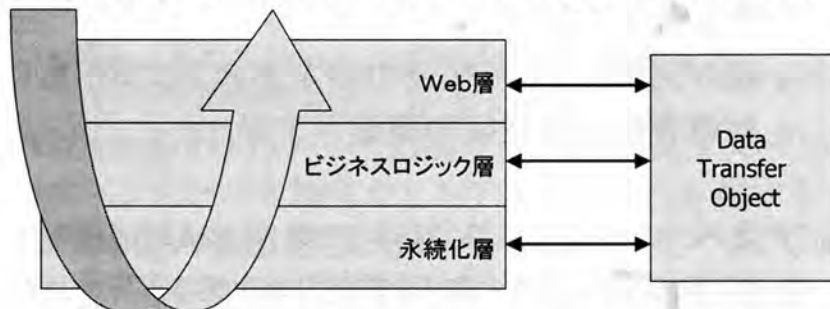
サンプルコード

```
public aspect PerformanceMeasureAspect {
    pointcut measurePoint() : execution(@PerformanceMeasurePoint * *.*(..));
    Object around() : measurePoint() {
        long start = System.currentTimeMillis();
        try {
            return proceed();
        } finally {
            long finish = System.currentTimeMillis();
            System.out.println(finish - start);
        }
    }
}
```

今後、ポイントカットは
アノテーションが主流となる！？

P(rogramming)からD(esign)へ Pure OODへの第1歩

- 今の設計ってオブジェクト指向でしょうか？



ぶっちゃけ構造化設計じゃないの？

何故こんな世界に行き着いたか？

- 各層は個別に差し替え可能である(べきだと言う再利用的な要件)
- 各層の処理を1つのクラスに組み込むと1つのクラスの責務が多くなりすぎる
- 一般的なJavaプログラミング(の多くの部分)はオブジェクト指向ではない
- だから概念モデルと実装モデルが剥離している
 - トレーサビリティ性の低下
 - 品質の低下

インタフェース再考

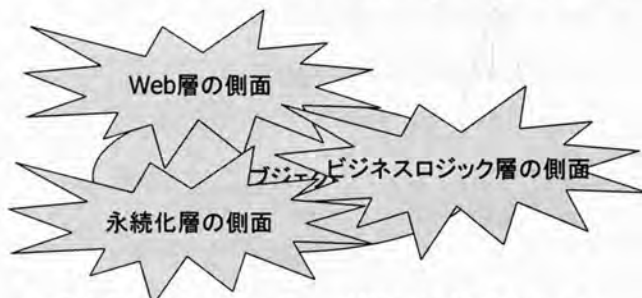
- インタフェース＝クラスの責務
- 本当だろうか？



同僚
インタフェースに対するレスポンスは、オブジェクトの状態だけでは無く、呼び元によって決定される！？

Aspectという言葉

- 呼び元(視点)によって振る舞いが異なったり、そもそもの振舞いが生成されたり。。。。
- これってまさに側面 (Aspect)。。。だよなぁ？



AOPは分離したAspectをコード的には分離させつつモジュール統合をするためのフレームワーク！！

レイヤ分割

例: View用の振る舞いを Weavingする

- 数値をカンマ区切りで表示するという要件は業務系のシステムでは多々あります(例: 1000=1,000のように)
- 一般的には呼び出し側で数値から文字列への変換を行う
- 微々たる物だが、利用者の視点に立つと若干不便と言える。

サンプルコード

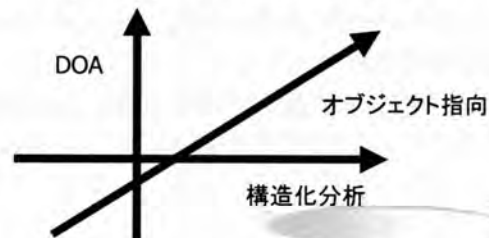
```
public aspect NumberCommaStringAspect {
    declare parents: @Entity * implements Dummy;

    private final static DecimalFormat DF = new DecimalFormat("#,##0");
    public String Dummy.getCommaString(String name) {
        try {
            Object o = PropertyUtils.getProperty(this, name);
            return DF.format(o);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    private interface Dummy { }
}
```

Future AOP

AOPは本流と成りうるか？(1)

- AOPは時間はかかるが本流となると考える
 - これまでの設計手法は、視点の違いはあれど次元の分割でしかなかった
 - AOPは、オブジェクト指向設計によって分割されたモジュールをさらに横断的にConcernの分離が可能
 - より重複の少ない分離が可能となる



Future AOP

AOPは本流と成りうるか？(2)

- これまでの設計手法は、分割することが主題
- 高度に分割されたモジュールはしばしばクライアント側で大量の結合コードを必要とする
- AOPはWeaverありきの概念であり、分割はもとより、結合にも主眼が置かれている
 - 単純な委譲ロジックはかなり削減されるはず

Future AOP

AOPは本流と成りうるか？ (3)

- 本当に??
- まだまだ前途多難です。
 - 認知度、設計ノウハウ共にまだまだ不足
 - 機能的にも不足(特にInter-Type宣言周り)
 - クラスにgenericsを用いたメソッドをWeavingしてjarを作ると、genericsの情報が落ちる
 - AJDTだと、Inter-Type宣言したメソッドにはエラーマークが付く
 - Inter-Typeの自由度が低い
 - 設計レベルからAspectを考慮するならInter-Type宣言は必須なので、もうちょっと自由度が高まらないと厳しいですね。

まとめ

- 正直、DI ContainerとセットとなったAOPの話題や、non-functional Concernの話題はそろそろ飽きてきてますよね？
- AOPは次のステージでディスカッションされる時期に来ていると思います。
- みんなで、functional Concernの実装(=AOD)を模索していきましょう。
- 最後に。。。。。

「全部知ってるAOPだよゴルァ！！」
って人、いたらごめんなさい。m(_._)m

SEA Forum May on AOP 参加者の声

Forum に参加した何人かの方々から寄せられた感想です

No.1: AOP についてはほとんど知識がありませんでしたが、自分なりに理解することができ、有意義でした。応用例のお話などからすると、使普及はもう少しかかりそうな印象です。今後も定期的にこのようなフォーラムに参加したいと思います。

No.2: AOP に関する基本的な概念が、このフォーラムに参加し、質問することで、よりはっきり理解できました。具体的にいうと、

1. 横断的関心事のうち、ポイントカットやアドバイスの起動タイミングで記述できるものだけを AOP が対象としている
2. 一貫した名前付けをすることが、ポイントカット指定する上で重要であるということです。チュートリアルは一時間という短時間でしたが、内容の濃いものだったと思います。もう少し時間を取って、AOP の実装方式について触れてもらえるとういと思いました。

No. 3: 仕事の都合で遅刻しましたが、プレゼン内容は非常に考えさせられるものがあり、面白かったです。

- ☐ オブジェクト指向プログラミングに特有というよりは、どんなプログラミングでも、書かれた記述とは独立したところで、共通処理が記述できるのは、魅力です。
- ☐ オブジェクトという抽象概念があるから、AOPも成立します。その視点からみれば、UNIXの「ファイル」概念(データ、プログラム、デバイス、プロセス、統計)もひとつの抽象ですから、「ファイル」の変化に対して、記述を独立してできる可能性もあるはずですね。(JAVAでOSを記述する試みもあるようです)単なるプロファイリングの域を超えて、コンピュータリソースの一元管理の可能性を感じました。その意味では、アプリケーション記法というよりは、OSの機能で実現すべき考えなのかもしれません。
- ☐ オブジェクト指向言語の、現実のソフト開発における欠点は、プログラムの「廃棄性」についての考察が足りないところにあると思っています。「再利用性」は一見便利ですが、再利用の単位を誤ると、プログラムの相依存や継承の複雑な網にとらわれて、現実には似たような、機能のはっきりしないオブジェクトの山になり、開発生産性が低下します。ソフト開発は、再利用性によって開発効率をあげるだけでなく、廃棄性によって常に過去にとらわれず新しいものを簡単に開発できる必要があるという2面性があると思います。AOPは使い方によっては、オブジェクト間の関係をさらに複雑にし、事態を悪化させる可能性があります。使い方による、というソフトはソフトとしての評価はどうか、考えさせられました。
- ☐ その意味では、JAVAは、ソフトウェアの階層構造のすべての分野で、簡単にソフトウェア単位=オブジェクトを簡単に作れすぎるといって、(あえて)欠点があると思っています。ソフトウェア単位をどう考えるかは難しい課題です。JAVAによる分業型開発の難しいところは、この「何でもできてしまう言語」をどう正しく機能を分類整理して、開発者にルールを徹底させるかにあります。このようなことは、プロジェクト全体が見えていないと難しいので、プロジェクトマネージャの能力の高さと、事前にプロジェクト内容が決まっている必要性が高いということです。ということは、JAVAでAgileやXP開発を指向するのは、かなり指揮者が優秀であることが前提で、これは現実には難しいことである

と感じました。(ゆえに高給取りがいるわけですが)オブジェクト指向プログラミングがメジャーになる以前は、OSレベルのプログラミング、言語をつかったアプリケーションのプログラミング、コマンドを使用したスクリプト記述、という3つの技術習得レベルに格段に差があったので、それぞれの役割が自ずから理解されていたと思います。「分かる」とは「分ける」ことだと言われていますが、現実のソフト開発においては、この3つのレベルを維持しながら各分野を発展させることが重要だと思っています。OS=モノリシック構造による、効率の追求、小さく、強く。コマンド=再利用性の高さの追求シェルスクリプト=廃棄性の高さの追求なんて、勝手に考えています。それなりの美しさがあると思っています。

No.4: ASPECT という用語を新しく導入したのは、そもそも何のためであるのか？ セッションの中でも言及されていたが、設計に使用できるのではないかというあたりが聞きたかった。いまさら言語仕様を聞いても。。

No.5: AOP は、横断的関心事を一箇所に凝集させるというメリットを持っているが、それ以外のことについては、これまで考えたことがなかった。しかし、他にもコーディング規約遵守の強制や、機能追加など、活用できるシーンが多いことを教えられたことは、非常に有意義だった。仕組みとして強力であるが故、諸刃の刃である点について、みんなが同じような感覚を持っていること、そしてそれを打破する決め手が未だないことが **AOP** の爆発的普及の妨げになっていると感じた。「**AOP** でパッチを当てるより、リファクタリングが重要」、「ポイントカットを複雑にするとテストが困難」といった、アンチパターンを知ることができたことが非常に有意義だった。そして、こうした情報が **AOP** の進化と普及に必要な基盤であると感じた。

No.6: あまり予備知識もなく(コンピュータ雑誌の情報程度)、ただただ興味本位で参加させていただけましたが、あまり話が高度にならず、非常にわかり やすかったと思います。特に、パネルディスカッションで、組み込み分野に関する話題が取り上げられたのは、思わぬ拾い物でした(私自身、組み込みが専門で、何か いい使い道はないかを考えていたもので)。またこうしたフォーラムに参加したいと思います。

画像問題のアスペクト

玉井哲雄

1 はじめに

2006年5月に東京で開かれたSEAフォーラム「アスペクト指向技術は実用に使えるか」で、私とともに企画を担当された小林修氏(SRA)が、パネル討論の際にアスペクト指向プログラミング(AOP)の源流と目されるGregor Kiczalesの論文[1]を取り上げて、議論の材料とされた。そこで私は、2001年11月に当時増原英彦氏(この日のSEAフォーラムの第一講演者)が文部省の在外研究でカナダのブリティッシュ・コロンビア大学(UBC)のKiczales氏のところに滞在していたのを訪れ、Kiczalesグループに向けてこの論文の内容に対して批判的な講演を行ったことを思い出した。フォーラムの場でその2001年に使ったスライドが発掘できたので、会場でそれを見せてどんな話をしたのかごく手短かに紹介した。

例によって岸田編集長からフォーラムの結果をまとめよという指令が来たので、これを機会にこのときの話を書き下ろそうと思い立った。実際、この話はUBCや仲間内の研究会や学生とのゼミで紹介したが、論文や読み物としてはどこにも発表していない。時宜を失したともいえるが、このAOPフォーラムの報告に紛れ込ませて、SEA会員の皆様のご批判の目に供するのの一興かと思う。

とここまで書いて、この話をどこにも出していないというのは誤りであることに気がついた。1999年1月に高知で開かれたウィンターワークショップ・イン・高知(情報処理学会ソフトウェア工学研究会主催)の参加表明論文(いわゆるポジションペーパー)に「局面指向プログラミングの批判的考察」と題して、4ページほどの小論を書いているのである[2]。しかし、書いた当人も忘れていたくらいだから、ほとんど読まれていないと思う。

以下、部分的にその小論を再利用しつつ、現在の目で見直した見直しを含めて、この話を書いてみよう。

2 どんな問題か

まず言うておかなければいけないのは、この時点では、AspectJはまだ存在していなかったということである。1995年頃からKiczalesなどのXerox PARCの研究者たちが、アスペクト指向プログラミングという概念を提唱していることはある程度知られていた。しかし、きちんとした論文はなかなか出版されず、技術レポートがWebに公開されているという状態がしばらく続いていたと思う。1999年のワークショップで槍玉にあげた論文も、そのようにしてWebから取ったものだった。実は、その内容をもとにした論文は、すでに1997年のECOOPで発表されていたのだが[1]、当時はうかつにもそれに気づいていなかった。

いずれにせよAspectJ以前だから、アスペクトの記述方法やそれをプログラムに織り込む手段は、問題ごとにアドホックに考えるという立場である。したがって、AOPの手法自身の説明も、とりあげた例題の性質に多分に、というよりきわめて強く依存している。

その例題とは、画像処理プログラムである。画像処理において、画面全体の画素ごとにビットとしての和や積をとったり否定をとったりする操作を定義し、それらの組合せでより高度な操作を実現するという方法をとると、個々の操作が分かりやすくなるだけでなく、それらの組み合わせによる高度な操作の意味も理解しやすい。しかし、それにより、基本演算が呼び出されるたびに画像全体にわたる画素情報を中間的に作り出す必要があり、データ領域に無駄が生じるだけでなく、計算速度にも大きな悪影響が及ぶ。

基本操作の概要を、以下に示す。原論文では LISP 風の表記で記述されているが、ここでは Java 風を書く。まず、画像を表現したデータを Image というクラスで表す。その仕様は次のようである。

```
class Image {
    Image(int w, int h) :
        幅 (画素数)w, 高さ (画素数)h の画像データを
        生成する。初期値はすべて白 (ビット 0)。
    int width() : 幅を返す。
    int height() : 高さを返す。
    Pixel getPixel(int i, int j) :
        座標 (i,j) の画素の値を返す。
    void setPixel(int i, int j, Pixel p) :
        座標 (i,j) の画素の値を p とする。
}
```

ここで、画素 (クラス Pixel) は白黒の 2 値データとし、ビットと同一視する。画素に対し単項演算 not, 2 項演算 and と or が通常のビット演算として定義されている。Image は要するに、ビットの配列と思えばよい。画面全体に対する and 演算は、Image のクラス・メソッドとして次のように定義される。

```
public static Image and(Image a, Image b) {
    Image x = new Image(a.width(), a.height());
    for (int i=0; i<a.height(); i++)
        for (int j=0; j<a.width(); j++)
            x.setPixel(i, j,
                Pixel.and(a.getPixel(i, j), b.getPixel(i, j)));
    return x;
}
```

ここで and 演算をクラス・メソッドとして定義しインスタンス・メソッドにしなかったのは、その方が原論文の Lisp 表現に近いからである。まったく同様に、画像の 2 項演算 or, 単項演算 not が定義される。さらに、画像全体を 1 行上に上げる演算 up を次のように定める。

```
public static Image up(Image a) {
    Image x = new Image(a.width(), a.height());
    for (int i=0; i<a.height()-1; i++)
        for (int j=0; j<a.width(); j++)
            x.setPixel(i, j, a.getPixel(i+1, j));
    return x;
}
```

同様に、全体を 1 行下に下げる演算 down も定義される。

さて、これらの基本演算 (原論文ではこれらをコンポーネントと呼んでいる) を組み合わせて、次のような複合的な演算を定義することを考える。

```
public static Image remove(Image a, Image b) {
    /* a から b の像を取り除く。 */
    return Image.and(a, Image.not(b));
}
```

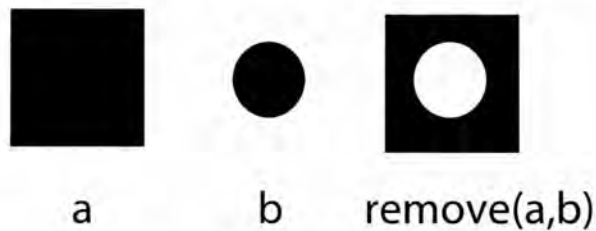


図 1: 関数 remove

```
public static Image topEdge(Image a) {
    /* a の上縁を求める. */
    return Image.remove(a, Image.down(a));
}
public static Image bottomEdge(Image a) {
    /* a の下縁を求める. */
    return Image.remove(a, Image.up(a));
}
public static Image horizontalEdge(Image a) {
    /* a の上下の縁を求める. */
    return Image.or(Image.topEdge(a), Image.bottomEdge(a));
}
```

図 1 に関数 remove の様子を, 図 2 に関数 edge の様子を示す.

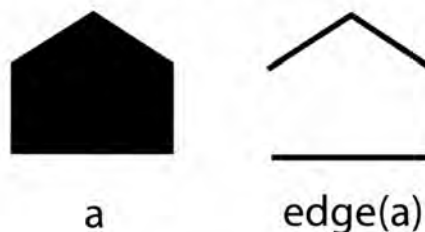


図 2: 関数 edge

これらの定義は分かりやすいが, 確かに効率が悪い. ループ計算が何重にも起こり, その度に, 中間的な配列構造が作られる. しかし, 効率を良くするようなコードを埋め込むと, モジュール構造が見にくくなる. そこで AOP では, ループを統合するという変換操作をアスペクト・プログラムとして別に記述する. その記述法は, 図 3 のようである. 細部は重要でないが, これはもとのプログラムのデータの流れがグラ

```
(cond ((and (eq (loop-shape node) 'pointwise)
            (eq (loop-shape input) 'pointwise))
      (fuse loop input 'pointwise
        :inputs (splice ...)
        :loop-vars (splice ...)
        :body (subst ...))))
```

図 3: アスペクト・プログラム例

フとして表現されていることを前提として, そのグラフで結合関係にある 2 つの節点 (node) が, 同じよう

なループを持っている場合に、それを融合するという操作を表している。このような操作が可能のようにデータフロー・グラフを作っておく必要があるが、実はそのためにもとのプログラムにも新しい記述方法を導入してある種の変更を加えている。

このやり方が、AspectJ 以前という時代性を表している。つまり、ここで用いられるアスペクト記述は、この問題に特化した特別な記述法なのである。また、そのアスペクトを元のプログラムに編み込む方法も、この問題用に特別に作られたものである。それで初めて、効率的なプログラムが生成される仕組みが得られるわけである。

しかし、図3のアスペクト・プログラムは決して分かりやすいばかりか、and, or, not というタイプのループの合成には成功しているが、ループの形が異なる up, down の合成には成功していない。

3 抽象クラスを利用した解法

もっとうまい方法はないだろうか。上に出てきたメソッドは、いずれも Image というデータを返すという形をしている。一般に、ある型のオブジェクトを返す関数（メソッド）は、通常その型のオブジェクトを作成して、それへのポインターを返すことになる。しかし、外から見たオブジェクトはそのサービス（メソッド）の集合だから、「データ」としてのオブジェクトを返す必要はなく、そのオブジェクトに適用できるメソッドが実質的に与えられればよい。具体的には、getPixel(int,int) というメソッドが定義できればよい。

抽象クラスと継承関係を利用すれば、同じような効果が実現できる。すなわち、以下に示すように Image という名の抽象クラスを作り、not, and, or, up, down その他の演算を、すべてその subclasses として、その getPixel というメソッドを変えるという形で実現する。例として and の場合を示す。なお、記述を簡単にするため（効率への配慮もあるが）、これまでメソッドとして書いてきた width() と height() を public な変数に変えている。

```
abstract class Image {
    public int width, height;
    abstract Pixel getPixel(int i,int j);
}

class AndImage extends Image {
    Image im1, im2;
    AndImage(Image x, Image y) {
        im1 = x; im2 = y;
        width = im1.width;
        height = im1.height;
    }
    public Pixel getPixel(int i, int j) {
        return Pixel.and(im1.getPixel(i,j),
                          im2.getPixel(i,j));
    }
}
```

ただ、このように1つの演算ごとに1つのクラスを定義するのはむだが多いので、Javaの無名クラスを利用して以下のように簡素化する。

```
class ImageOp {
    public static Image not(final Image x) {
        return new Image(x.width, x.height) {
            public Pixel getPixel(int i, int j) {
```

```

        return Pixel.not(x.getPixel(i,j));
    }
}
}
public static Image and(final Image x, final Image y) {
    return new Image(x.width, x.height) {
        public Pixel getPixel(int i, int j) {
            return Pixel.and(x.getPixel(i,j), y.getPixel(i,j));
        }
    }
}
}
//以下 or, up, down の定義は同様.
...
public static Image remove(final Image x, final Image y) {
    return new Image(x.width, x.height) {
        public Pixel getPixel(int i, int j) {
            return and(x, not(y)).getPixel(i,j);
        }
    }
}
}
public static Image topEdge(final Image x) {
    return new Image(x.width, x.height) {
        public Pixel getPixel(int i, int j) {
            return remove(x, down(x)).getPixel(i,j);
        }
    }
}
}
//bottomEdge も同様.
...
//それで最終的に
public static Image horizontalEdge(final Image x) {
    return new Image(x.width, x.height) {
        public Pixel getPixel(int i, int j) {
            return or(topEdge(x), bottomEdge(x)).getPixel(i,j);
        }
    }
}
}
}
}

```

この方法は一見したところ不自然に見えるかもしれないが、画素配列というデータを保存する代わりに、そのデータ要素を取り出すための操作を保存するという常套的な方法を実現したものと考えれば、それほど不自然でもない。もとの論文ではうまく書けずにごまかしてある、up や down もまったく同じように書ける。コンポーネントの記述自身、すっきりしているし、何よりアスペクトのプログラミング部分がまったくいらなくなる。

4 関連する手法

ここで取った方法は、以下のようなすでによく知られている手法と関連がある。

1. すでに述べたように、データを保存する代わりにデータを生成する操作を保存するという手法
2. データが必要になった時に計算するという遅延評価
3. not, and, or などの演算は 1 種のフィルタと見なせるが、それらをパイプによって結合し無駄な中間データを省くフィルタ・パイプ手法

その意味で、過去のよい設計手法の応用といえる。また、設計パターンではコマンド・パターンと見なすこともできる。ただ、GOF によるコマンド・パターンはこのような性能についての考慮をしたものではないようだ。

しかし、問題もある。画像処理の場合は、1 画素に対する操作がメソッド呼び出しに比べて軽いものだから、このようにメソッド呼び出しが連鎖することによるオーバーヘッドは、効率上大きな問題となりうる。これに対し、いくつかの解決策が考えられる。

1. 画像上のごく一部のデータを、少ない頻度で参照する場合は、このプログラムのままでよい。
2. メソッドの呼び出しをインライン展開する。ただし、この例ではメソッド呼び出しが深く入れ子になっているので、コンパイラによる機械的な展開は難しいかもしれない。それでも、原論文にあるようなアドホックな局面プログラムよりはるかに一般的な方法といえる。
3. 多くのアクセスが必要な画像は、上の形でそれを表現したオブジェクトからたとえば画素の配列で実現したオブジェクトに簡単に変換できるので、そのような変換を行う。その変換の際にも、複合化したループが 1 つに統合されるという効果がある。

この 3 番目の方法を具体的に実現するには、たとえば次のようなプログラムを書けばよい。

```
class ConcreteImage extends Image {
    Pixel[] [] pixels;
    ConcreteImage(int w, int h, Pixel[] [] p) {
        width = w; height = h; pixels = p;
    }
    ConcreteImage(Image im) {
        width = im.width; height = im.height;
        pixels = new Pixel[width][height];
        for (int i = 0; i < width; i++)
            for (int j = 0; j < height; j++)
                pixels[i][j] = (im.getPixel(i,j));
    }
}
```

ここで 2 つ目の構成子が、抽象クラスとしての画像をもらって、配列で具体化された画像データを作り出す働きをする。

5 Kiczales グループの反応とその後

この話を 2001 年に Kiczales のグループにしたときの反応が面白かった。グループでも若い学生は、そもそも AOP の原典ともいべきこの論文を知らない。それよりも、これとは別に話したわれわれの役割に基

づく協調モデル Epsilon に興味を持ってくれた。一方、Kiczales を初めとする教員や年長の研究者・学生はこの画像処理例題の話を大変熱心に聴いてくれた。とくに Kiczales は大変口惜しがり、筆者の方法の粗を探そうと努めたのが面白かった。Kiczales によれば、そもそもこの問題は、Xerox PARC の画像処理グループが実際に困っていたのを Kiczales たちが相談に乗ったというもので、現実の要求に基づくものだという。それで当時の資料を引っ張り出してきて、隠れた要求があり、それに応じるには筆者の方法だとまずいところがあるのではないか、というような議論があったが、最終的にはわれわれのやり方のよさを認めざるをえなかった。

それにしても、今の時点で見ると、AOP にとって AspectJ が作られたことはいかに大きなことだったかがつくづく思われる。もし、この時代のようにアスペクトの記述言語と織込みの処理系を問題ごとに一つ一つ作るということが続いていたとしたら、アスペクト指向プログラミングがこれほどもてはやされるようにはならなかったろう。

しかし、この画像処理の例題を現在の AspectJ で扱おうとしてもうまくいかない。AspectJ で処理速度の効率化をはかるような例がまったく扱えないわけではないが、この例で必要なループの融合のようなプログラム変換はできそうにない。AspectJ は一つの新しいモジュール化の方法を提供したが、モジュール化の可能性は多様であり、アスペクトだけが道ではないというメッセージを伝えたいというのが本稿の意図である。

謝辞 同僚の増原英彦氏には、今回東京と大阪で開催したフォーラムでも大変よい話をしてもらったが、本稿の元となった議論を 2001 年に Kiczales のグループする機会を作っていたいただいた。改めて感謝したい。

参考文献

- [1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland. Springer-Verlag, June 1997.
- [2] 玉井哲雄. 局面指向プログラミングの批判的考察. In *ウインターワークショップ・イン・高知 論文集*, pages 65-68, 高知, 1 月 1999. (社) 情報処理学会.

フォーマルアプローチはシステム危機を救うか？

主催：ソフトウェア技術者協会（SEA）

協賛：経営情報学会

経済産業省で作成が進んでいる「情報システムの信頼性向上に関するガイドライン」は以下のような文で始まっています。

現在、我が国の国民生活及び社会経済活動の IT 利用度は、コンピュータの処理性能の飛躍的向上やインターネットの普及等の結果、かつて無いほど高まっている。このため、情報システムの障害による業務・サービスの停止や機能低下の社会的影響は日々、深刻化してきており、システムの信頼性・安全性向上は緊急の課題となっている。

このガイドライン策定のきっかけの一つが、昨年来の東証でのシステムトラブルであることは間違いないでしょう。これ以外にも、大規模なシステム危機の例として 2002 年 4 月のみずほ銀行のシステム統合時のトラブルや、2003 年 3 月の航空管制システムのダウンなどは記憶に新しいところです。

ソフトウェア開発の専門家の間では、大規模なシステムがあまりにも複雑で、トラブルを避けることが非常に困難になりつつあることは以前から常識になっていますが、これを打破するためのブレイクスルーとして、ソフトウェア開発の上流工程やテストングに関して注目が集まっています。今回の SEA フォーラムでは、これらの技術の中から「フォーマルアプローチ」にスポットライトを当てます。ソフトウェアの仕様記述に数理論理的な手法を応用する「フォーマルアプローチ」は、比較的古い歴史を持っていますが、実システムへの応用となると非常にミッションクリティカルな一部の分野で限定的に使用されるに止まっています。ソフトウェア開発の現場では、フォーマルアプローチの基礎となる数学や論理学といったものに対するアレルギーが強いのも事実です。

今回のフォーラムでは、フォーマルアプローチおよび要求工学の専門家をお招きし、最新の動向をお話して頂くとともに、その有効性と限界について活発な議論をおこなう引き金にしたいと考えています。

***** 開催要領 *****

1. 日時：5月31日(水) 14:00～18:00

2. プログラム

13:30 - 14:00 受付

1400 - 15:30 発表(3件、それぞれ30分)

「フォーマルメソッド：開発現場での使用経験」 佐原伸 (CSK システムズ)

「フォーマルアプローチの限界はどこにあるか？」 飯島淳一 (東京工業大学, 経営情報学会会長)

「システムリスクとフォーマルアプローチ」 山本修一郎 (NTT データ 技術開発本部)

15:30 - 16:00 Coffee Break

16:00 - 18:00 討論：フォーマルアプローチの効果と限界

コーディネータ：田中一夫 (CSK システムズ, SEA 代表幹事)

パネリスト：上記講師3人＋野中哲 (トゥルーロジック, SEA 幹事)

3. 会場：全国情報サービス産業厚生年金基金(JJK)会館 7F B会議室

4. 定員：50名

SEA フォーラムスペシャルを終えて

野中 哲*

2006年6月19日

5月31日の午後JJK会館で「フォーマルアプローチはシステム危機を救うか」と題して、SEAフォーラムが開催された。5月は、ほんの2週間前にも同じ会場で「アспект指向」を題材にフォーラムが開催されたばかりであり、いったどれだけの人が参加してくれるのか非常に不安であったが、募集を始めると思いの外集まりが好調で、最終的には50名以上もの人たちが申し込んでくれた。当日参加してくださった皆様、本当にありがとうございました。

当日のレポートは別途なされると思うので、ここでは個人的なフォーマルアプローチに対する思いを、少々書いてみたいと思う。2005年の4月に、自分で会社を立ち上げようと思った時、普通のソフトウェア会社となにか違う特色を出さねばと思い、思案の結果、自分が一番強く興味を持っている、数理論理的手法のソフトウェア開発への応用を活動の中心にしようと決めた。それ以降、細々とではあるが、フォーマルアプローチの普及のための活動を行ってきた。そうしてやっと一年が経過したことになる。その感想はといえば、一番強く感じるのは、数理論理的手法に対する世間のアレルギーだ。「いい技術だと思うが使い物にならない」という意見を非常に多く耳にする。一方、この風向きもかわりつつあるような兆候も見られる。たとえば、このフォーラムにこれだけ多くの人が集まったということ。これはソフトウェア開発に対する危機感と数理論理的手法への期待感なのではないかと思う。

私が今回のフォーラムについて持っていた興味は「なぜ、あなたはフォーマルアプローチは使えな

いと思うのか？」というものだった。当日のディスカッションで会場から、これに関連して、いくつか参考になる意見を聞くことができた。曰く、

- 作成するドキュメントが増え、手間が増えるのではないか？
- 要員を別途訓練しないとイケないが、その費用が捻出できない
- その一方、人の入れ替わりが激しい
- マネジメントに理解が得られない
- 初期開発では使われたが、その後担当者がプロジェクトを離れてしまったため、保守されなくなった

フォーマルアプローチの推進を目指す者としては、これらの疑問の一つ一つに丁寧に答えを提示していく必要が有るだろう。「フォーマルアプローチはあまりにも特殊な技術だと考えられ過ぎている」という思いが私にはある。ただ一口にフォーマルアプローチといっても、その内容にはいろいろなレベルが存在している。高度な数学的素養や訓練を必要とする手法があるのも確かだが、一方普通にCやJavaでプロミシングができるならば、簡単な教育で実践可能でかつ効果的な手法もある。フォーマルアプローチは、ソフトウェア開発プロジェクトの全ての問題を解決してくれるような万能薬ではないが「曖昧で不明確な仕様」に起因するようなトラブルに対して効果があるのは事実である。フォーマルアプローチへの入り口が少しでも広がり、より多くのソフトウェア開発現場で、この手法が使われ、少しでも高信頼性、高品質なソフトウェアが世の中に広まるよう、今後も活動してゆきたいと考えている。

* (有) トゥルーロジック

形式手法：開発現場での使用経験

証券会社向けシステムでの形式手法適用の成功事例：
後続く人のために

株式会社CSKシステムズ
通信グループ
佐原伸

1

形式手法VDMで何ができたか？

- ◆ 証券会社向けパッケージソフトウェア
 - ◆ 全体の約5%をVDMを使用して開発
- ◆ 効果
 - ◆ 欠陥発生数 16分の1
 - ◆ 生産性 2.5倍
 - ◆ 開発期間 45%

2

形式手法の構成要素

- モデル（レベル1）
 - 仕様記述言語
 - VDM, Z, B
- 段階的洗練（レベル2）
 - ◆ 約3000の変換公式
- ◆ 検証（レベル3）
 - ◆ 証明

我々の事例では
ここまで使った

3

VDM

- ◆ ウィーン開発手法
 - ◆ IBMウィーン研究所で最初に開発
 - ◆ VDM-SLはISO標準（1996）
 - ◆ VDM++は欧州連合ESPRIT計画で開発
- ◆ CSKがVDMTools買収(2005)
 - ◆ 無料で配布中⇒open or free

4

実在の日本語仕様

信用取引の決済日（期日）を得る。

弁済期限とは、信用建玉に対して当社がお客様に信用を供与する期限をいいます。弁済期限は、現在のところ6ヶ月のみを取扱っています。

弁済期限が6ヶ月であるということは、信用建玉の建日（信用建玉が約定した日）の6ヶ月目応答日が信用期日となり、この日を超えて建玉を保有することは法律で禁じられています。信用期日が休日の場合には、直近の前営業日が信用期日となります。

5

日本語仕様の曖昧さ

- ◆ 弁済期限と決済日と信用期日との関係は？
- ◆ 応当日とは？
- ◆ 応当日が月末日を超えたらどうなるのか？
- ◆ 直近の前営業日が5ヶ月後になった場合はどうするのか？

6

VDM++仕様(事後条件)

```
static public 信用期日を得る : Date -> Date
信用期日を得る(aDate) == is not yet specified

post
  let
    mk_(年, 月) = 暦.6ヶ月後を求める(aDate.年(), aDate.月()),
    日 = aDate.日(),
    期日候補 = 期日候補を求める(年, 月, 日)
  in
    RESULT = (期日候補.過去の平日を得る()) and
    if 月初から期日候補日まで休み(期日候補) then
      RESULT.月() = 前月を得る(年, 月)
    else
      RESULT.月() = 月;
```

7

VDM++仕様(補助関数1)

```
static public 期日候補を求める : int * int * int -> Date
期日候補を求める(年, 月, 日) ==
  let 月末日 = 暦.月末日を求める(年, 月)
  in
    if 月末日.日() < 日 then
      月末日
    else
      暦.整数三つ組から日付を得る(年, 月, 日);
```

VDM++仕様(補助関数2)

```
static public 6ヶ月後を求める : int * int -> int * int
```

```
6ヶ月後を求める(年, 月) == 暦.正則月を得る(年, 月+6);
```

```
static public 月初から期日候補日まで休み : Date -> bool
```

```
月初から期日候補日まで休み(期日候補) ==
```

```
forall day in set 1, ..., 期日候補.日()
```

```
暦.お休みか?(暦.整数三つ組から日付を得る(期日候補.年(), 期日候補.月(), day));
```

```
static public 前月を得る : int * int -> int
```

```
前月を得る(年, 月) ==
```

```
let mk_(-, 前月) = 暦.正則月を得る(年, 月-1) in
```

```
前月;
```

9

VDM++仕様(本体)

```
static public 信用期日を得る : Date -> Date
```

```
信用期日を得る(aDate) ==
```

```
let
```

```
mk_(年, 月) = 暦.6ヶ月後を求める(aDate.年(), aDate.月()),
```

```
日 = aDate.日(),
```

```
期日候補 = 期日候補を求める(年, 月, 日)
```

```
in
```

```
期日候補.過去の平日を得る() --前月の日付になることがある。
```

```
post ...
```

10

VDM++仕様(テストケース)

```
test() ==
  let c = new SBCalendar(),
      sDate = SBCalendar.信用期日を得る in
  return
    sDate(c.整数三つ組から日付を得る(2004, 7, 30)). =
      (c.整数三つ組から日付を得る(2005, 1, 28)) and
    sDate(c.整数三つ組から日付を得る(2003, 8, 2)). =
      (c.整数三つ組から日付を得る(2004, 2, 2)) and
    sDate(c.整数三つ組から日付を得る(2005, 8, 29)). =
      (c.整数三つ組から日付を得る(2004, 2, 27)) and
    sDate(c.整数三つ組から日付を得る(2003, 11, 1)). =
      (c.整数三つ組から日付を得る(2004, 4, 30))
```

11

導入事例

- ◆ 欧米
 - ◆ アムステルダム花市場 (開発工数1人年)
 - ◆ ロシア人工衛星 (VDM仕様100万行以上)
 - ◆ 米国戦闘爆撃機 (ボーイング)
- ◆ 日本
 - ◆ 列車信号制御 (JR鉄道総研)
 - ◆ 証券システム (CSKシステムズ)
 - ◆ ICカードシステム (フェリカネットワークス)

12

今後の課題と対策

- ◆ 課題

- ◆ ドメイン・モデルの作成
 - ◆ 仕様フレームワーク+仕様ライブラリ
- ◆ 実時間処理のモデル化機能 (VDM VICE拡張)
- ◆ モデルチェックング・ツールとの関係
- ◆ シミュレーションツールとの関係

- ◆ 対策

- ◆ VDM研究会→VDMコンソーシアム
- ◆ アカデミックとの関係
 - ◆ 九州大学、国立情報学研究所、北陸先端科学技術大学院大学、産業技術総合研究所

5

まとめ

- ◆ VDMは使えるか？⇒ Yes。しかし魔法ではない！
 - 品質・生産性とも高くなる
 - 大規模システムでも使える
 - ◆ リリース後のトラブルがかなり少ない
- ◆ VDMを教育できるか？⇒ Yes & UMLより簡単
 - ◆ 日本語資料を使ったセミナーでは落ちこぼれ無し (2003年6月以降)

14

フォーマルアプローチの限界はどこにあるか？

東京工業大学・大学院社会理工学研究科
経営工学専攻
飯島淳一
(経営情報学会)

IS-FMG, T.I.T. All Right Reserved, 2006



経営情報学会のご紹介

- ☐ 1992年に、文系と理系の経営情報系学会が合併して設立
- ☐ ミッションは、「**情報および情報技術の活用に関する対話の場の提供**」
- ☐ 会員数 約1700名(2006年4月現在)

- ☐ 主な活動
 - 春と秋の研究発表大会(毎回120件程度の発表)、6月3日、4日にOA学会と合同大会(中央大)
 - 年に数回のシンポジウム:4月21日にクスマノ氏特別講演会を実施(情報システム工学会)
 - 10あまりの研究部会と2つの特設研究部会(『情報投資と経営成果』、『医療経営と情報技術』)
 - 積極的な提言活動:『東証問題』(システム統合特設研究部会)など

- ☐ 新たな試みー**ICT利活用モデル学会**としてー
 - 会員メルマガによる迅速な情報共有
 - 2007年度より以下を実施予定
 - ・ 研究者マップ
 - ・ 論文誌のPDF配信
 - ・ 正会員の種別化(デジタル会員の設定)
 - 賛助会員(一口5万円)のメリット
 - ・ 正会員3名分の学会誌を配布
 - ・ 広告費1年間無料
 - ・ 学会サイトにお名前をリスト、そこからリンク
 - ・ 研究者マップへのアクセス(予定)

内容

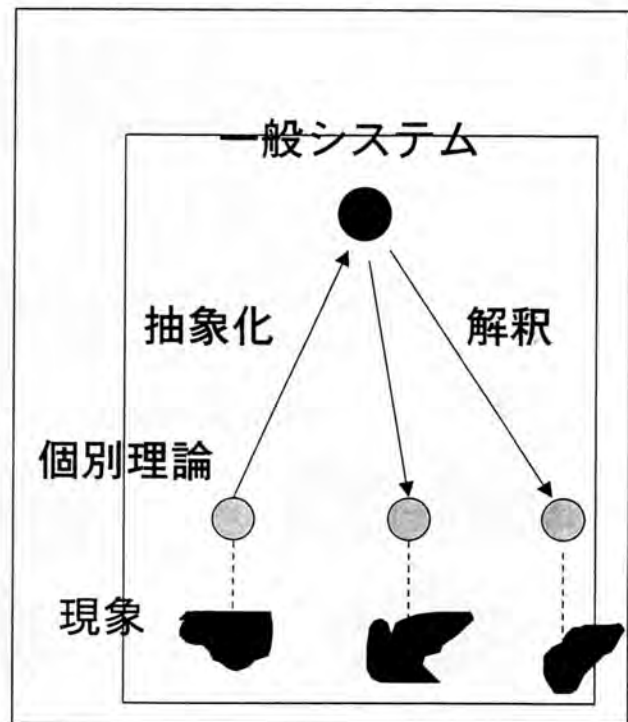
- 数理的システム理論の目指したもの
 - 情報システム学には多元主義的アプローチが必要
 - フォーマルアプローチの強みと弱み
-

数理的システム理論 の目指したもの

一般システム理論の目的

：一般システム研究協会趣意書(1954)より

- 異なる分野における概念や法則間の同形性を追求する.
- モデルが欠けている分野へ援用したり, 研究の重複をなくす.
- 共通の言語を提供することで, 科学の統一をはかる.



システム, 時間システム

- **システム**とは, 対象間の数学的關係

$$S \subset V_1 \times \dots \times V_n$$

である。

- **時間システム**とは, 入出力が共に時間関数となっている特別な入出力システム

$$S \subset X \times Y$$

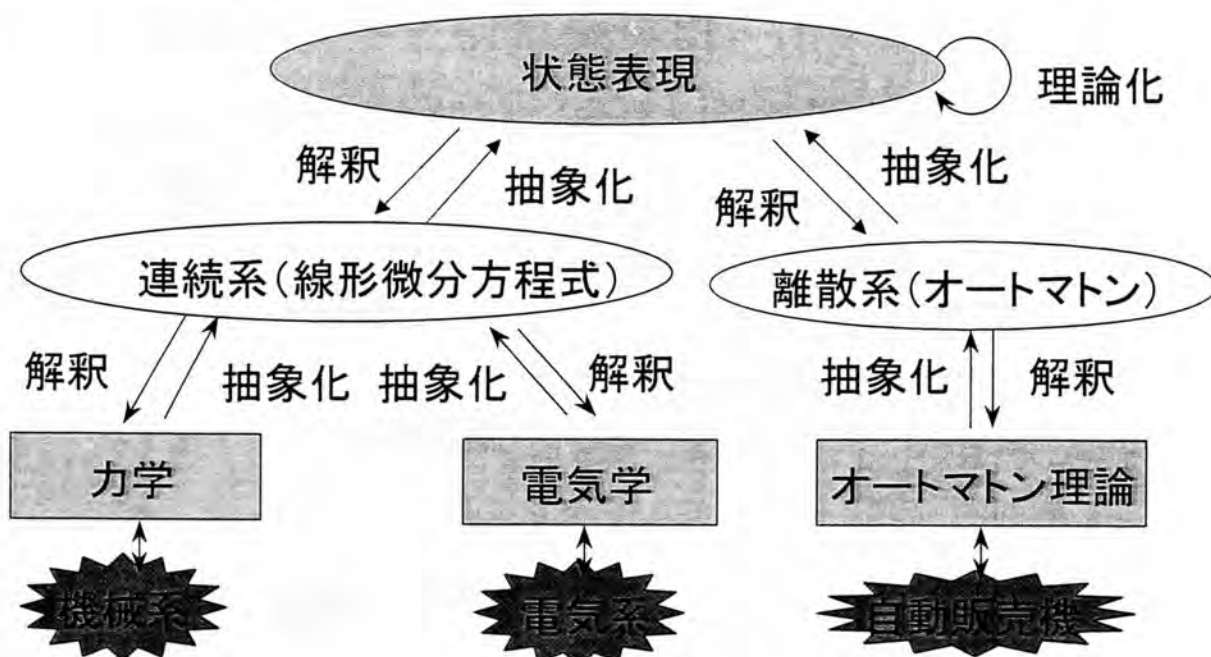
である。

ここで,

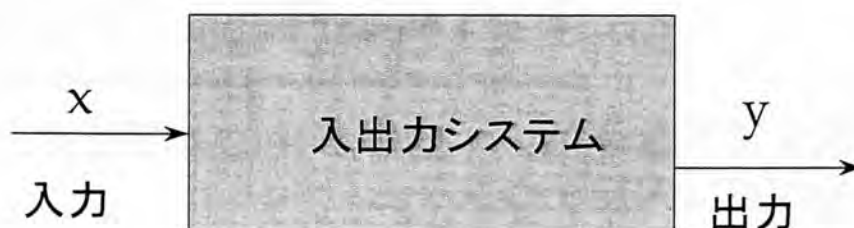
$$X \subset A^T = \{x | x: T \rightarrow A\}, Y \subset B^T = \{y | y: T \rightarrow B\}$$

である。ただし, A, B は集合, T は**時間軸**。

システム理論における状態表現の位置づけ



状態空間表現の対象となる入出力システム



- 外部表現は $S \subset X \times Y$ で与えられる. 入出力は, 共に時間関数, すなわち, $X \subset A^T$, $Y \subset B^T$ である。
- 機械系, 電気系, 共に**連続**時間システムであり, 自動販売機は, **離散**時間システムである。

状態空間表現のシステム論的表現

□ 状態遷移関数

$$\Phi: C \times \cup \{X^t | t \in T\} \rightarrow C$$

(連続系)

$$\Phi(c, x^t) = e^{Ft} c + \int_0^t e^{F(t-\tau)} G x^t(\tau) d\tau$$

(離散系)

$$\Phi(c, x^t) = \delta(\dots(\delta(c, x^t(0)), \dots, x^t(t-1)))$$

□ 出力関数

$$\mu: C \times A \rightarrow B$$

(連続系)

$$\mu(c, a) = Hc$$

(離散系)

$$\mu(c, a) = \lambda(c, a)$$

どのような入出力時間システムが 状態空間表現を持つか？

入出力連続時間システムSが、

- 1) 線形
- 2) 零入力に対する出力空間S(0)が有限次元
- 3) 因果的
- 4) 定常的

であるならば、(先の型の) **状態空間表現**を持つ。

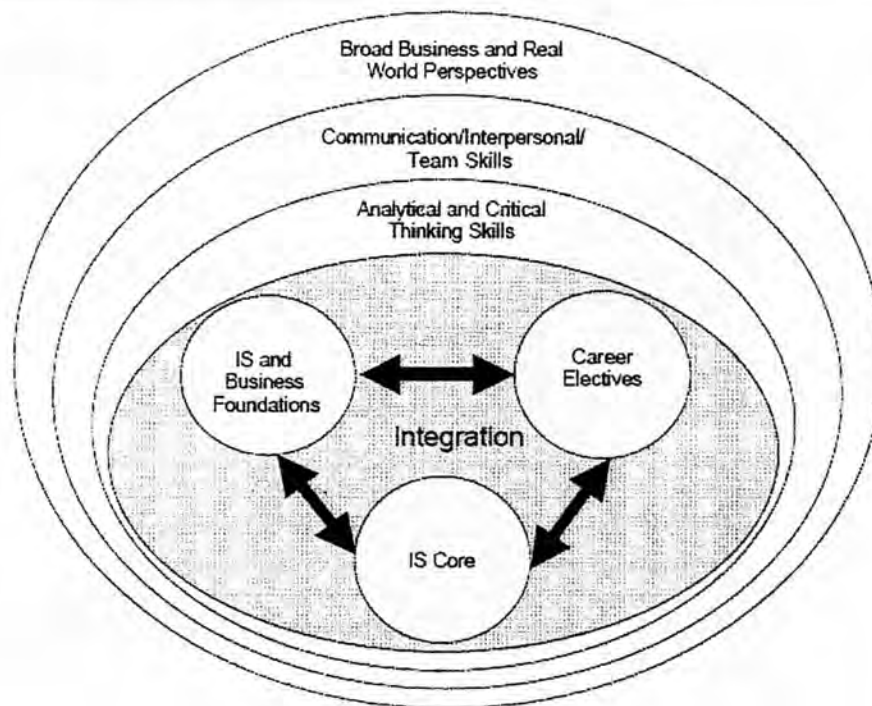
なぜ、数理的・一般システム理論が普及しなかったのか？

- 数理論理的表現が難しげ
 - 集合論にもとづく表現は、一般にはまだ普及していない。
- 抽象的思考の困難さ
 - second orderなので、個別理論に比べて、実体から距離があり、記号処理が中心となる
- 具体性のある新たな発見がない
 - 証明された命題や定理は、second orderレベルのもので、現象の世界まで落とすことが難しい。

結局、工学の世界ではあまりありがたみが感じられなかった。⇒ 情報システムやビジネスに対象をシフト

情報システム学には 多元的アプローチが必要

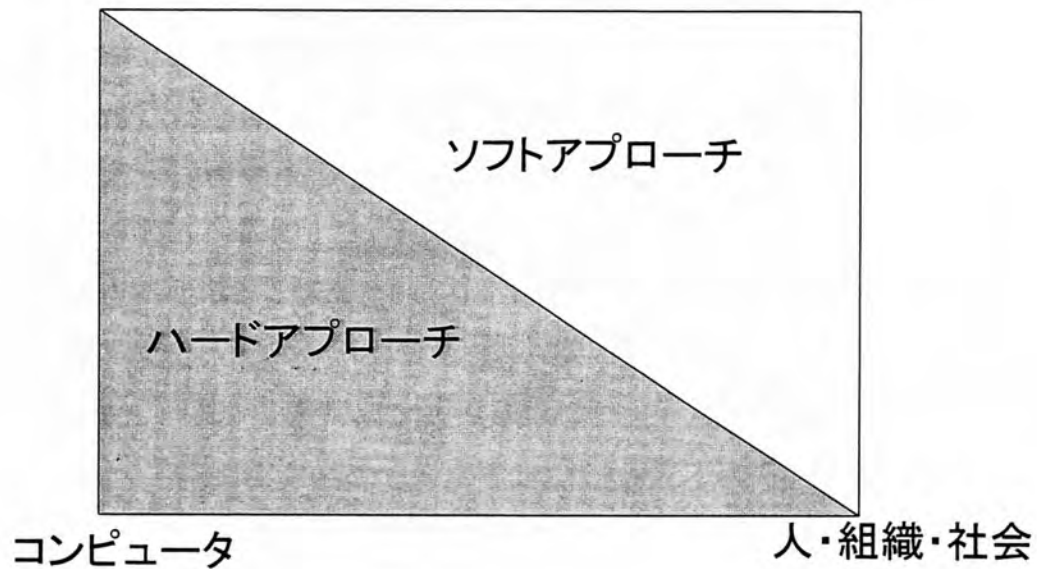
Skills, Knowledge, and Values of MSIS Graduates



断片化されたアドホクラシー

- 研究は個人的なものが多く、分野全体としての調整はゆるやかである。
- 研究者の評価は、そのグループ特有のやり方にしながった貢献で判断される。
- 分野はオープンで、ある程度の教育を受けたものであればアマチュアでもこの分野の標準に影響を与え得る。
- この分野への参入障壁は低く、一つの断片から他の断片への移行はきわめて容易である。
- コミュニケーションでは、常識的な用語が用いられる。

情報システム学における多元主義の必然性



フォーマルアプローチの 強みと弱み

フォーマルアプローチで できることとできないこと

□ できること

- 予測可能な例外発生に対する対応とシステムの頑健性に対する保証

□ できないこと

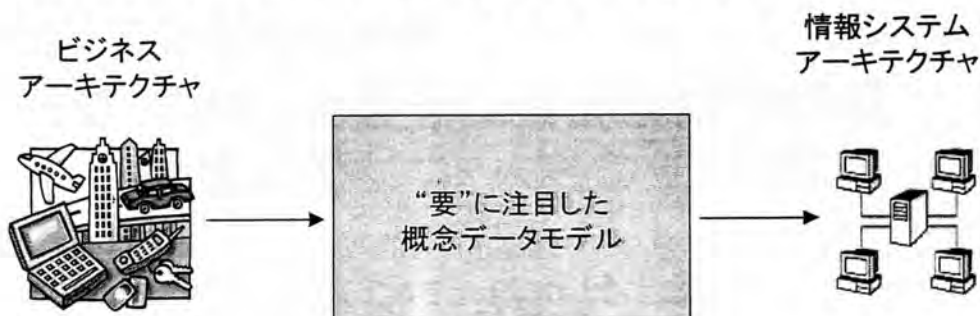
- 失敗の種類は予測不可能
- 範囲を定めて対応

抽象化と一般化

□ フォーマルアプローチのトレーニングは、抽象化と一般化のトレーニングとして有効

- 抽象化: ESシェル, DSSジェネレータ
- 一般化: モジュール, コンポーネント

□ ビジネスの“要”を捉える際に有効



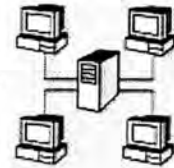
ビジネスアーキテクチャと整合した 情報システムアーキテクチャの設計

ビジネス
アーキテクチャ



“要”に注目した
概念データモデル

情報システム
アーキテクチャ



結論

- フォーマルアプローチは、誰もが使いこなせるものではない。
- 情報システムについての議論では、ITだけを取り出して議論することはできない。
- フォーマルアプローチの強みと弱みを意識すべき。
- 一般化や抽象化能力を養うために、フォーマルアプローチのトレーニングは重要

だけど、わたしはフォーマルアプローチが好き！

システムリスクとフォーマルアプローチ

平成18年 5月31日

株式会社NTTデータ
技術開発本部 副本部長

工学博士 山本修一郎

Copyright 2006 NTT DATA CORPORATION

本日のアジェンダ

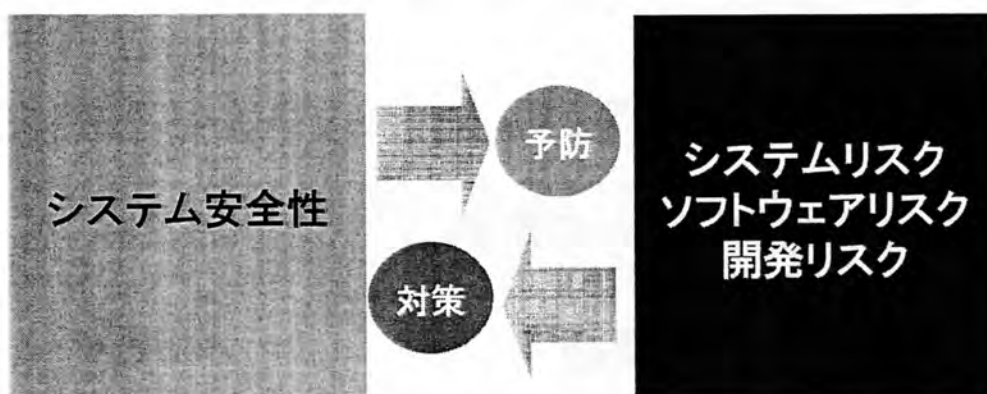
- システムのリスク
- 現実社会と形式モデル
- 形式的な取り組みの事例
- まとめと今後の課題

1 システムのリスク

Copyright 2006 NTT DATA CORPORATION

システムリスク

「システムリスク」という社会的・技術的な概念を
どのように合理的に把握、管理すべきかというところに工学的な焦点をあてる



- 望ましくないことが決して起こらない
- 望ましいことがいつかは必ず起こる

- 望ましいことが決して起きない
- 望ましくないことがいつかは必ず起こる

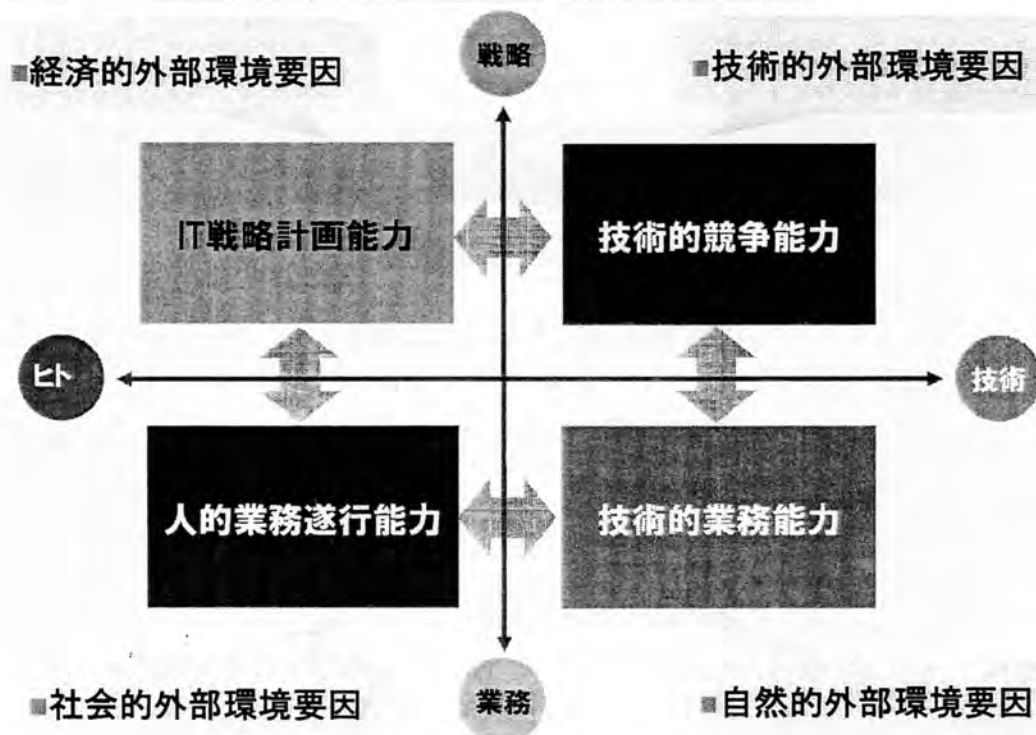
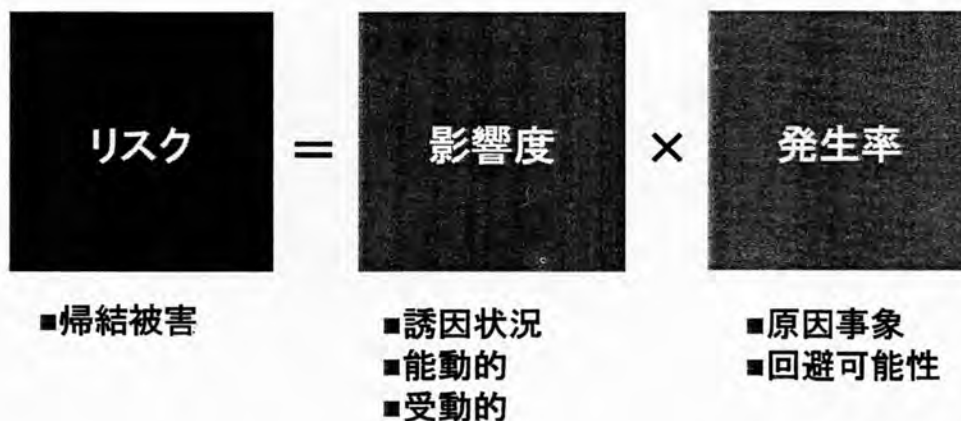
帰結
被害

誘因
状況

原因
事象

「ITリスクとは、コンピュータと通信を中心とした技術手段と、当該技術手段の適用局面において、特定の事象がそのときの状況と結びついて生ずる帰結の不確かさの程度を定性的または定量的に表したものをいう」

堀江正之、IT保証の概念フレームワーク — ITリスクからのアプローチ、森山書店、2006



(参考)堀江正之、IT保証の概念フレームワーク、森山書店、2006



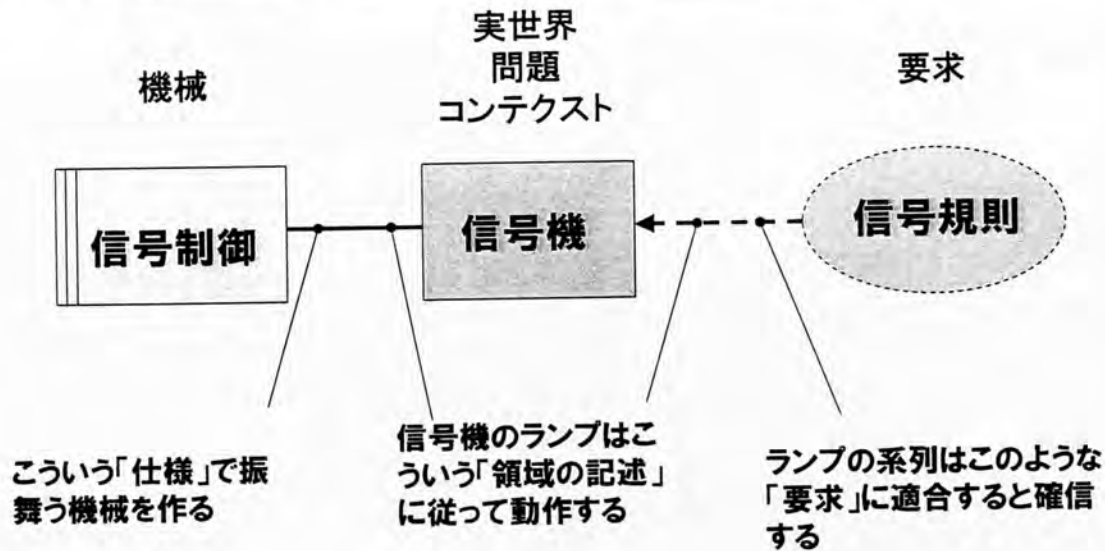
対象	損失の例	システム例	分類	非機能要求の例
ビジネス	■経済的損害	■座席予約	信頼性	■可用性 ■故障率
ミッション	■使命の不達成	■衛星制御	性能	■応答性 ■スループット ■時間間隔
安全性	■生命・環境の破壊	■放射線治療	セキュリティ	■アクセス権 ■バックアップ ■暗号化
			操作性	■年齢 ■学習時間 ■エラー率 ■好感度
			安全性	■システム動作に対する制限 ■システム状態に対する制限

(参考) Kotonya, Sommerville, Requirements Engineering, 1998



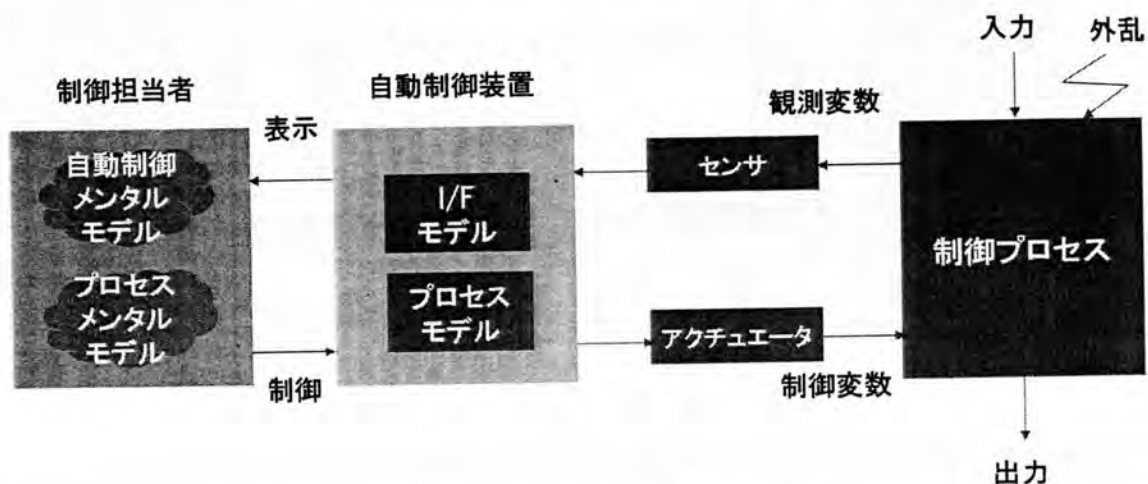
共通点	説明
期間	■希望に基づく不可能な見積り
仕様化	■何を作るかについて利害関係者との合意に失敗
範囲	■追加要求による拡大
要員	■プロジェクトからの要員の離脱
生産性	■予測と実態の乖離

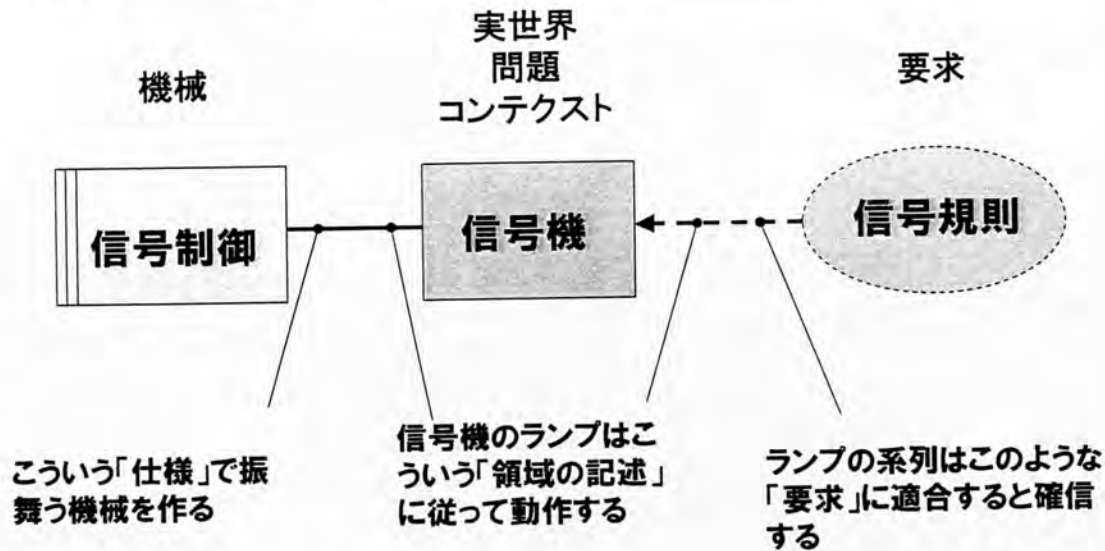
(参考) Tom DeMarco and Tim Lister, Risk Management during Requirements, IEEE Software, 2003



Traditional accident models were devised to explain losses caused by failures of physical devices in relatively simple systems. They are less useful for explaining accidents in software-intensive systems and for non-technical aspects of safety such as organizational culture and human decision-making.

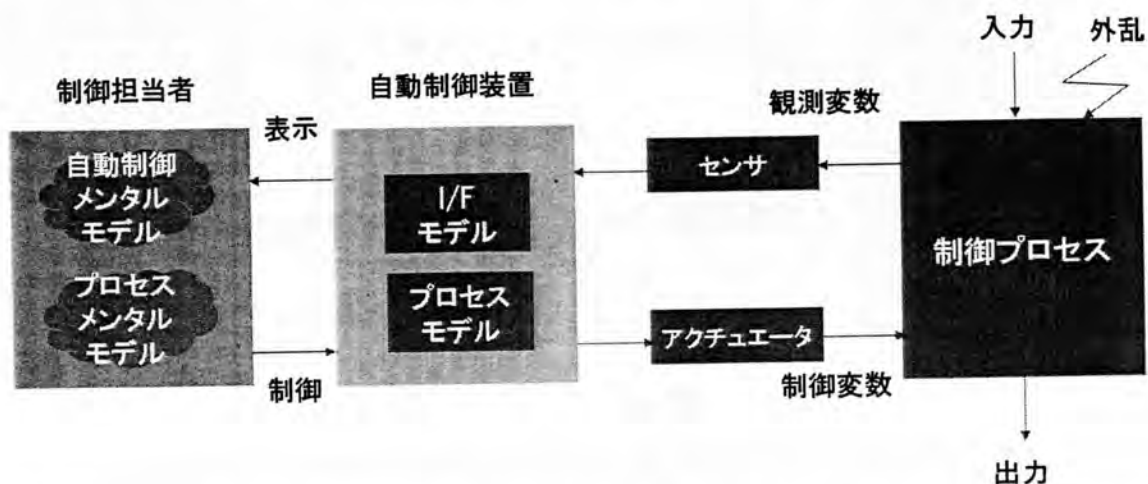
Nancy G. Leveson, A Systems-Theoretic Approach to Safety in Software-Intensive Systems





Traditional accident models were devised to explain losses caused by failures of physical devices in relatively simple systems. They are less useful for explaining accidents in software-intensive systems and for non-technical aspects of safety such as organizational culture and human decision-making.

Nancy G. Leveson, A Systems-Theoretic Approach to Safety in Software-Intensive Systems



3 取り組みの事例

■ ポリシー制御
■ 電子社会

Copyright 2006 NTT DATA CORPORATION

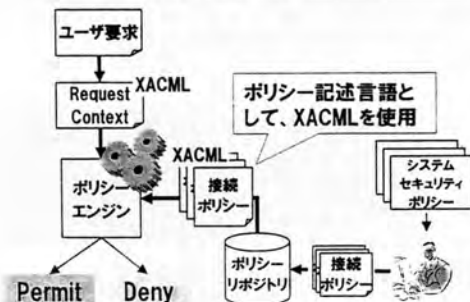
ポリシーを用いた接続制御技術

セキュリティ確保の観点から、利用者からの接続要求に対して、VPN接続してよいかどうかの判断が必要

利用者からの多様なVPN 接続要求に対して、システム・セキュリティポリシーを考慮した接続可否判断の実現するためには、パケットフィルタのような単純な条件設定ではない接続制御機能が必要

ポリシー制御技術を適用

- ポリシーは、If *condition* then *action*形式で表されるルールの集合
- 要求やイベントの内容が、*condition*に一致したとき、*action*が発火することでシステムを制御
- 実装には、XACMLを用いた

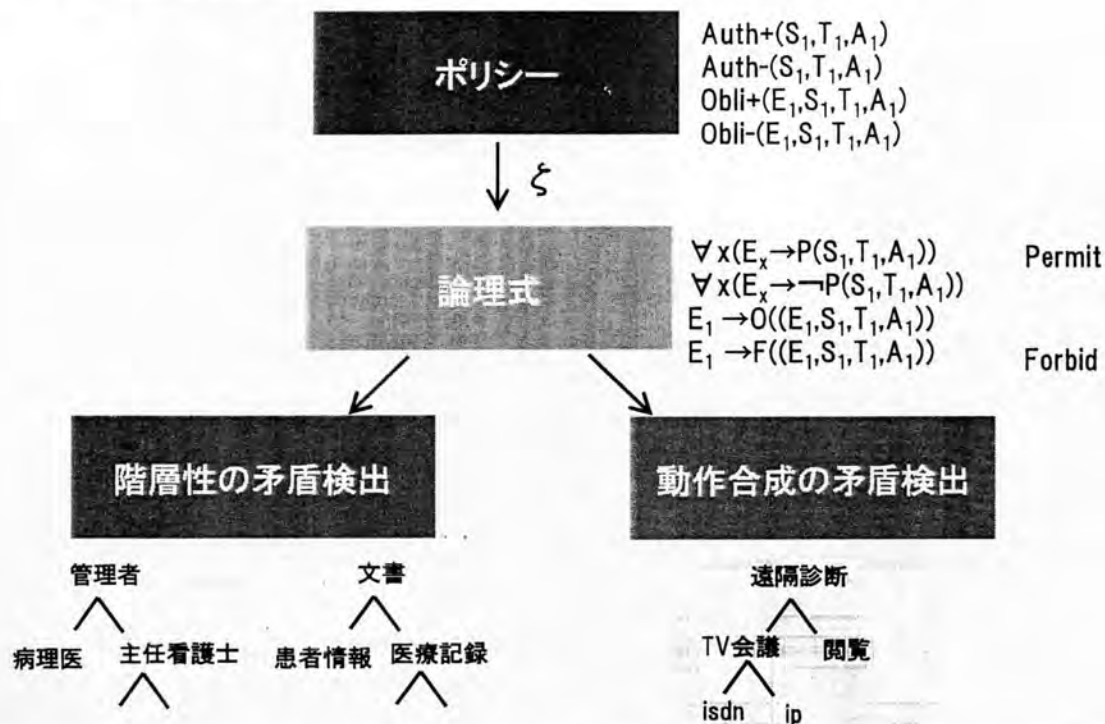


処理フロー

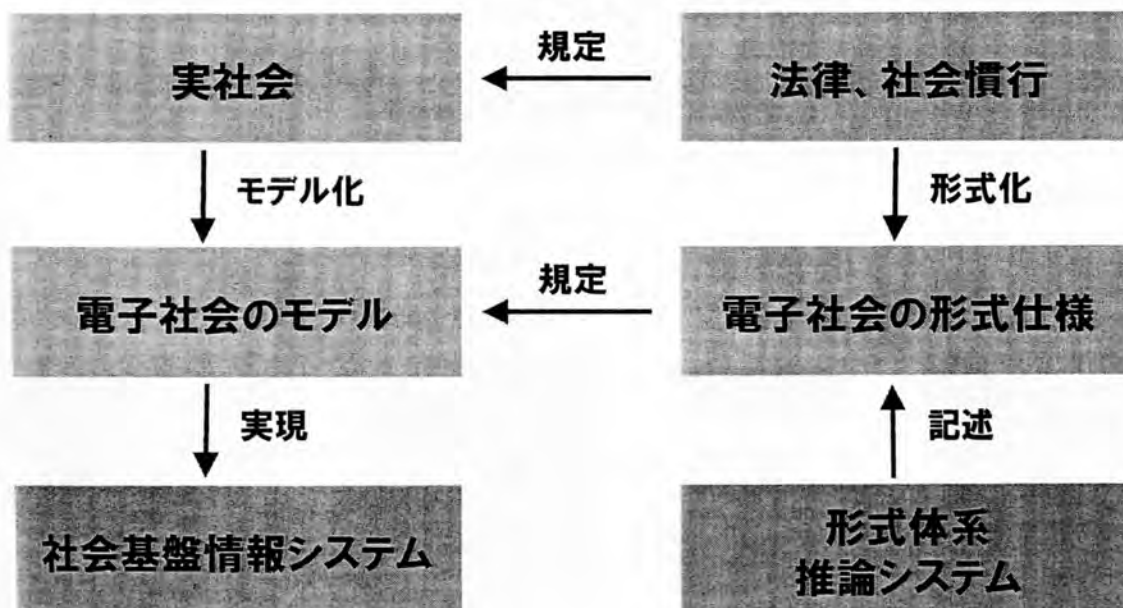
```
<Policy>
...
<Rule>
  <Target>
    <Subjects>
      <Subject>...</Subject>
      <Subject>...</Subject>
      <Subject>...</Subject>
    </Subjects>
    <Resources>
      <Res>
        <Subject>
          <Attribute
            Resource: AttributeID="Identifire:Subject:Role"
          </Res>
        </Subject>
      </Res>
    </Resources>
    <Action>
      <Act>
        <AttributeValue>administrator</AttributeValue>
      </Act>
    </Action>
  </Target>
</Rule>
...
</Policy>
```

XACMLによるアクセス制御ポリシー

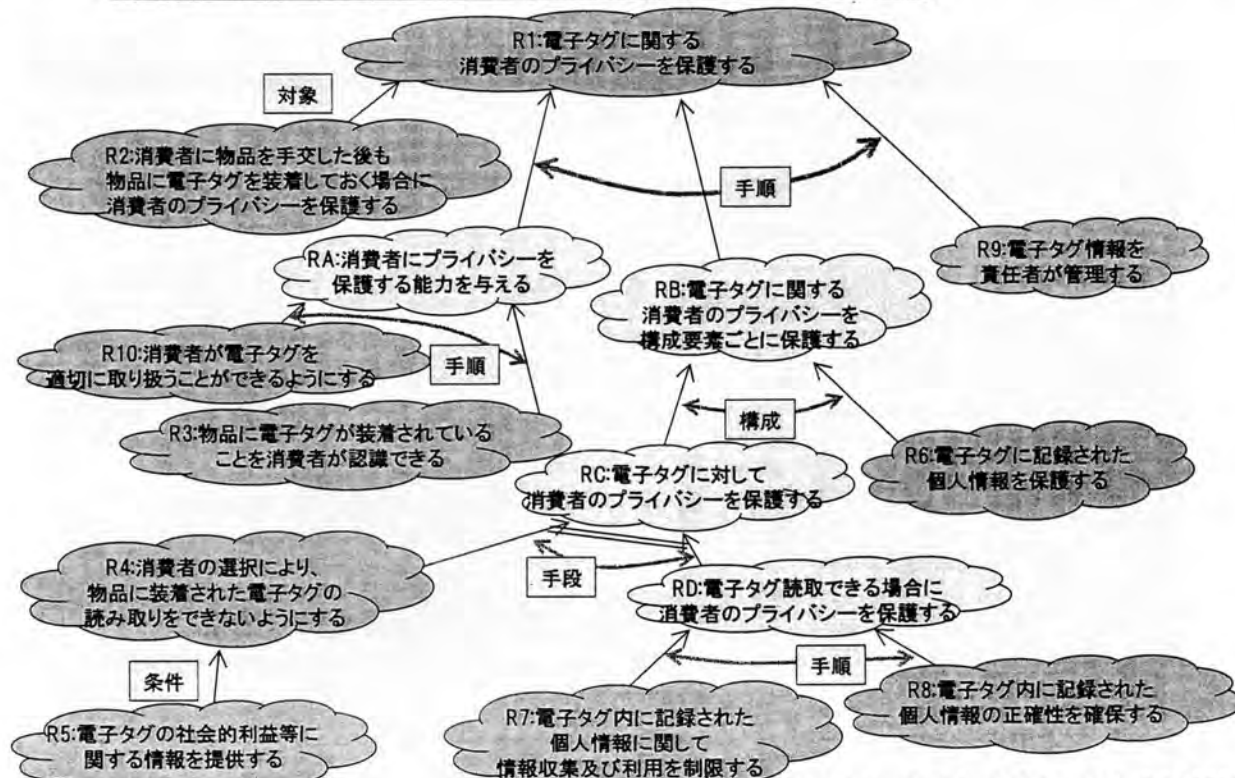
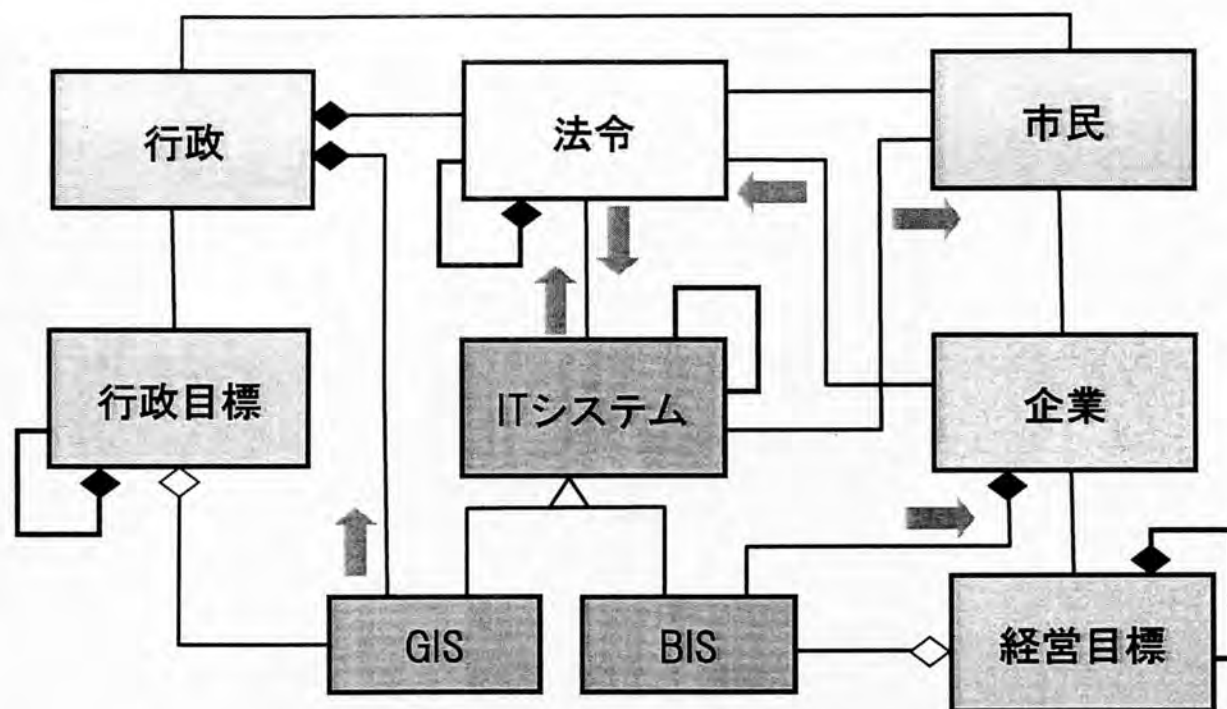
XACML: OASISにより標準化作業中のポリシー記述言語
(eXtensible Access Control Markup Language)



(参考) Kamoda et al., Policy Conflict Analysis Using Tableaux for On Demand VPN Framework, TSPUC 2005



(参考) 片山卓也、検証進化可能電子社会、情報処理学会誌、vol.46, no.5, pp.515-512, 2005.



(参考)山本他、信学会、KBSE研究会、5.15.2006

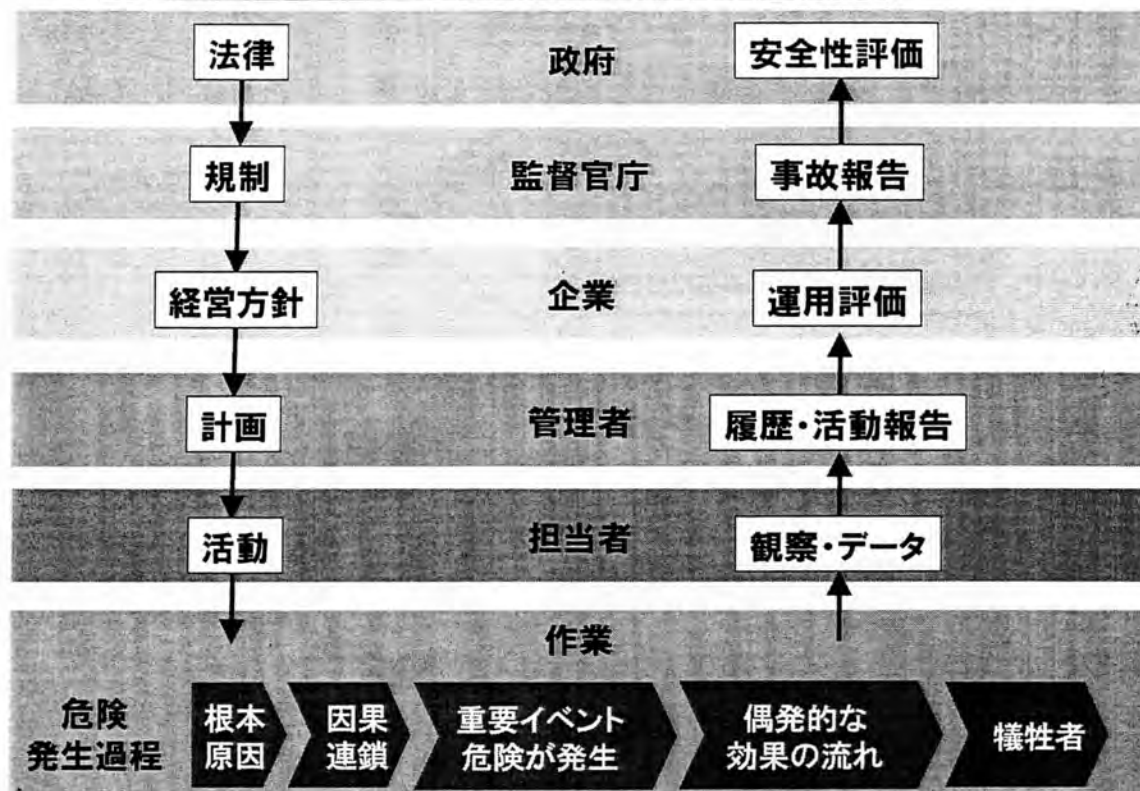
4 まとめと今後の課題

Copyright 2006 NTT DATA CORPORATION

一連の『東証』の問題に対する経営情報学会 システム統合研究部会の見解

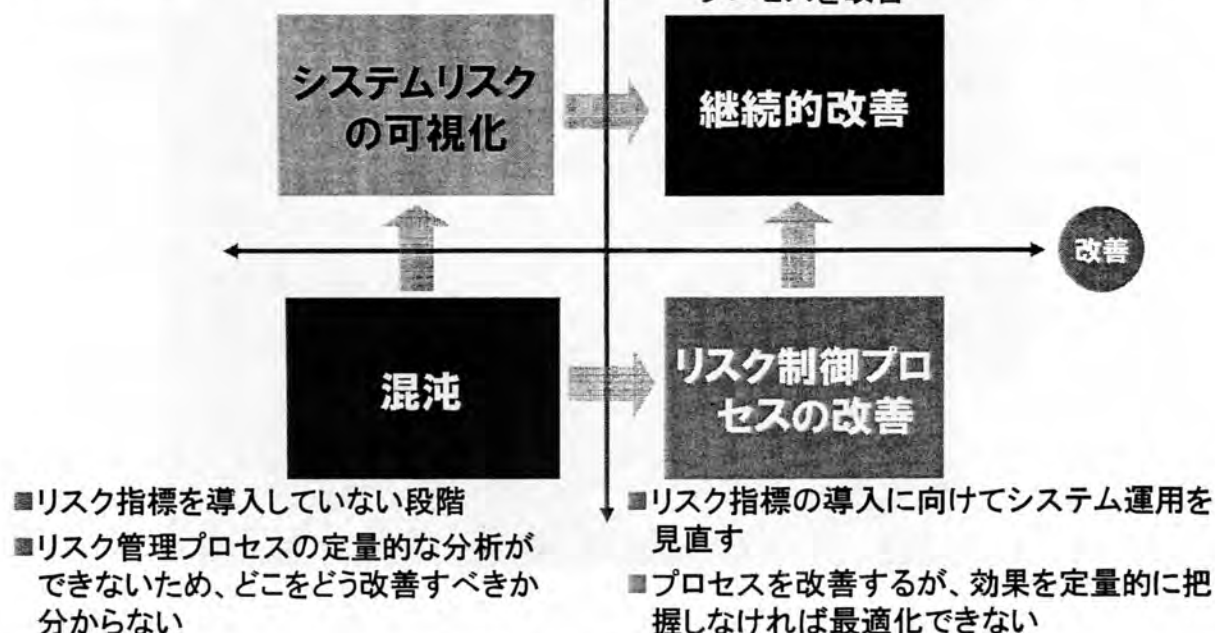
論点	対策
証券ビジネス	■証券取引所ビジネスが情報システムで成り立っていることの理解
アウトソースIT ガバナンス	■発注側の機能
キャパシティ問題	■eビジネスでは、物理的な制限がない、または見えない ■処理能力の増強には限界があり、技術的に対応が困難、あるいは法外な費用がかかる ■予防対策としての小口取引の手数料体系のあり方や、非常時の流入制限などの制度的スキームについても、きちんと議論したうえで、事前に関係者の合意形成をはかり、事態の收拾を円滑に進めることができる体制を整える必要がある。
システム障害は 避けられない	■システム障害は自然災害と同じように、発生することを前提に対処法を考えて、その原因を短時間で究明し、改修できるように、トラブルや変更に強い柔軟な構造を持つシステムを作る ■処理能力を超えそうになったり、取り消しが出来ないというような、異常状態に対して、市場閉鎖のような措置をすばやく発動するような、障害発生を前提としたルール作りや体制の整備 ■システム障害が発生した場合には、できるだけ情報を開示して、広く建設的な議論を巻き起こし、できるだけ、再発防止のためのノウハウが社会的に蓄積するしくみ ■航空機事故における調査機関のような、第三者機関による公正な分析や判断ができるようなしくみ

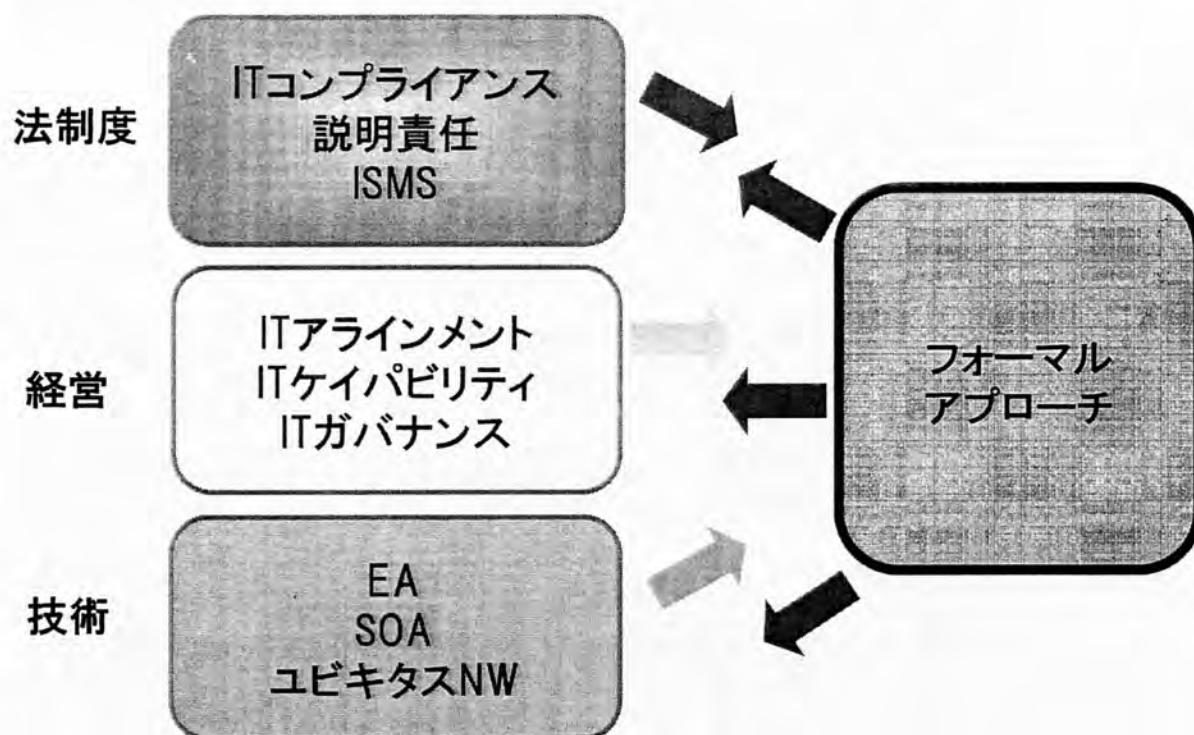
(参考)経営情報学会, <http://www.jasmin.jp/jp/pr/social/event/20060206/index.htm>



- リスク指標の導入によりリスク監視プロセスの状況を管理
- プロセス改善に向けた課題を抽出

- リスク指標によりシステムリスクをリアルタイムに監視
- 最適化に向け、継続的にリスク制御プロセスを改善





SEA Forum 「フォーマルアプローチはシステム危機を救うか？」 レポート

2006/06/18

東京工業大学大学院

社会理工学研究科経営工学専攻

修士2年 中村 英二

場所：全国情報サービス産業厚生年金基金（JJK）会館

日時：5月31日 13:30-18:00

プログラム内容：

13:30-14:00 受付

14:00-15:30 3名の発表

「形式手法：開発現場での使用経験」佐原伸さん（CSK システムズ）

「フォーマルアプローチの限界はどこにあるか？」飯島淳一教授（東京工業大学、経営情報学会会長）

「システムリスクとフォーマルアプローチ」山本修一郎さん（NTT データ 技術開発本部）

15:30-16:00 Coffee Break

16:00-18:00 討論：フォーマルアプローチの効果と限界について（コーディネータ：田中一夫さん、パネリスト：佐原さん・飯島教授・山本さん・野中さん）

1. 発表者3名の方々の発表内容

フォーラムの始めに株式会社 CSK システムズ佐原さんによる、証券会社向けシステム開発での VDM の使用結果に関するプレゼンテーションが行われた。

証券会社向けシステム開発に形式手法 VDM を適用した結果、欠陥発生数がほとんどなく、生産性が2.5倍、開発期間が45%に短縮できたという成功事例が紹介された。これらの便益が享受できる原因の理由として、VDMをはじめとする形式仕様言語は使用の曖昧さを排除できるという強みがある。例えば日本語で証券取引の仕様を記述した場合、曖昧な記述が混在するケースが多々ある。以下は、プレゼンテーションの中で紹介された日本語での証券取引仕様の一部である。

仕様取引の決済日（期日）を得る。

弁済期限とは、信用建玉に対して当社がお客様に信用を供与する期限をいいます。弁済期限は、現在のところ6ヶ月のみを取り扱っています。

弁済期限が6ヶ月であるということは、信用建玉の建日（信用建玉が約定した日）の6ヶ月目応答日が信用期日となり、この日を超えて建玉を保有することは法律で禁じられてい

ます。信用期日が休日の場合には、直近の前営業日が信用期日となります。

以上の日本語仕様において、弁済期限や決済日、信用期日などの用語が出てきたが、それぞれどのような意味があるのか、それぞれどのような関係があるのか、などの疑問が生じてくる。日本語などで仕様を記述すると少なからずこのような曖昧さがでてきてしまい、この仕様を見る人によって捉え方が異なってしまうリスクがある。この点においてVDMで仕様を記述することは以上で挙げた曖昧さを排除することができ、欠陥発生数やそれに伴う開発期間の延長という問題を改善することが出来る。プレゼンの最後には、VDMを誰に対しても教育することができ活用させることができるか、というお話があった。佐原さんは肯定的な立場を取られており、VDMの教育可能性を述べていた。この点については、後の討論で賛否両論の意見が出された。

続いて、東京工業大学大学院経営工学専攻 飯島教授から「フォーマルアプローチの限界はどこにあるか？」というテーマでお話いただいた。

お話の始めに飯島教授の専門の一分野である（一般）システム理論に関する説明がされた。一般システム理論の目的は、異なる分野における概念や法則間の同型性を追及するというもので、それにより各分野への援用や研究の重複をなくし共通言語を提供するというものである。このような考え方は、形式仕様言語のようなフォーマルアプローチと親和性がある。しかし一般システム理論は実世界（特に工学の世界）では普及せず、その理由として以下の3つの点が挙げられた。

- ① 数理論理的表現は一般的には理解することが難しい
- ② 対象とする事象が second order であるので、抽象的となりイメージがつかみにくい
- ③ 現実の世界まで具体化し活用することが難しい

以上の点のいくつかはフォーマルアプローチにも同様にいえる。例えば、①及び②は、フォーマルアプローチの理解・学習の難しさの一因ともいえる。

また、VDMをはじめとする形式仕様言語は、予測可能な例外発生に対する対応とシステムの頑健性に対する保証に対しては有効だが、予測不可能な例外・失敗には対応することが出来ない。このような例外は、仕様をVDMで規定する前段階の問題でありその責任は組織に依存している。この点がフォーマルアプローチの限界であり、この点をカバーするために組織体を含めた全体的なリスク対応が必要である。よってフォーマルアプローチ利用者は、その強みと弱みを意識しそれだけでは万能ではないことを理解する必要があると述べた。

プレゼンテーションの部の最後にNTTデータの山本さんから、システムリスクとフォーマルアプローチの関係に関してのお話をいただいた。

お話の中で印象的だったのは、リスクとは影響度と発生率を乗じたものであり、たとえ発生率が低くても影響度が高ければ大きなリスクとなりうるというものであった。例えば、昨年に起こった東証のシステムダウンなどのリスクは、発生率はきわめて低いと想定され

ていたであろうが、実際に起こったときの影響度は甚大であり結果として大きなリスクであった。このように、影響度と発生率の両面からリスクを考えることは当然ではあるが最も重要視して考える事柄であると感じた。

また、フォーマルアプローチの適用対象として IT システムに限らず、経営分野や法制度まで含めた幅広い分野での適用の概念を紹介された。法律などの分野にもフォーマルアプローチを用いようとする山本さんの考え方は、会場内でさまざまな議論がなされたが私個人としてはユニークな視点であると感じ、実際どのように実現できるのか興味が湧いた。

2. 討論の概要

Coffee Break を 30 分挟んで野中さんによるプレゼンテーションと、フォーマルアプローチの効果と限界についてのパネルディスカッションが行われた。これらの議論で交わされた主な要点は、VDM は教育すれば誰でも使いこなすことができるかという点と VDM の使用機会に関する点であった。

VDM の教育可能性については、どの程度で使いこなせたといえるかという境界の引き方の違いによって賛否が分かれた。フォーラムに参加されていた法政大学の劉教授の研究室では学部 4 年の学生全員に VDM を用いて開発をさせた結果、VDM の表面的な利用、つまり VDM の記号や構文規則の基本的な理解は全員理解できたが、VDM を効果的・実用的なものとなるように表記できる段階に達していたのは、1、2 名程度しかいなかったという。よってフォーマルアプローチを、表面的・機械的な理解だけでなく効果的な利活用まで理解できるようになるには、相応の学習期間とある程度の才能が必要なのではないだろうか。

また、VDM の使用機会についての討論は、VDM を含めフォーマルアプローチは確かにそれだけで全てのリスクをカバーしたりエラーをなくしたりすることはできないが、それでもフォーマルアプローチが使えるところで使わないケースが多すぎるという意見があった。

3. 私見

大学院生という私の立場から、今回のフォーラムの参加は今後の研究活動を行っていく上で有用な示唆をいただいた。私の研究のアプローチは、情報システムに対しフォーマルアプローチを用いてシステムを抽象化しシステムの特徴・性質を研究するというもので、現在 BPM(Business Process Management)を対象に研究を行っている。

今回のフォーラムでは実社会でのフォーマルアプローチの有用性についての議論がされたが、大学でフォーマルアプローチを学び研究に利用している立場として、社会でもっと普及し活用してもらいたいという気持ちであった。また、討論の中で VDM を学習するメリットの 1 つとして、学習を通して抽象的な思考力をつけることができそのような思考力は社会の様々な場面で応用できるというものがあったが、それを聞いて、私が所属する研究室での論理的・抽象的な思考のトレーニングが社会に出ても活用できると感じ光栄だった。

最後に、今後 VDM を社会でより普及させるために私が感じたことを述べる。VDM は抽象的で数学的な思考力を必要とするため、PM や SE は従来のやり方と親和性が少なく必要以上に難しいと感じてしまうと思われる。また、現在のプロジェクトやシステムが VDM を利用するのに適しているかどうかを判断することができないため、より一層 VDM を導入することにためらうようになってしまうだろう。

この対処策としては、VDM が現在のプロジェクトにおいて導入すべきであることを明確かつ客観的に示す必要があると思われる。つまり、いままでの VDM の成功事例や科学的な根拠をもとにして、企業の業種・規模・その他特性などを分割し、どのような業種・規模のシステム開発において VDM を利用したらメリットが得られるかという見取り表を作成することが有用であると考えます。このような見取り表が存在することで、VDM を導入するかどうかの判断が客観的に明確となり、導入の強いインセンティブになりうると思われる。

新連載

ソフトウェア・エンジニアリング入門

Software Engineering



新谷勝利

(しんたに かつとし)

独立行政法人 情報処理推進機構 (IPA, <http://sec.ipa.go.jp/>)
ソフトウェア・エンジニアリング・センター (SEC) プロジェクト
総括グループ所属。1984年に米IBMソフトウェア・エンジニア
リング研究所でソフトウェア工学の訓練を受け、同社ソフトウ
ェア開発部門や日本IBM大和研究所ソフトウェア開発部隊への技
術移転を担当。社外では、国際基督教大学で教鞭をとったり、ソ
フトウェア技術者協会 (SEA, <http://www.sea.jp/>) の幹事を
務めたりしている。現在所属するSECは、ソフトウェア工学の普
及を目的として、2004年10月に設立された組織。

第1回

ソフトウェア開発を支える “ディシプリン”

ここ数年、情報システムのトラブルが頻発するのを受けて、ソフトウェアの品質を向上させるための取り組みである「ソフトウェア・エンジニアリング (ソフトウェア工学)」が注目されています。ソフトウェア工学はともすれば、学術的なものというイメージでとらえられがちです。“難しくて高尚なもの”であり、実際の開発現場ではあまり役に立たない、と考える人もいるかもしれません。あるいは、重要なのは理解できるが、目の前の仕事を片付けたり、新しい技術を習得するのに手一杯でとてもそこまでは手が回らない、という人も多いでしょう。

しかし、ソフトウェア開発に携わるすべての人にとって、ソフトウェア工学が基礎的な素養であることは間違いありません。この連載では全3回で、ソフトウェア工学の基本について、情報処理推進機構 (IPA) ソフトウェア・エンジニアリング・センター (SEC) の方々にわかりやすく説明していただきます。抽象度が高い話も出てきますが、できるだけ皆さんの身近な活動に結び付けられるように解説してもらいますので、最後までお付き合いください。ソフトウェア工学の基本を身に付けて、広い視野を持ったプログラマーになってください。(編集部)



最近のソフトウェア開発に関連する話題が、新聞や雑誌などのメディアをにぎわしています。それは、ソフトウェア開発の遅れ、ソフトウェア品質の問題、さらに、ソフトウェアの運用が企業活動そのものに影響を与えている、といったことです。開発の遅れではシステム統合の遅れにともなう企業統合自体の遅れが、品質に関しては組み込み製品のリコールがあります。運用ではオペレーション・ミスによる巨額の損失などが、本当に次から次へとといった感じで報告されています。

実は、こうした現象は、ネットワークを活用する大規模ソフトウェアが開発・運用され始めた1960年代後半にはすでに、世界的な規模で認識されていました。そして、その対策として1960年代末には「ソフトウェア・エンジニアリング」という用語が案出されるに至りました。しかし、それから30年以上経過した今になっても、似たような過ちは繰り返されています。なぜでしょうか？ 私はソフトウェア・エンジニアリングが理解されていないし、理解させようとする努力が不十分であったと思っています。この連載によって、皆さんがソフトウェア・エンジニアリングに目を向けてくださることを期待しています。

航空機などで事故が起こったときには、事故調査委員会が原因を調査し、同じようなトラブルが再発しないように徹底した手段が取られるはずです。同様に、ソフトウェア業界においても、失敗から学ぶという姿勢を持たなければなりません。

ソフトウェア・エンジニアは、ソフトウェアの開発、運用、保守にかかわる専門家です。しかし多くのソフトウェア・エンジニアはこれまで、専門家であれば当然行わねばならない「過去からの学習の蓄積と活用」に積極的ではなかったと言えます。「暗黙知」を「形式知」に変換し、多くの仲間と共有できる共有財産というべきものを、積極的に作ってこなかったのではないのでしょうか。

ここでもう一度、ソフトウェア開発における本質的なもの、すなわちソフトウェア・エンジニアリングについて学習することは、非常に意義があるものと考えます。「ソフトウェア・エンジニアリング入門」連載の第1回である今回は、ソフトウェア・エンジニアリング

とは何なのか、ということについて、例を示しながら説明していきます。

もっとも、この記事で説明するのは、あくまでも基本的なことです。多少なりとも興味を持った人はぜひ、「SWEBOK」などの専門書を読むことをお勧めします*1。こうした専門書はともすれば「無味乾燥で読みにくいもの」と思われがちですが、この連載を読んだ後であれば、多少なりとも読みやすくなっているはずです。

ソフトウェア・エンジニアリングは開発者の「基本的素養」

一口でソフトウェア・エンジニアリングと言っても、扱う対象はきわめて広い範囲にわたります。電気・電子・情報分野における世界最大の学会である米IEEE (Institute of Electrical and Electronic Engineers) は、ソフトウェア・エンジニアリングを以下のように定義しています。

- (1)ソフトウェア・エンジニアリングは、ソフトウェアの開発、運用、および保守におけるシステムティックであり、ディシプリンに基づいた定量的なアプローチの適用である。換言すれば、ソフトウェアへのエンジニアリングの適用である。

(2)ソフトウェア・エンジニアリングは、(1)で示したアプローチに関する研究である。

初めて読んだ人には、わかるようなわからないような定義ですね。しかし、今回の解説を読み終えたら、この定義が何を伝えたいかを理解できると思います。それでは早速、解説を始めましょう。

ソフトウェア開発において本質的なものとは何でしょうか？ それは「ソフトウェア開発の過程において、私たちが実行しなければならないと共通に認識されることがら」です。このときに用いられる概念が「ディシプリン (discipline)」です。「discipline」という言葉を英和辞典で引くと、「躾 (しつけ)」という訳語が当てはめられています。すなわちソフトウェア・エンジニアリングとは、「ソフトウェア開発にかかわるものが躾あるいは基本的素養として身に付けておくべきものの集大成」

*1 体系的に勉強したい人には、「ソフトウェア・エンジニアリング基礎知識体系—SWEBOK2004」(松本吉弘訳、オーム社発行、2005年)と、「ソフトウェア工学の基礎」(玉井哲雄著、岩波書店発行、2004年)の2冊をお薦めします。前者は、IEEE (米国に本部を置く電気・電子・情報に関する学会で、アイ・トリプル・イーと呼称されています)の委員会におけるソフトウェア・エンジニアリング分野での活動成果をまとめたものです。幅広いソフトウェア・エンジニアリングの知識分野への「ポイント」を提供することで、多くの参考文献が分野ごとに整理され紹介されています。参考文献のいくつかは日本語訳がありますが、ほとんどは翻訳されていません。「ソフトウェア工学の基礎」は、最初から日本語で執筆・発行された書籍で、SWEBOKで定義している知識分野における具体的な手法について例を挙げて説明しています。

*2 対象をモデル化し、そのモデルを容易にかつ厳密に記述することを目的とする言語を、「仕様記述言語」、「形式仕様記述言語」、あるいは単に「形式言語」と呼びます。形式言語の一つであるPDL (Program Design Language) は、IBMの米連邦政府システムズ事業部ソフトウェア・エンジニアリング部門で、Pascalをベースにして1980年代初頭に開発されました。形式言語としてはほかに、1973年にIBMのウィーン研究所で開発されたVDM-SL (Vienna Development Method Specification Language) が有名です。VDM-SLの日本語版が現在、商用アプリケーション開発や組み込みソフトウェア開発に使われている事例があります。VDM-SLについては、J.フィッツジェラルドなど著、荒木啓二郎など訳の「ソフトウェア開発のモデル化技法」(岩波書店、2003年)を参照してください。

*3 ある変数を取りうる値の範囲を定義しており、これをデータ空間といいます。

*4 ここにおけるプレース・ホルダーは、大ざっぱに言って変数と同じ意味であると考えて差し支えありません。

ということになります。

以降では、ソフトウェア・エンジニアリングを構成するディシプリンのうち、代表的な四つについて説明します。

- (1) WHATとHOWを分離する
- (2) 段階的に詳細化する
- (3) ソフトウェア開発をライフサイクル・プロセスでとらえる
- (4) 定量化およびメトリクスに基づいて管理する

WHATとHOWを分離する

ソフトウェア開発におけるWHATとHOWの分離は、ソフトウェア・エンジニアリングの代表的なディシプリンです。どんなに簡単なコードであっても、バグが発生する可能性はゼロにはなりません。バグを発生させないようにするには、バグがどのような場合に発生するかを理解し、対応策を考えなければなりません。

それも、早期のほうがより良いのです。例えば、(1)プログラムを書く前の設計の段階で、(2)設計書を書く前の仕様の段階で、(3)仕様書を書く前の要求の段階で、という具合です。前述の組み合わせにおいて、前段階のものがWHAT、後段階のものがHOWに当たります。WHATとHOWの分離というディシプリンを適用することで、バグの発生する可能性についてより深く考えることをソフトウェア開発の過程でエンジニアが意識するようになります。バグの発生を未然に防げるようになるのです。当たり前のことのように感じるかもしれませんが、しっかり理解していただくために例を挙げて説明していきましょう。

リスト1をご覧ください。これは、思考を容易に、かつ厳密に表現できるように開発された形式言語の一つである「PDL (Program Design Language)」*2を用いてプログラムを記述した例です。1から3までカウントアップ

していくプログラムで、ちょっと見ると簡単に見えます。しかし、こんな簡単なプログラムにもバグは入りこむのです。

リスト1でWHATに当たるのは、< bumpの仕様 *3>のブロック・コメントです。一方、beginからend bumpまでのコード・ブロックがHOWに当たります。以下ではこの例を通じて、ブロック・コメントでは表現しきれないものがあること、そして、開発者がそのことに気づかないことについて説明していきます。それによって、WHATとHOWを分離するにはどうすればよいのか、ということが理解できるようになります。

まず、このプログラムの機能について考えてみましょう。プログラムの機能は、入力と出力のペアで表現できます。リスト1におけるHOWの部分は以下に示す入力と出力のペアで表現されます。

入力 → 出力

1	2
2	3
3	4

リスト1では、入力と出力はいずれも、counterは、1..3の範囲の値*3を持つcounterというプレース・ホルダー (place holder) *4を用いると定義されています。

説明のためにここでは入力と出力を、(入力値、出力値)という一括りのカッコで表すことにします。すると、(1, 2) および (2, 3) のペアで定義される機能は、counterの定義の範囲内で成立しますが、(3, 4) は範囲を超えてしまうことがわかります。

こうした不具合は、ユニット・テストでプログラムを実行して初めて見つかるのでは遅過ぎます。コーディング時に、こうした制約に気づくように設計がされている必要があります。

先ほど、仕様はWHATを表現すると述べました。したがってこの不具合は、仕様をまと

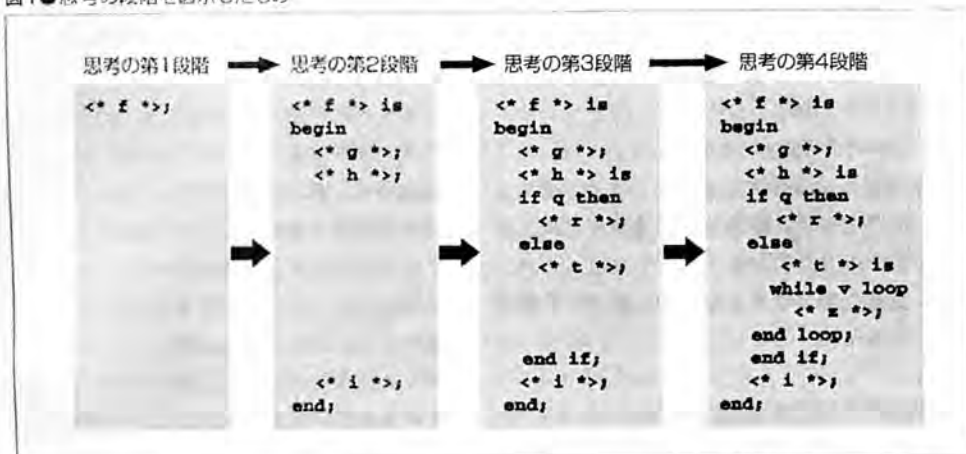
リスト1 ●形式言語を用いて記述したプログラム

```
type one_to_three is range 1..3;
procedure bump (counter: in out one_to_three)
< * bumpの仕様 *3 > is
begin
  counter := counter + 1;
end bump;
```

リスト2 ●リスト1のブロック・コメントに解決策を明示する例

```
< * counter < 3 -> counter := counter + 1
  | ? * >
```

図1●思考の段階を図示したもの



*5 type宣言でデータ空間を1から3と指定していますので、「counter = 3」と「counter > 3」の二つの条件を考慮しなければならないが議論の余地があります。これは、言い換えれば、1から3までというデータ空間をどのように保証できるかという問題に帰結します。

*6 ここで説明したディシプリンは、データ空間（データの取り得る範囲）のあいまいさに関する処理に適用できます。例えば、2005年11月に起こった東京証券取引所における取引全面停止という重大トラブルの原因となった事象にも有効だと考えられます。

める段階で解決されている必要があるのです。では、どうしたら、この問題にあらかじめ気づいて、解決できるようになるのでしょうか？

それには、リスト1のブロック・コメント（<* bumpの仕様 *>の部分）において、入出力値（3, 4）のペアで発生するであろう問題に対する解決策を明示しておく必要があります。PDLを用いる場合、具体的にはリスト2のように記述します。

リスト2の1行目はcounterの値が3より小さい場合には、counterに1を加えた値をcounterに代入することを示しています。2行目の「|？」は、counterの値が3以上の場合*5に、プログラムが何をしたいのかが不明である、ということを示しています。

コンピュータ・プログラムに「？」（何をしたいのかが不明）が存在することは許されません。したがって、このように仕様を記述することで、「？」の存在をコーディングの前にあらかじめ認識し、それに対して何かを決めなければならないことがわかります。

この例は簡単に見えますが、WHATとHOWの分離というソフトウェア・エンジニアリングのディシプリンと、それを実現するための手法を学習する必要があることをわかっていただけるのではないかと思います*6。

段階的詳細化で仕様を明確にしてい

先ほどの「|？」の部分は、どのように展開すればよいのでしょうか。「|？」のままでよいわけではありませんが、すべての詳細を仕様

の段階で決定できるとは限りません。私たちはこのことを、経験を通して知っています。

「？」は仕様の段階で明解にするのが望ましいのですが、遅くとも設計の段階では明確化しておく必要があります。この明確化をするときに登場するのが、「段階的詳細化」というディシプリンです。

段階的詳細化は、思考の段階でもあります。図1の例で考えてみましょう。思考の第1段階では、全体の機能<* f*>が認識され、第2段階では、それが、<* g*>、<* h*>、<* i*>の三つの機能に分解されています。第3段階になると<* h*>がさらに詳細化され、続く第4段階では<* h*>のサブ機能である<* t*>がさらに詳細化されています。

ここでは、当初三つに分割された機能が、それぞれに詳細化されるタイミングが異なるであろうということに注目してください。実際私たちは、気になる部分の詳細化をほかに先がけて実施することにより不安を取り除いています。

この例からわかるように段階的詳細化は、WHATとHOWの分離を思考の段階で構造化したものであると言えます。これは「構造化手法」として知られているディシプリンであり、仕様作成、設計、コーディングのそれぞれの段階で適用されるものです。

ソフトウェア開発は、仕様作成→設計→コーディングと段階を踏むにつれて、思考が実行可能な形式として詳細化されてゆく過程と考えることができます。次に、この段階的な詳細化がソフトウェア開発ライフサイクル・

*7 ISO/IEC 12207, ソフトウェアライフサイクルプロセス, (財) 日本規格協会, 1995年。

*8 ウォータフォール型モデルは, 1970年にW.W. RoyceによりIEEE WESCONで発表されています。この歴史的発表論文「MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS」は, 次のWebサイトから入手できます。
<http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>

プロセスにおいてどのように適用されているかについて説明しましょう。

ソフトウェア開発をライフサイクル・プロセスでとらえる

ソフトウェアを開発する際には, ここまでの例で説明した仕様書作成, 設計書作成, コーディング以外にも, 様々な段階を踏む必要があります。ソフトウェア・エンジニアリングの原典とも言えるSWEBOKの2004年度バージョンであるSWEBOK2004は図2のような構成になっています。ソフトウェア・エンジニアリングのディシプリンの一つに, 「これらの主プロセスあるいはソフトウェア開発ライフサイクル・プロセスというスコープ(範囲)でソフトウェア開発に関与する」というものがあります。コーディングを担当するプログラマは自分が担当する範囲にしか目を届かせない傾向がありがちです。このディシプリンに基づいて, ライフサイクル・プロセスのスコープを持ってソフトウェア開発に当たる必要

があります。

もう少し詳しく説明しましょう。図2の各項目は, ソフトウェア・エンジニアリングが対象としているものです。(1)の各項目はソフトウェア開発における主プロセスと言われているものです。それらを効果的に実施するためには, (2)および(3)が必要とされます。これらは, すべてソフトウェアを開発, 運用, 保守する際の「活動(アクティビティ)」を調査・分析し, ある括りとして認識したものです。この括りのことをプロセスといいます。

プロセスという用語は, 種々の分野で異なった意味で用いられています。国際標準化機構(ISO, International Organization for Standardization)における定義は, 「プロセスとは, 互いに関連を持った活動の集合で, 入力を入力に変換するもの」となっています*7。ソフトウェア開発, 運用, 保守はこのプロセスの集合であり, ソフトウェア開発の始まりから破棄までの「生涯」ということで, しばしばライフサイクル・プロセスと言われています。

ライフサイクル・プロセスは経験的に, 要求→仕様作成→設計→構築→テスト→運用・保守, という時間軸で考えられます。括りの抽象度を上げると, こうした順序で仕事が進められているように見えます。

このことによって, ソフトウェア開発は, 上流・中流・下流過程へとまるで「滝」のように順序だって進められるというように認識され, 「ウォータフォール型モデル」*8と呼ばれることが多くなっています。

しかし前述の定義に見られるように, プロセスは「変換の括り」であり, 「変換の順序」ではありません。すでに「思考の段階と詳細化」の中で説明したように, 思考や思考の実装は, 時間によって一律に進められるものではありません。ある部分はほかの部分より先に詳細化が進められるのが普通です。例えば, ある機能が取り出せるとして, その機能の技術上の検証, あるいはフィージビリティ・スタディを先にしたい場合には, その部分だけを, 設計→コーディング→テストします。また, 分割した機能を別々のグループで独立して並行開発する場合もあります。

こうした状況を踏まえたプロセスのネットワークを作ることにより, 実際の開発の流れ

図2●ソフトウェア・エンジニアリングが対象とする項目(SWEBOK2004より)

(1)ソフトウェア開発の時間軸でのある塊で, 一般的に主プロセス, あるいはエンジニアリング・プロセスと言われているもの

- ・ソフトウェア要求
- ・ソフトウェア設計
- ・ソフトウェア構築
- ・ソフトウェア・テスト
- ・ソフトウェア保守

(2)ソフトウェア開発の主プロセス, あるいはエンジニアリング・プロセス全体を通して, その実施を支援する支援プロセスと言われているもの, およびそこで使用される手法など

- ・ソフトウェア構成管理
- ・ソフトウェア・エンジニアリング・マネジメント
- ・ソフトウェア・エンジニアリング・プロセス
- ・ソフトウェア・エンジニアリングのためのツールおよび手法
- ・ソフトウェア品質

(3)さらに, 全体にかかわり基本的な考え方, 具備しておかなければならない知見

- ・ソフトウェア・エンジニアリングに関連するディシプリン

に合致したライフサイクル・プロセスにすることが出来ます。今後は新規開発より保守開発が増えるでしょうから、以下の保守まで考えたさらに複雑なライフサイクル・プロセスに言及します。

運用・保守の段階に入って要求が明確になる事例はいくらでもあります。むしろ、システム開発の最初の段階で保守の段階の詳細までわかっていることが稀であるということは読者の皆さんがよくご存じでしょう。これはすでに、1960年代のIBMのOS/360の開発の状況を研究したM. Lehmanにより、「ソフトウェアの進化」*9という用語で説明されています。

定量化およびメトリクスに基づいてソフトウェア開発を管理する

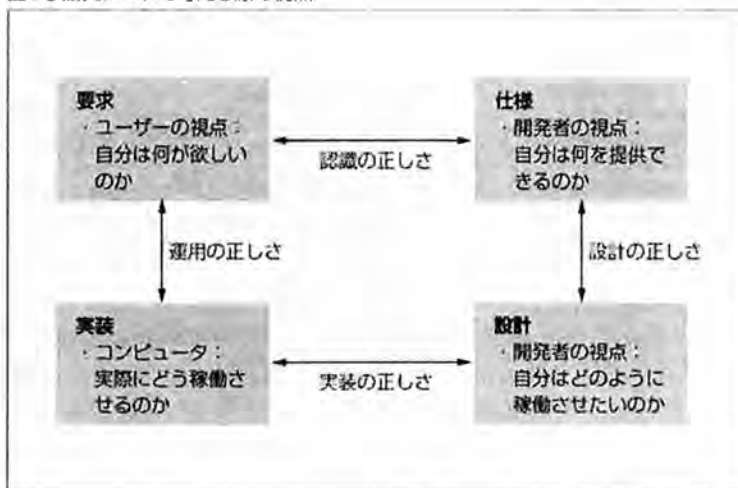
最後に、ソフトウェア開発にかかわる活動が適切になされているのか、段階ごとに適切な成果を生んでいるのか、といったことをどのようにして保証するのかを考えてみましょう。これは「定量化およびメトリクス(metrics, 計量)」というソフトウェア開発管理のディシプリンが該当します。このディシプリンは、SWEBOKにおいてソフトウェア・エンジニアリングの一構成要素として定義されています。

一般にエンジニアリングは、管理対象を測定し、数値化するアプローチを現場に導入することにより、QCD(Quality Cost Delivery, 品質・コスト・納期)の改善を図るものです。的確に管理をするためには、対象を表す数値が認識されていなければなりません。ここでは、定量化およびメトリクスの一分野として、図3を通して品質について考えてみましょう。

図3には「認識」「設計」「実装」「運用」という、四つの正しさが定義されています。それぞれの正しさを定量的に確認できれば、品質を高めることができます。しかし、それぞれが異なる視点とその視点を表現する「言語」を用いている場合、正しさを正確に議論することはどの程度可能でしょうか？

ほとんどの場合、要求と仕様は自然言語で記述されるので、本質的にあいまいさを含んでいます。設計も、「WHATとHOWを分離する」で説明に用いたような形式言語を用いない限り、あいまいさが混入する可能性があります。

図3●品質について考える際の視点



ます。要求の表現にあいまいな自然言語を用いている限り、仕様や設計でいかに形式度を上げていったとしても、プログラム・コードが実現するものと元の要求との間に食い違いが生じることは容易に想像できます。それにもかかわらず、ソフトウェア開発においては、品質を上げることが求められています。それは、どうすれば実現できるのでしょうか。

一つのアプローチは、前述したように仕様、設計に形式言語を導入することです。一方、一般的に取られているアプローチは、要求機能の分割と分割したそれぞれにおける段階的詳細化の各ステップで、直前と直後の間で正しさを検討することです。これには、「レビュー」や「インスペクション」が有効とされています。また、実装の段階では、テストが有効です。

こうした各手法を実施すると、評価および測定データを得ることができるので、これらに対して定量的なアプローチをするわけです。定量的アプローチは、最後のテスト段階で表面化する不具合の検知と対策において採用するのでは遅すぎます。それぞれの正しさを判定する段階で、適用しなければなりません。

これは、ソフトウェア開発ライフサイクル・プロセスに沿って開発活動を進めているときに、実行中の状況を少なくともマイルストーンの段階ごとに「見える化」していくということにほかなりません。連載の次回は、これらについて議論するときに用いる定量化およびメトリクスについて説明します。【●】

*9 M.M. Lehman, L.A. Belady, Program Evolution, Academic Press, ISBN 0-12-442441-4, 1985.

編集後記

☆

船便状態からなんとか脱却すべく、まだ努力中です。

☆☆

この号はもう少し早く印刷に回す予定だったのですが、SS2006 が間に入ったために遅れました。

☆☆☆

冒頭のレポートは Golden Week 明けに開かれた久しぶりの SEA SPIN Meeting の記録です。

☆☆☆☆☆

次のレポートは、5月東京そして6月大阪と続けて2回開催された Aspect 指向 Forum の報告です。企画された小林修さんと玉井哲雄先生に、それぞれ力のこもった報告およびエッセイを書いていただきました。

☆☆☆☆☆☆

3番目のレポートは、大型システムのトラブルとフォーマル・アプローチをテーマとして5月末に開催された特別 Forum の報告です。

☆☆☆☆☆☆☆

巻末に数録した論文は、幹事会メンバーの1人である新谷勝利さんが日経ソフトウェア誌に書かれたものの転載です。

☆☆☆☆☆☆☆☆



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/>