SEAMAIL

Newsletter from Software Engineers Association

Volume 11, Number 8 1998

目　次

編集部から		1
SIGENV PostSS Workshop '97 レポート		2
CRC カード手法を用いた開発事例を題材に		
1997 年 6 月 21 日 (土) @ リバーサイドホテル (熊本)		
オープニング: CRC カード手法の概要	武田 泰英	4
事例発表 1: 中規模開発での CRC 使用事例	武田 泰英	8
事例発表 2: オブジェクト指向をどう現場に導入すべきか	古閑 幸一	18
利用ツール紹介: CRC と開発プロセス	武田 泰英	28
全体討論		40

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飛躍的に増加し，現在，北は北海道から南は沖縄まで，500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30社を数えます．支部は，東京以外に，関西，横浜，長野，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： 坂本啓司

常任幹事： 荒木啓二郎　高橋光裕　田中一夫　玉井哲雄　中野秀男　深瀬弘恭

幹事： 市川寛　伊藤昌夫　大場充　河村一樹　窪田芳夫　熊谷章　小林修　桜井麻里
酒匂寛　塩谷和範　篠崎直二郎　新谷勝利　杉田義明　武田淳男　中来田秀樹
布川博士　野中哲　野村行憲　野呂昌満　端山毅　平尾一浩　藤野誠治　二木厚吉
堀江進　松原友夫　山崎利治　和田喜久男

事務局長： 岸田孝一

会計監事： 辻淳二　吉村成弘

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　中園順三
ネットワーク分科会(SIGNET)：小林俊明　人見庸　松本理恵
プロセス分科会(SEA-SPIN))：青山幹雄　伊藤昌夫　坂本啓司　高橋光裕　田中一夫　増井和也

支部世話人 関西支部：臼井義美　中野秀男　横山博司
横浜支部：野中哲　藤野晃延　北條正顕
長野支部：市川寛　小林俊明　佐藤千明
名古屋支部：筏井美枝子　石川雅彦　角谷裕司　野呂昌満
九州支部：武田淳男　平尾一浩
広島支部：大場充　佐藤康臣　谷純一郎
東北支部：河村一樹　布川博士　野村行憲　和田勇

賛助会員会社： ジェーエムエーシステムズ　東芝アドバンストシステム　ＳＲＡ　ＰＦＵ
東電ソフトウェア　構造計画研究所　さくらケーシーエス　サン・ビルド印刷
富士通　新日鉄情報通信システム　オムロンソフトウェア　カシオ計算機
キヤノン　中央システム　安川電機　富士通エフ・アイ・ピー
ＳＲＡ東北　アスキー　新日本製鉄エレクトロニクス研究所　ダイキン工業
東北コンピュータ・サービス　オムロン　アイシーエス　ＳＲＡ中国　日本電気ソフトウェア
富士電機　ブラザー工業　オリンパス光学工業　リコー　（以上29社）

ＳＥＡＭＡＩＬ　Ｖｏｌ．１１，Ｎｏ．８　　１９９８年８月２０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　sea@sea.or.jp
印刷所　有限会社錦正社　　〒１３０　東京都墨田区錦糸町４－３－１４
定価　５００円（禁無断転載）

編集部から

☆

めずらしく船便でない SEAMAIL です.

☆☆

昨年６月のソフトウェア･シンポジウム'97 の後，熊本で行なわれた SIGENV(環境分科会) のワークショップ･レポートがタイミングよくできあがってきてくれました.

☆☆☆

ワークショップやフォーラムなどのイベントは，企画･開催･運営もなかなかたいへんですが，終った後のレポートのとりまとめは，それにもましてしんどい作業です.

☆☆☆☆

それはちょうど開発プロジェクト完了後のドキュメント書きと同じように，われわれの多くがもっとも苦手とする仕事のようです (?).

☆☆☆☆☆

その意味で，今回のレポートのとりまとめをほとんど１人でやられた田中慎一郎さん，ほんとうにご苦労さまでした.

☆☆☆☆☆☆

他の分科会や支部のみなさんも，ぜひ，それぞれのイベントの報告をどうぞ.

☆☆☆☆☆☆☆

SIGENV PostSS Workshop '97 レポート

ＣＲＣカード手法を用いた開発事例を題材に

1997 年 6 月 21 日（土）@リバーサイドホテル（熊本）

ワークショップレポート全体概要：

CRC（Class/Responsibility/Collaborator）カード手法とは，あるクラス（オブジェクト），その責務，実現するための協力者を書き込めるようになっているカードを用いたオブジェクト指向分析／設計手法です．この手法は、非常に単純かつ体験的な手法であるため，他の手法にない長所を備えることが評価されている反面，これまでは日本では実際に開発に適用されたという事例を目にすることがほとんどありませんでした．

我々，ソフトウェア技術者協会環境分科会(SIGENV)では，以前よりこの手法に注目していた経緯もあり，今回のワークショップで同手法を用いた開発事例を２例ご発表いただき，同手法を実際の開発に適用する際に見られる効果や問題点／留意点について討論を行ないました．大変遅くなりましたが，ここにその発表内容と討論の記録をご報告いたします．

ワークショップでは，まず参加者の知識整理のために事例発表者の武田さんに「CRC カード手法の概要」を解説いただき，つづいて最初の事例「中規模開発での CRC 使用事例」をご発表いただきました．ここでは，主に CRC カード手法を適用した開発時にどのような現象が発生するか，従って導入にはどのような注意が必要かなどをまとめていただいています．

次に，２番目の事例として古閑さんに「オブジェクト指向をどう現場に導入すべきか」のタイトルでご発表いただきました．古閑さんには，別の経験を基に，CRC カード手法導入の動機や経緯，問題点，既存の開発手法にどう取り入れるべきかなどをまとめていただいています．

両事例からは，導入時に考慮すべき具体的な内容が伺え，今後適用を考える組織にとって大変参考になる内容と思われます．

続いて，当初の予定では２事例に対する総合的な質疑応答を考えていましたが，武田さんが同手法を実際に適用された際の「プロセス」に関しても資料を作成していただいていると伺い，「CRC と開発プロセス」のご発表もお願いいたしました．

同発表では，CRC カード手法による開発プロセスを支援するためのツールの試作についてもお話いただきました．CRC 手法は単純であるだけに，このような情報の記録／整理／検索のための支援ツールが不可欠であるとも考えられ，大変興味深い内容です．

残りの時間は発表者にも何らかのフィードバックが行えるよう，質疑から離れて視点を変え，全体討論を行いました．取り上げたテーマは 1) セッション実施時の工夫について，2) カードや責務の整理方法について，3) 大規模システムへの適用に関して，となりました．

発表および討論は，それぞれセッション毎に独立してまとめてあります．参加者一覧と当日のプログラムを下に，当ワークショップの参加者募集チラシを記録のために全体討論の後に添付しました．本報告が何らかの形で皆様のご参考になれば幸いです．

ワークショップ参加者：

	氏 名	所 属
1	遠藤 新太郎	メディアドライブ（株）
2	古閑 幸一	ヒラタソフトウェアテクノロジー（株）
3	後藤 浩	（株）シスプラン
4	小林 修	（株）ダダ
5	塩谷 和範	（株）SRA
6	薗田 圭二郎	（有）ソフトベア
7	武田 泰英	（株）アドバンテスト
8	田中 慎一郎	（株）SRA
9	中松 都勇	（有）ソフトベア
10	藤尾 好則	熊本県立大学
11	渡邉 雄一	（株）アスキー

プログラム（実績）：

時間帯	セッション	発表者など
09:30 - 10:00	オープニング CRC カード手法の概要	武田 泰英
10:00 - 11:00	事例発表 1 中規模開発での CRC 使用事例	武田 泰英
11:00 - 12:00	事例発表 2 オブジェクト指向をどう現場に導入すべきか	古閑 幸一
12:00 - 13:00	（昼食休憩）	
13:00 - 14:00	利用ツール紹介 CRC と開発プロセス	武田 泰英
14:00 - 17:30	全体討論 ・セッション運営方法 ・分析から設計／実装への転換 ・大規模開発への適用	参加者全員

CRC カード手法の概要

(株) アドバンテスト
武田　泰英

1. CRC カード

CRC カードとは，図1に示すような 6in. x 4in. のカードです．
このカードは，Class，Responsibilities，Collaborators という3つの部分からなり，クラスの挙動とクラス間の関係を表せるようになっています．

1) Class

Class のところにはクラス名を書きます．後ほどご紹介しますが，通常は，まず対象アプリケーションを実現する上で不可欠だと思われるクラスを見つけ出しておき，それ毎にカードを用意しておく形をとります．

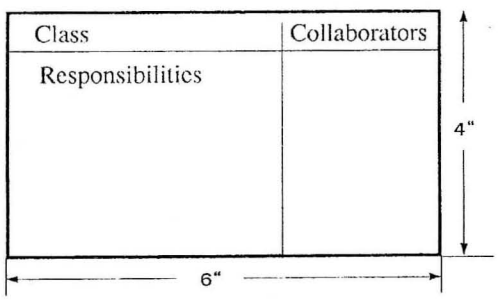

図1 CRC カード

2) Responsibilities

Responsibilities は，実行すべき責任です．このクラスがどういったことをやるのか，どういった責任を持っているのか，あるいは回りに対してどういうサービスを行なうのかを書きます．

3) Collaborators

Collaborators とは，協力者のことです．書かれた責任を実行する際に 一部の作業を他のクラスに請け負ってもらう場合，その相手クラスの名前を書きます．

このような形で，カードに責任や協力者を書いていくことによって，モデル化を進めていきます．基本的に，責任と協力者，すなわち「やることと関係」だけに着目することになっていて，データ構造などは前面に出てきません．

例を示します．あまりうまい例ではありませんが，Display というクラスを考えます．

このクラスの責任は，「表示する」と「再表示する」です．Display は，例えば指定された対象の内容を画面に表示します．一方，Contents というクラスは「内容を教える」という責任を持っています．このことから Display は Contents の協力を得て表示対象の内容を知り，それを表示することができます．

この関係を表すために，Display のカードの Responsibilities のところに「表示する」，「再表示する」という責任名を書き，

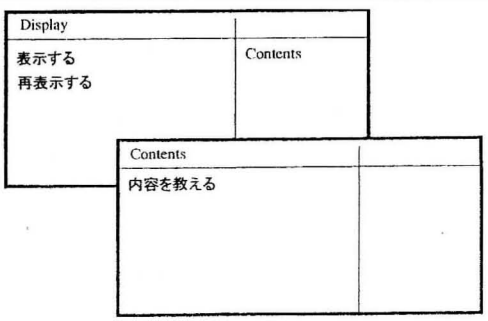

図2 CRC カード（記述例）

Colaborators のところには「Contents」というクラス名を書きます．そして協力者である Contents のカードの Responsibilities のところには「内容を教える」という責任を書きます．

2. モデル化のプロセス

CRC カードを用いたモデル化のプロセスは，「知っているところから知らないところへ」と言われています．簡単に手順をご紹介します．

要求の理解
クラス抽出
　適切なクラス名
シナリオ設定
　What if
　What happens when
セッション（ウォークスルー）
　責任
　協力関係
　クラス

図3 CRC カードによる開発プロセス

1)要求の理解

まず，前提となる要求を理解することが必要です．これは CRC 手法には直接関わりのない，一般的な準備段階です．

2)クラス抽出

次に，その要求の中からクラスの候補になるものを探し出します．例えば，要求として書かれたものの中から，名詞を中心としてクラスになりそうなものを挙げていきます．そして挙げた候補に対して，そのクラスが「システムの中でどういう位置付けになるか」という特徴を適切に表す名前をつけ，カード化します．

3) シナリオ設定

次に，要求の中からシステムとしてどのようなことが起こるかを規定していきます．これは，What if とか，What happens when の形で行ないます．つまり，あるデータが入力された時には何が起こるかというようなシナリオを設定していきます．

4) セッション

そして，設定されたシナリオについてシミュレーションを行います．これが CRC カード手法の中心になるもので，何人かで集まってテーブルを囲んだり，畳の部屋で車座になって行ないます．

ここではクラス候補として書かれたカードの責任や協力関係を確認していきます．あるいはこれまで出てきていない新しいクラスや関係を見つけていきます．

これは通常，「セッション」とか「ウォークスルー」などと呼ばれます．

始めに，要求のところで見つけ出したクラス候補に対応するカードを参加者に割り振り，分担を決めます．そして設定してあるシナリオについて，誰かからシナリオの実行が始まります．

まず，シナリオの出発点の分担者が「私はこういうことをしなければいけない」とカー

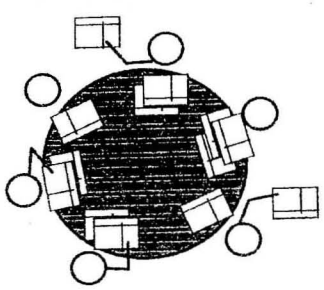

図4 セッション

ドを持ち上げて言います．そして，必要があれば「ついては，これこれをやってほしい」と協力者を求めます．

協力者クラスが既に存在し，その担当者が納得すれば，例えば「わかりました」とカードを上げて答え，相手に提供するサービスを責任としてカードに書きます．場合によって，新しくクラス（カード）を作ったり，協力者なしで行なうべきかを検討したりもします．

この「何らかの責任を実行中はカードを上げる」という行為はなかなか良いアイデアだと思います．これは，伏せてあるカードはいわゆるクラスで，上げた瞬間にそれがインスタンスになると捉えられます．そしてこれにより，単にカードを使う手法ということではなく，むしろカードを挙げている人がオブジェクトの役割を演じることになるのです．

3. 特徴の整理

この手法の特徴と言われるものを整理してみましょう．

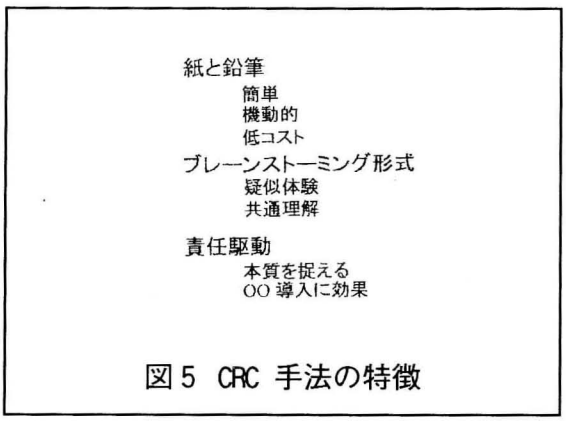

図5 CRC 手法の特徴

1) 紙と鉛筆

特別な道具立てが不要です．つまり，とても簡単で，どこでもでき，またコンピュータも何も必要ないために大変低コストで行なえます．

2) ブレーンストーミング形式

モデル化は，いわばブレーンストーミング形式で行なわれます．参加者はシステムの一部を演じることにより，システムがどう動くかということを疑似体験できますので，理解が大変深まります．また，参加者全体でシステムを演じるので，共通の理解も得られます．

3) 責任駆動

カードには責任と協力者が記載され，最初はデータ構造などは書かれません．このため，細部に振り回されることなく，本質を捉えることができると考えます．つまり，オブジェクト指向的な考え方のモデルに近いものが自然な形でできることになると思います．

CRC カードは，もともとオブジェクト指向的な考え方をどうやって身につけさせるかを目的に考え出されたものですので，そういう意味でもオブジェクト指向の導入に非常に効果があると思います．

4. カードの拡張

手法自体が簡単なので，カードに拡張を加えるのも簡単にできます．いろいろとカードのシンタックスに拡張を加えたものが存在します．

これは，後でご紹介する「Using CRC Cards」に載っている例です．Sub Class と Super

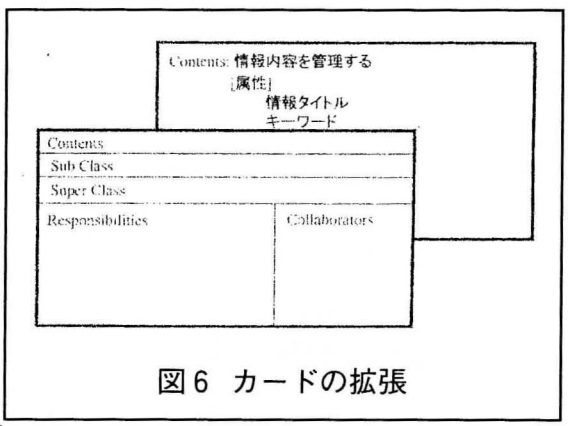

図6 カードの拡張

Class が拡張されています．また，これまでの説明ではカードの表しか使いませんでしたが，裏も使います．裏にはクラスの名前の他に，そのクラスがどういうものなのかということをもう少し詳しく書き出したり，重要なデータ構造などについて書きます．

5. 参考文献

参考文献をわかる範囲でご紹介しておきます．

「Using CRC Cards」 には，カードの説明から始まって，セッションが実際にどのように行なわれているかなどが演劇の台本のような形で書かれています．

「An Introduction to Object-Oriented Programming」では，オブジェクト指向のプログラミングについて CRC カードを使って説明が述べられています．また，責任駆動についてもコンパクトにまとまった説明がついています．

「Designing Object-Oriented Software」では，CRC カードも使いながらどのような設計をしていくかが書かれています．

Kent Beck 等のものは，最初に CRC カードについて発表された論文だと思っています．

非常に簡単に御紹介しました．これらの文献を当たっていただきますと，より一層，理解が深まると思います．

表 1. 参考文献

Kent Beck, Ward Cunningham
“A Laboratory For Teaching Object-Oriented Thinking”
Proceeding of the 1989 OOPSLA
Sigplan Notices, vol. 24, num. 10, Oct. 1989
http://c2.com/doc/oopsla89/paper.html

Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener
“Designing Object-Oriented Software”
Prentice Hall, Englewood Cliffs, NJ, 1990

Timothy A. Budd
“An Introduction to Object-Oriented Programming”
Addison-Wesley, 1991
邦訳：「オブジェクト指向プログラミング入門」，トッパン

Nancy M. Wilkinson
“Using CRC Cards”
An Informal Approach to Object-Oriented Development
SIGS Books, 1995

SEAMAIL vol. 7, No. 4-5-6-7（合併号）, 1993

SEAMAIL vol. 9, No. 2-3（合併号）, １９９４

David Bellin, Susan Suchman Simone
“The CRC Card Book”
Addison Wesley, 1997

（記録：田中慎一郎）

中規模開発での CRC 使用事例

(株) アドバンテスト
武田　泰英

1. はじめに

中規模開発で CRC を使ってみてどうだったかをお話しします．中規模開発とは，極端に小さなシステムではないという程度の意味です．導入してまだ間もなく，定量的な分析などはできていませんので，どちらかというと定性的な，主観的な報告になります．

さて，CRC の利点としては，先に「CRC カード手法の概要」でお話ししたように，簡単，機動的，低コスト，本質をとらえる，共通の理解が得られるなど，いろいろあります．しかし，CRC カードを使えば何でもうまくいくというわけではありません．ここでのお話は，こういう利点に誘われてついフラフラと始めてしまった結果がどうなったかというものになります．

1.1 対象アプリケーション

既存のアセンブラで書かれたシステムソフトを C 言語で書き換えるプロジェクトに適用中です．

元々はアセンブラで 100K ステップ程度のものですが，15 年以上前に書かれたものでもあり，機能的にも貧弱なところがあります．そこで，その機能を包含して機能アップを図ったものに置き換えることになりました．具体的には，LSI をテストするハードウェアを制御するものです．論理的なまとまりとしてとらえると数百のコマンド，単純にコマンド・レジスタの数を勘定すると数千のオーダーになるという程度の規模のソフトウェアです．

1.2 導入準備

1) CRC カード手法の学習

まず，「CRC カード手法の概要」でお話したような，簡単な CRC カードの説明を行ないました．ここではカードシンタックスとセッションの進め方，特にカードを上げ下げすることを強調しました．

2)オブジェクト指向の学習

それから，オブジェクト指向の勉強会を行ないました．これは Designing Object-Oriented Software の 1 章から 5 章までを輪講する形で行ないました．

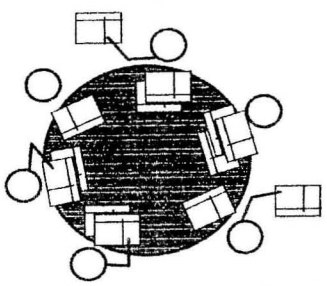

図 1　導入準備

3)パイロットセッション

そして，パイロット的に簡単なセッションを実施しました．これは，今後の開発時にあると便利なドキュメントツールや辞書検索ツールについて行なったものです．数百行程度の簡単なものですが，これを実際に作成しました．

1.3 私の立場

このような背景で実際の開発に CRC カード手法を応用しましたが，結果としては「失敗」になると思います．しかし，私の立場は「だから CRC は駄目だ」というものではありません．私の適用経験に基づいて，CRC カード手法を開発に利用する場合にはどのような問題が起こりがちか，どんな注意や配慮や体制が必要かなどについて，個人的見解をお話ししていきます．

2. 観察されたこと

ではセッションでどのようなことが起こったかをご紹介します．

2.1 停滞／混乱

セッションは時々，混乱したり，停滞したりします．それ自体は重要なことも多いのですが，無駄な時間となる場合もあります．気がついたことをまとめてみました．

1) 複数のモデル候補

いくつかのモデルが考えられる場合があり，どちらが良いのかを全員が考え込んでしまうことがありました．

これには二つのケースがあり，一つは「どちらでもシステムの挙動としては同じ場合」，もう一つは「それぞれシステムの挙動が違う場合」です．

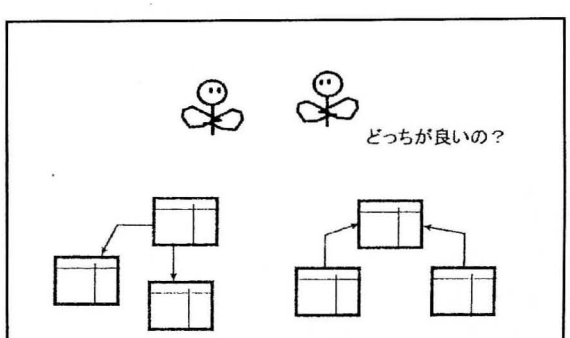

図2 停滞／混乱 1

後者の「システムの挙動が違う場合」というのは，要求がきちんと理解されていないか，必要な調査が済んでいないことを意味します．この場合，要求者でなければ決められないことも多く，そのために情報不足のまま，どちらにしようかといつまでも無駄に悩んでしまうことがありました．

前者の「挙動としては一緒」のケースですが，これに関しては，どういう指針で決定するかをあらかじめ決めておく必要があったと考えています．例えば，システムの実行時間を重視するのかとか，保守性を重視するのかとか，そういう具体的な方針です．

2) シナリオにおける条件の膨張

シナリオの条件の洗い出しに終始してしまうことがありました．

あるシナリオのシミュレーション中に，「こんな場合はどうするの？」とか「こんなケースもあるんじゃない？」などという発言ができることがあります．このような場合，時として条件が膨らみ過ぎ，今どのようなシナリオを実行しているのかがわからなくなり，まるで条件の洗い出しを行なっているかのようになることがあります．

これはシナリオの粒度が大き過ぎると起こりやすいと思います．シナリオが，例えば「LSIシステムを制御する時に何が起こるか」というような曖昧なものであるにもかかわらず，無理矢理に実行するとこのようなことになってしまいます．

3) 想定条件の不一致

想定する条件が違ったまま意見が対立していて，無意味に時間がかかることがありました．

いくつかの条件でそのシナリオが実行される場合があります．注意していないと，それぞれの参加者がある固有の条件で実行される別々の場面のみを想定していて，その条件で一番良いと考えることを主張して譲らないというようなことが起こります．

4) データかオブジェクトか

ある情報に関して，それが単なるデータなのか機動性のあるオブジェクトなのかという捉え方で混乱することがありました．

例えば，「雑誌」という名前がシナリオに出てきたとします．これがクラスとしてカードになってしまっていればあまり問題はないのですが，協力者に渡すパラメータのような部分に出てくると，各人で捉え方が違ってしまっていてシナリオの理解が噛み合わないことがあります．

5)想定する議論レベルの相違

考えていることの深さが違い，話が噛み合わないことがありました．

一方はスケジューラやプロセスという大きなレベルで進めているのに，他方はバッファやスタックという実装寄りの話を考えている場合です．

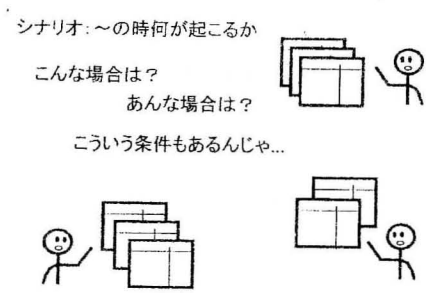

図 3 停滞／混乱 2

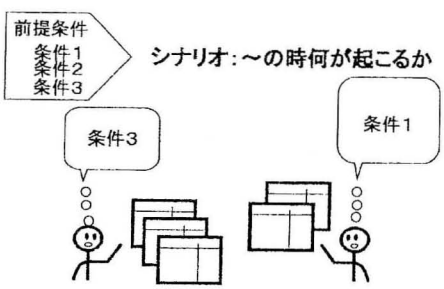

図 4 停滞／混乱 3

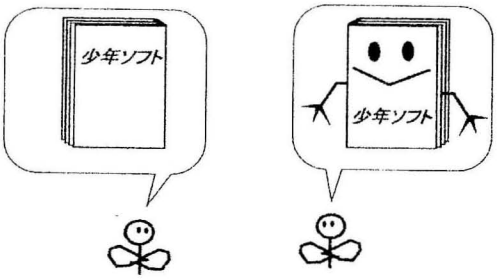

図 5 停滞／混乱 4

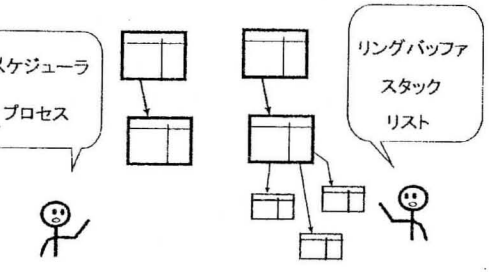

図 6 停滞／混乱 5

これは，対象としているアプリケーションの実行されるプラットフォームや OS などが決まっていて，それらからのバイアスがかかっている場合があります．これはかかる人とかからない人がいます．

6) 一部分にのみ集中

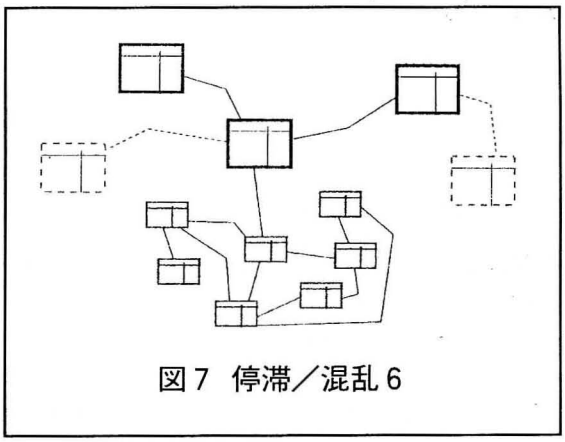

図 7 停滞／混乱 6

大局が見えなくなって，一部ばかりを詰めていき，他の部分は荒いまま放置されることがありました．

これは，要求の理解がきちんとなされていないことが原因で，良くわかっていないところを避けて，わかっているところばかりに力を入れてしまうということのようです．

良くわかっていない，あるいはきちんと決まっていない部分があることはシナリオを用意する段階でもある程度わかっていますが，セッションを始めてからでないと気がつかないこともしばしばあります．そしてそういう部分が放置されてしまうことが起こりがちです．

曖昧な部分が見えてくるのは CRC カードを使うことのメリットです．ですから，その時点で要求者に確認をするなりしていけば良いのですが，そのコントロールがうまくできないと後で却って問題になります．モデル化が進むという側面と曖昧な部分が見えてくるという側面は，それぞれの対処方法が異なります．それぞれの方針をきちんと決めておかないと，機能モデルを作っているはずが実装の設計に踏み込んでいたということも起こり得ます．

7) 時間経過による忘却

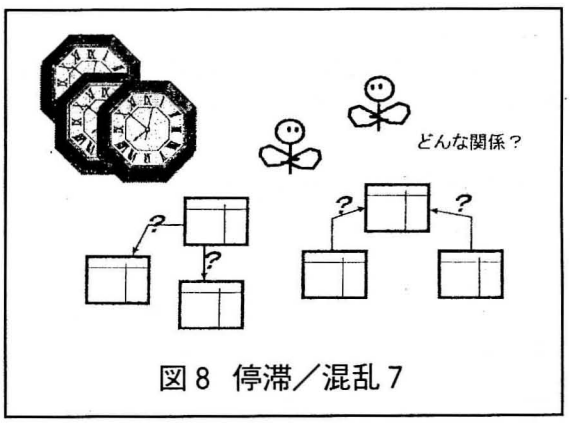

図 8 停滞／混乱 7

セッション中は良くわかっているのに，時間が経つとわからなくなってしまうことがありました．

小さなアプリケーションですとカードを見比べるだけでもそれなりに思い出しますが，ある程度の大きさになると扱う関係も多くなるため，はっきり思い出すのが非常に難しくなります．

やはり，カードだけではドキュメントとして不足だと感じます．

8) 勝手な個別シミュレーション

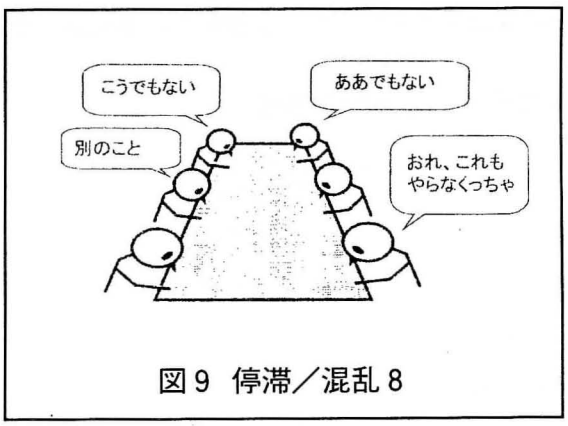

図 9 停滞／混乱 8

セッション中，全員が下を向いて自分のことだけを考え込んでしまうことがありました．

責任実行中の人はカードを上げると決めているのですが，いつの間にか誰もやらなくなり，それぞれの頭の中だけで勝手にシミュレーションを進めてしまうのです．結果的には共通の理解が得られませんし，実感を伴うと

いう部分が希薄になるので，会議場を出た瞬間に忘れてしまう原因にもなっていたと思います．

2.2 心理的側面

セッション中の参加メンバはいろいろな反応を示します．注意した方が良いことをまとめてみました．

1) モデルへの参加者の性格の影響

参加者の性格が不適切な形でモデルに反映されることがありました．参加者はそれぞれ違うということに注意しなければなりません．

特に導入時点では，やたらと照れる者や，話すのが苦手で指名されるとひどく困った顔をする者，めんどくさがり屋で頼まれるとすぐ嫌という者などがいます．慣れてくるとそういう側面は消えてくるのですが，このような参加者の性格は，必ずモデルに反映されます．

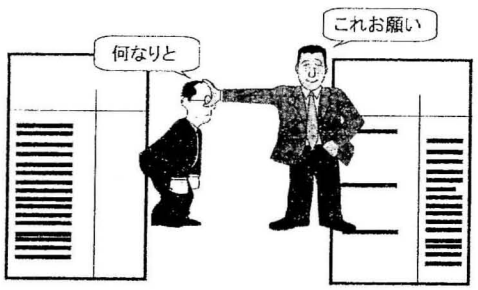

図１０ 参加者の性格とモデル１

例えば，頼まれると嫌といえない者がいる一方，何でも他人に頼む者がいます．こういうメンバでモデルを作成すると，やたらと責任が多いクラスとやたらと協力者が多いクラスができてしまいます．

これは性格的な問題の他，組織上の上下関係などに影響される場合もあります．このような場合には，モデレータ役の人が調整する必要があるかもしれません．

2) クラス責務の混乱と乱用

書き直しが多くなってカードが汚くなってきた場合に，既に出ているもっともらしい名前の責任に何でも押しつける傾向が見られました．

適当なところで新しいカードに書き写すなどの整理が不可欠ですが，これもそれを指摘する役割が必要かもしれません．これを放置しておくと，先ほどの時間が経つと忘れるという傾向を助長することになります．

ポストイットなどを用いて書き直しが楽なように運用面で工夫する考え方もありますが，途中の変更履歴を残すという意味との兼ね合いもあるので難しいところです．

3)ステップの勝手な省略

一人で先走って考えてしまい，途中のステップを飛ばしてしまう者が出ました．その結果，頼まれた協力者の方が何をしないといけないのかついていけなくなることがあります．

参加者はそれぞれ，頭の回転が早い者もいれば，じっくりと考えたいタイプもいます．

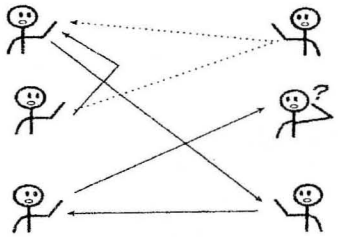
図１１ 参加者の性格とモデル２

また，カードの上げ下げが面倒ではしょりたいと考える者もあるようです．
このようなはしょる行為が増えてくると，次第に全員が下を向いて自分のことだけ考える状態になってしまいます．

4) 経験の違いによる拘り

オブジェクト指向的な考え方に馴染めずに，妙な拘りを持つ者が出ました．例えば，本のクラスを作ろうとした時に，「本が勝手にページをめくったり，内容を印刷したりするはずがない」というような拘りを持つものが出たりします．このような時は，「そうだね．では本を操作する人を作りましょう」などとうまく折れてやることも重要だと思います．
CRC カードはオブジェクト指向的な考え方を導入するのに向いているとはいえ，それでもそれについていけない者も出てきます．CRC セッションをやって行くと，物の考え方自体は手続き的な考え方とは違う，かなり自然なものになってきます．しかし自然なと言っても，あくまでその人がそれまでに経験してきたことに立脚したものなので，そこから抜け出すのは非常に難しいと思います．このため，ここで意見の対立が起こることもあります．

2.3 手法の拡張

セッションを進めていくうちに，カードの記入方法や理解のためのグラフ化など，各自がいろいろと工夫を始めます．これらの工夫はとても有効ですが，全体で統一が取れていないと混乱のもとにもなります．カードや理解用のグラフなどの使い方はある程度ルール化すべきだと感じています．

1) インデントによる階層化

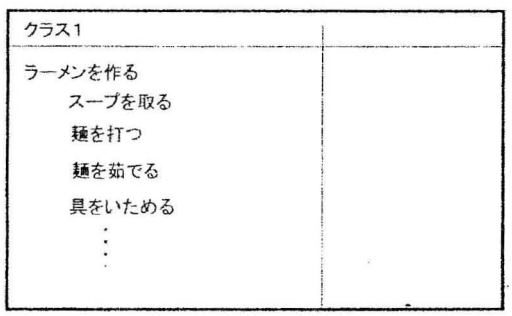

図１２ カードの書き方の例１

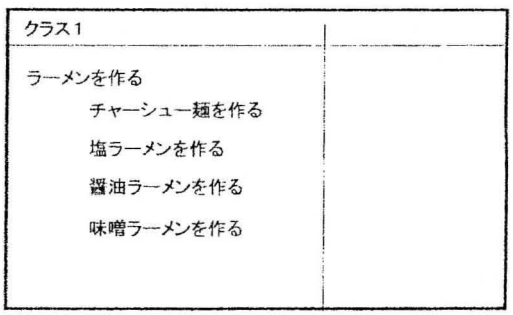

図１３ カードの書き方の例２

カードの書き方のルールをそれぞれが勝手に工夫し出しました．
例えば，ラーメンを作る手順を考える時に，その中のサブタスクに当たるようなものをインデントして書く者が出ました．これは「ラーメンを作る」という責任のアルゴリズムを思い出すためのメモ書きに使っているのです（図 12）．
一方，同じようにインデントをつけるのですが，アルゴリズムではなく，「ラーメンを作る」という責任の細分類を書き並べる者も出ました（図 13）．
見た目には同じようなインデントの仕方なのですが，意味的には全く異なります．どちらが良いということではありませんが，方針を決めておかないと後でカードを見た時に混乱してしまいます．

2)必要なパラメータの記入

責任を果たす際に必要なパラメータを書きたくなる場合があります．これもそれぞれが勝手な書き方を始めました．

パラメータの数が多いと，横にはみ出したり，他のことが書けなくなったりといろいろ不都合が出ます．これに関しては，現在は制約を設けていないのですが，カードの中にはあまり細かいことは書かない方が良いと思っています．

できれば，ある程度書き方のルールを事前に決めておいた方が良いように感じます．

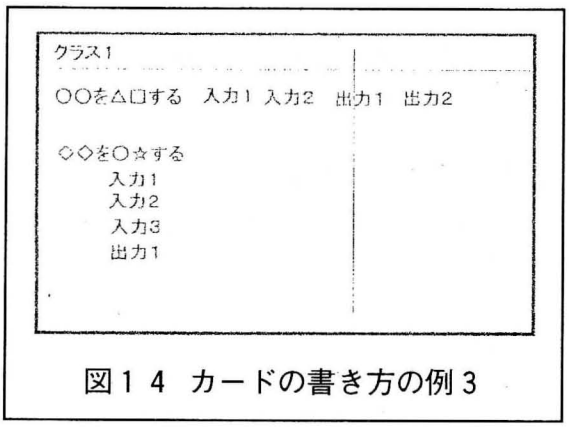

図１４ カードの書き方の例３

3) 理解用の補助グラフの利用

カードだけでは記録性／表現性が低いことがわかりました．そこで，図 15 のようなグラフを併用することになりました．

丸がクラスで矢印がメッセージの方向です．矢印の上には，そのメッセージにより起動される責任を書きました．さらには起動される順番まで書き始めるようになりました．

このような方法は，時間が経つと忘れるという傾向を抑制するにはとても役立つのですが，逆にカードを使わずにこればかりで済ませようとする傾向も出てしまいます．

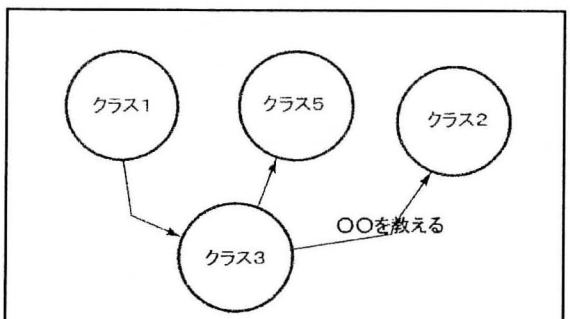

図１５ 呼び出し関係グラフ

この現象は，全員が下を向いて勝手に考えたり，手順をはしょる現象と同じで，カードを使って疑似体験をする意味が薄れてしまいます．この面からも，カードを使うことを仕向ける役割を持つ人が必要だと思います．

このグラフについてもう少しコメントしておきますと，自分達で決めた記法なのに読み違いをすることがあり，混乱がありました．

矢印はメッセージの方向で，矢印の上には起動される責任を書いています．例でいうと，「教える」のはクラス２なのですが，矢印の方向で混乱して，クラス３が教えるのだと誤解するものが出ました．これがきっかけで，矢印の上には「教えてもらう」と書いた方が良いとかそれはおかしいとか，議論が混乱することがありました．

参加者は次第に慣れてくるのですが，これを他のメンバに伝えるために使おうとした場合には，やはり同じ問題が起こると思います．また，丸と矢印を用いたのがいけないのですが，この図が次第にデータフローになってしまう傾向もありました．

しかし，やはり何か全体を把握するための補助ツールが必要だと感じています．

3. CRC の利点と罠

最後に CRC の利点とそれに付随する罠についてまとめてみます.

1)簡単に行なえる

CRC の利点の一つとして,「簡単に行なえる」というものがあります. これは, カードやセッションを使った手法が「とっつきやすい」というだけのことであり, 実際に扱う問題が簡単になるわけではありません.

当然ですが, 元々の問題に含まれる「要求される複雑さ」とか「取り扱うべき条件」が簡単になるわけではありません. 他の手法を使う時に気をつけなければならないことは, すべて同じように気をつけなければなりません. そこを誤解しないことも非常に重要です.

2)機動性が高い

紙と鉛筆があればどこでもできるので機動性が高いと言われています. しかし, 会社などで行なう場合には「どこでやるか」,「誰を集めるか」などの問題があります. このため, 実際にはそれほど機動的ではありません. 特に,「有識者」のスケジュール調整が大変です.

これは, セッションが混乱／停滞するのは, 仕様が曖昧であったり, 理解が不足していることが多いためです. 明確に回答できるメンバが参加することはセッションの進行を非常に助けますが, 逆に言うと,「有識者」がいないとセッションが無用に混乱しかねません.

3)低コスト

特別なものが必要ないので低コストだと言われています. これはその通りなのですが, やはり停滞／混乱は発生し, その際に物理的なロスが出ます. 物理的ロスといっても, 実際には時間を含めてもさほど大きいものではありません. しかし, うまくいかなかったという「心理的圧迫」が非常に大きなものになる可能性があります.

これは特に導入時に大きく, 初期の段階で「やはり従来方式でやれば良かった」という印象を持たれてしまうと, その後の開発におけるグループ全体の志気にも関わることになります.

4)責任駆動設計

CRC カードを使って責任駆動設計をするために, 本質をとらえたモデルが作りやすいと言われています. しかし, どうしてもメンバそれぞれの拘りが出てきます.

先ほども申しましたように, 考える時のベースはそれぞれの個人の経験や, 個人的常識になります. それがそれぞれ違ったり, また変な常識を持っている者がいたりしますので, 必ずしも常に本質をとらえたモデル化が容易にできるわけではないと感じます.

また, いろいろなことを考えているうちに想像が膨らみ過ぎて脇道にそれ, これもやろう, あれもやってあげようと, 本質から外れた部分で規模が大きくなることもありそうです.

5) セッションを通じた共通の理解

セッションの形で分析／設計を行うために共通の理解が得られると言われます. 確かに, セッションを終える時には全員が良く理解していると思っています. しかし, 実際にはそれぞれが別々の理解をしていることがしばしばあります.

例えば，同じ単語の理解でもメンバそれぞれで違っていたり，セッションの途中で意味が変わってきた単語を前の意味で使っていたりします．ですから，共通の理解を得やすいことは間違いないのですが，本当に同じ理解をしているかどうかを確認する手段が必要だと思います．

方法としては，あまり良いものは思いつきません．やはり面倒でも辞書をきちんと作るとか，セッションの進行をモニタして誤解があると思われた部分に確認を入れる役目の人を用意するとかくらいです．

4. まとめ

最後にこれまでの経験をまとめてみます．

4.1 要求の重視

CRC に限りませんが，やはり要求が大事だと考えます．ここでは要求とシナリオを同じものとして扱っています．

1) シナリオの明確化が重要

シナリオを設定する時に，入出力条件，シナリオの実行前／実行後の状態変化についてを明確にしておかないといけません．そうでないと，シナリオを実行した結果がどこへ飛んでいくかがわかりません．

2) シナリオの階層化が重要

シナリオの階層化は特に重要だと考えます．このため，階層化の仕掛けを用意すべきだと考えます．シナリオがいろいろな条件／状態を包含した大きなものですと，どうしても進まなくなります．そのような場合は，セッション中に条件毎のシナリオに分解していかないといけません．この時に，元々の要求，それを細分化した要求という形で，階層化して一緒に管理しておく必要があると思います．

4.2 段取りの重要性

1) CRC をどこでどのように使うか

CRC はいろいろな局面で使用できますが，どこでどのように使うかは最初に方針を立てておくべきでしょう．例えば，要求分析の段階で使うとか，要求を見い出すために使うとか，お互いに議論をする道具にするとかです．

2) どういうモデルを作成するか

CRC をどこで使うかにも関わりますが，開発の中でどのようなモデルを作成するかも，最初に決めておく必要があります．そして作成するモデルの中で CRC がどのような位置付けになるかも明確にしておかなければなりません．

やはり CRC だけでモデルを記述することは難しいと考えますので，それをどのようにするかもあらかじめ決めておかないといけません．

3) セッションが停滞したらどうするか

セッションの混乱はかなりの場合，要求の理解が曖昧であることが原因です．このため，その場にそれを明確に回答できる人がいない限り，問題を整理してセッションを中止する勇気を持つことが重要です．また，そういう合意をとっておくことも必要でしょう．

同様に，意見の対立の場合，うまく譲り合いをすることも肝要です．どうやって実現するかは難しいことですが，いつまでもどちらが正しいかを議論するよりも，「とり合えずＡさんの意見でいきましょう」とする方が，いろいろな意味で楽です．もしもＡさんの解が間違っていたならば，必ずどこかでそれが明らかになりますので，まずは譲り合って進行を速めることを全員が念頭におくようにすべきだと思います．

5. 質疑応答

1) セッションは何名ぐらいでやるのが適当と考えますか．

３人だと少ないが，６人だとまとめるのが大変だと感じます．

2) 実際には何名でセッションをやっていますか．

当初５名．現在は４名です．

これはそれだけしかセッションに参加できるメンバがないということで，６人だと大変だからという理由ではありません．

3) CRC カードでやる以上，開発に関わるメンバはできる限りセッションに参加すべきだという考え方と，分析／設計は一部のメンバでやるという考え方があると思いますが，どのように考えますか．

個人的には，セッションへの向き不向きもあるので，分析チームと開発チームは分けた方が良いと思います．しかし，分析チームのやったこと／考えたことをどのように開発チームに伝えるかという問題が残ります．

4) CRC の罠が実装の段階に大きく影響をすることはありますか．

まだ実装の段階に入っていないので，これからまだどんな膿が出てくるかちょっと心配しています．

5) カードの裏は使いましたか．

使いました．

最初は簡単に属性をメモする程度でしたが，段々書きたいことが増えています．先ほど例に出た入出力パラメータなども書くようになり始めましたし，大きな責任に対するアルゴリズム的なものもメモしています．

（記録：田中慎一郎）

オブジェクト指向をどう現場に導入すべきか

ヒラタソフトウェアテクノロジー（株）
古閑　幸一

1. はじめに

まず，私の背景からお話いたします．

1.1 開発に対する個人的欲求

私の所属している組織は，平田機工という製造業の会社にソフトウェアの技術が欲しいという理由で作られた子会社です．私は２期生で，この時点ではまだあまり会社にソフトウェアの素地はなかったと思います．そのためか，入社した当初から個人的に「整然としたプログラム構造が作りたい」という欲求がありました．

入社して２年めくらいまでは，いわゆる機能分割の形でプログラムを作っていました．しかし，どうしても同じような機能があちこちに分散したり，無駄に作られたりすることがあり，もう少し整然としたプログラム構造が作れないものかと考えていました．

また，部品化できるプログラム構造を把握したいとも考えていました．それまでも「部品化」は良く言われていたのですが，なかなかうまくできるものではありませんでした．

それから，開発フェーズ間のギャップを小さくするための開発方法についても考えていました．これは，要求分析のレベルからコーディング／インプリメンテーションでのギャップが大き過ぎるように感じていたためです．

1.2 オブジェクト指向への取り組み

1) 責任／期待の関係

10 年くらい前に初めてオブジェクト指向に取り組みました．この時の方法は「責任／期待」の関係でシステムを構築するもので，情報隠蔽の考え方はまだあまりできていませんでした．

リーダー格のメンバが Booch の論文を読み，その解釈から「クラスには『責任』があり，その協同でシステムは出来上がる」という形で指導されました．しかし，この時にはクラスの責任者が消極的で，クラスとクラスの間にすき間ができ，システム全体としてうまく成り立たず，失敗でした．

2) OMT

その後，自分でも OMT を勉強しました．オブジェクトモデルの部分だけとり入れて，システムの分析／構築を行ないました．しかし，どうもシステムが組み難いという印象を受けました．

3) CRC カード

その後，1992 年の SEA 若手の会で CRC カードを知りました．この考え方は私がはじめにやっていた「責任／期待」の考え方に近く，それをより整理して体系化したものだとの印象を受け，今後この考え方をプロジェクトに適用してみようと思いました．

1.3 CRC に関する当初の印象

最初に CRC 手法を知った時には次のような印象を受けました．

1) シナリオベースの考え方は大変有効である．

「シナリオ」の考え方を採り入れることは，いろいろな局面を洗い出せるので，とても有効だろうと考えました．また，シナリオに沿ってシステムを構築すると目的に沿った構築ができるので，そういう意味でも正確なシステムができそうだと考えました．

2) メンバで分析をやるので認識が高まる．

最初に紹介を受けた時は，この点で非常に有効な方法だと考えました．実際にどうなるかは後で説明します．

3) 比較的とっつきやすい．

我々の作成するシステムは FA なので，ロボット，倉庫など，オブジェクトが出やすいドメインだと考えています．その意味でも比較的とっつきやすいと思いました．

4) 変更に柔軟に対応できる．

基本的に期待と提供の関係で結び付いていますので，期待が変わることはその線を引き直すこととととらえられます．このため，柔軟に対応できるだろうと考えました．

一方，疑問点や心配点として以下のことがありました．

a) 責任駆動が機能駆動にならないか．

責任駆動と機能駆動は全く違うものだと考えています．このため，うまく適用できずに，SA/SD と変わらなくなってしまうのではないかと心配しました．

b) メンバ全員で取り組む時間がとれるのか．

開発対象がパッケージではなく受託なので，開発期間を考えた場合，それだけの時間がとれるのかを心配しました．通常はどうしても並行して多くの作業が実施されるため，システム分析段階でそれだけの時間が全員でとれるのかが心配でした．

c) 分析を設計にどうむすびつけるのか．

これは現在でも抱えている疑問です．
しかし，それまでの方法よりは比較的スムーズに分析結果を設計に生かすことができ，また設計結果をコード化しやすいだろうと考えました．ただ，具体的にどうすれば良いかは思いついていませんでした．

d) 文書化が難しいのではないか.

ある程度フォーマライズされた記述法がないと忘れてしまうことが予想され，後の工程が大変になりそうな気がしていました.

e) 実装段階での制約をどの段階で考慮するのか.

例えば，ポーリング形式が良いのか／イベント駆動形式が良いのかや，マシンのパフォーマンスを考えた時の分散のさせ方はどうするのかなどを，いつどのように考えるべきか疑問でした.

f) タスク分割はどうするか.

特にタスク分割は，比較的前の工程で行なうべきなのか，できるだけ後の工程に残すべきなのか，疑問に感じていました.

2. 1994 年の Like CRC 事例

1992 年に若手の会で初めて CRC を知り，1994 年に実際に開発に適用してみました．その時の経験をお話しします.

2.1 対象システム

1) 対象ドメインは FA システム.

ロボットが７台，倉庫が２基あり，CAD/CAM の要素がありました．簡単にいうと，お客様のコンピュータから CAD データをもらい，ロボット用の切断データに変換し，切断されたものをコンベアで出荷用の倉庫に入れるシステムです.

2) 開発規模は見積りベースで 1000 人日ほど.

実際には 2000 人日くらいかかってしまいました．スケジュール的には，設計２カ月，実装３カ月，保守１年程度のものです.

3) メンバは分析時３名.

オブジェクト指向経験者が私一人，入社２年めくらいのものが２名でした.

2.2 導入時の状況

1) 機能仕様は半決まりの状態.

完全に機能仕様が決まっていることはほとんどあり得ませんから，このケースに限らず，ほとんどの場合このような状態での着手となると考えます.

2) シナリオがない状態.

これは無謀だったのですが，シナリオはやりながら出てくるだろうと考え，何もない状態から始めました.

3) クラスも全く洗い出していない状態.

まずはクラスも洗い出すことなしに始めました.

4) 事前教育は「若手の会資料」を渡した程度.

武田さんのお話の中でも段取りを十分にとらないといけないというものがありましたが，少々段取りが悪かったようです．私が理解できたので，簡単な資料だけで十分と思ったのですが，そもそも私自身も良く理解できていなかったのかもしれません.

2.3 観察された結果

最初のトライとしては少々プロジェクトが大き過ぎたように思います．安易に始めたので，少しダメージが大きかったようです.

1) セッションの記録不足による混乱

まず，どんなシナリオだったのか，その詳細がわからなくなってしまいました.

やっている時はスムーズに進むのですが，後で思い出せなくなるのです．あるメンバの言葉に「やっている時は楽だけど，後が大変」というものがありますが，これがすべてを物語っているかもしれません.

また，シナリオで通したルートがわからなくなりました．このために実装で見直しが発生してしまい，大変な無駄となってしまいました．これらは，記録不足というか，記録が全くできていなかったということだと反省しています.

シナリオを文書化する危険性もあるとは思いますが，ベースとなる基本的なもの，とっかかりになるものは文書化しておくべきだったと考えます.

そしてルートに関していえば，私たちも武田さんと同じように，呼びだし関係図に順番を書き入れるということをやりました.ホワイトボードなどのスペースは限られていますので,段々書き込めなくなることがあり，この辺りの工夫も必要だと考えています.

2) 低品質クラスの導出

導出されたクラスですが，Meyer の５つの基準[1]に照らした時に全く満たされていませんでした.

まず，クラス間の結びつきが強過ぎ，あるオブジェクトを部品として取り上げようとすると，関連するオブジェクトが芋蔓式についてきてしまう状況でした．また，クラスの管理するデータの意味が弱いという結果もありました．これらは責任ベース／機能ベースでの導出方法に失敗があったと思います.

---

[1] Meyer の 5 つの基準とは，モジュールの分解しやすさ，モジュールの組み合わせしやすさ，モジュールのわかりやすさ，モジュールの連続性，モジュールの保護性を指す．（参考文献：「オブジェクト指向入門」BERTRAND MEYER 著，二木厚吉 監訳，酒匂寛・酒匂順子 共訳．アスキー出版局）

クラスを最初に見つけておいてそれをベースにするアプローチをとるべきなのですが，出てきたクラスをただ並べて責任でのみ結び付けてしまったために，後で見直した時に「なぜそのクラスがそんなデータを管理するのかわからない」ということがたくさん起こりました．

それから，クラスのグループ化が難しいものとなってしまいました．自分たちは C 言語で開発していて，ライブラリなどを作成するのですが，この際にグループ化する根拠がほとんどなく，クラス間の関係も複雑だったので，グループ化が非常に難しいものとなりました．これは，開発段階で担当者に分担するのですが，この時にグループ化ができなかったということでもあります．

3) 形式化されたカード記述方法の欠如による混乱

まず，「少し違うから一緒にできない」似たような責任がたくさんできてしまいました．さらにこの責任を簡略化して書いた結果，後で見た時にその詳細な違いがわからないことにもなりました．

また，実装段階で関数仕様に落せないという現象も起こりました．パラメータの話もセッション中には出ているのですが，きちんとした記述方法が決まっていないので，記録に残っておらず，実装段階で関数に落せなくなってしまったのです．

これらも「やっている時はスムーズだが，後が大変」となります．

4) 能動的な意見が出ない

これは，オブジェクト指向の理解が浅いためか，カードベースで進めたためか，それともシステムに興味がないのかはわからないのですが，あまり自主的な意見が出なかったのが気になりました．

一つは準備不足のためだと思いますが，もう少し積極的な姿勢が出るような教育や動機づけが必要だと感じました．

2.4 1994 年事例のまとめ

1994 年の事例から注意しなければいけないと感じたことをまとめてみました．

1) 事前準備について．

まず，要求定義はある程度まで済ませておく必要があると思います．また，オブジェクト指向の基礎知識や CRC の基礎知識に関する事前教育はきちんとしないといけないと感じました．始めれば何とかなるだろうとの考え方は非常に危険だと思います．

2) 次フェーズへつなげるために．

後に続く設計フェーズなどにうまくつなげるために，フォーマルな責任の記述方法の設定が必要だと感じています．

3) シナリオを大切に．

シナリオのきっかけになるネタは事前に十分準備しておくことが重要だと感じています．また，シナリオの演出家とまとめ役に専念する人を作ることは非常に役に立つと思います．

この人は，森の中の木にならずに森を見るわけですが，実際のプロジェクトでは，例えば3人の内の1人がその役になったのでは，残り二人でセッションをやることになってしまうので，少々難しいことでもあります．

4) 静的分析によるオブジェクトの抽出について．

静的分析によるオブジェクトの抽出は大変有効だと感じています．これは OMT のオブジェクトモデルのことを指しているのですが，システムの骨格が理解でき，責任導出では見えないクラスや括りが見えてくると思います．

これをやっても特に CRC のシナリオが変わってくることはありません．別の見方を提供するだけです．いろいろな理解を持てることは役に立つことだと思っています．

3. 1996 年の Like CRC 事例

1995 年はここで作成したシステムのメインテナンスをやっておりましたので，次の事例は年が明けた 1996 年になります．

3.1 対象システム

1) 対象ドメインはこれも FA システム．

ロボットから加工メニューを送れという要求受け，そのレシピを送ってやります．また，装置運転状態を監視するという一般的な監視制御をします．

2) 開発規模は，400 人日程度．

スケジュール的には，設計，実装，保守，それぞれ1カ月程度のものです．

3) メンバは，分析時4名．

オブジェクト指向経験者は私一人で，未経験者が3人です．これは 1994 年とは異なるメンバです．

3.2 事前準備

1) 機能仕様はある程度固まっていた．

このケースは，パッケージ化して売りたい製品の開発でした．パッケージなのでお客さんの影響を受けず，また事前に仕様を固めておく必要があるということで1年前くらい前から作業が行なわれていました．このため，かなりの部分まで仕様は決まっていました．

2) シナリオは事前に用意した．

この時は私がシナリオライターとなって，機能毎に A4 で合計 20 枚くらい用意しました．また，オブジェクト間の結び付きもあらかじめ図示しておきました．

こうなると CRC ではないという意見もあるかと思いますが，このケースではあらかじめいろいろ用意しておいて検討のベースにしてもらいたいと考えました．

3) 責任の記述方法もある程度形式化しておいた．

これは，実装時の言語が決まっていましたので，その言語形式に近い形で記述するようにあらかじめ決めました．

3.3 観察されたこと

前回の反省と，短期間で開発したいという要求がありましたので，今回は事前準備が過度になったかもしれません．

1) 分析段階から実装に拘る現象が出た．

シナリオや責任の結び付きを最初からかなりはっきりと提示してしまったために，実装の段階のことに目がいってしまいがちだったようです．特に，私の上司に当たるシステム歴が一番長いメンバは，経験的に上流工程から実装を意識した開発を身につけていることもあり，どうしてもその方面に関心がいってしまうようでした．

逆に経験が浅いメンバは，システムの挙動自体に注目することがしやすいようでした．CRC を適用する場合は，できるだけ固有の経験／知識で固まっていないメンバを集めて行なう方が良いように感じます．

2) シナリオから大きく外れる話は出なかった．

シナリオを絶対のもののように受け止めてしまって，それから外れるケースは考えられないようになってしまいました．シナリオの事前準備が悪く働いた例かも知れません．

3) 実装段階でシナリオの見直しが発生した．

シナリオでは，何らかの変更があった場合にはそのオブジェクトが変更を連絡するという形で書かれていました．しかし，実装上はそれが難しく，あるオブジェクトがそれをずっと監視している形でないと実現できないことがわかり，シナリオの見直しにつながったことがありました．

4. 両事例を通じての雑感

4.1 全体を把握している SE が必要

クラスの関連を把握して，全体を見渡せる人がいるべきだと感じます．

きちんと独立した理想的なオブジェクトで構成されるシステムであれば，あるクラスに対して依頼をすればそれをどう実現していても構わないはずなのですが，実際には，全体として矛盾していることとか，無駄になっていることができてしまいます．このため，デッドロックは

ないかとか，無駄なシナリオやルートはないかとか，同じような責任がクラスに跨って存在しないかなど，全体を見渡せる人が必要だと思っています．ただ，これをやれる人は，技術的にも人間的にも優れた人でないととてもむずかしい仕事だと思います．

また，そもそも自分たちのやっていた「責任駆動型」のやり方は，オブジェクト指向になっていなかったのではないかと感じています．これは自分たちの CRC に対するアプローチの仕方が間違っていたのだと思います．独立したクラスを最初にきちんと出さずに機能ベースでクラスを導出してしまうので，DFD 的なものとなってしまい，オブジェクト指向にはならなかったように感じています．

4.2 サブシステム化が有効

ある程度の規模になった場合，サブシステムに分割してその中を CRC で開発するのがよさそうだと感じています．そしてサブシステムに分割する際にも CRC を利用して，まずサブシステム間が少ない関連で済むようにすると良いように思います．このように，最初にサブシステム化することにより，広範囲に Web 化したクラス関係をどうグループかするかという 1994 年に出た問題が回避できると考えます．

なお，この場合でも静的分析はシステム全体でやるべきだと思います．システム全体を見渡す必要性があるので，オブジェクトモデルはシステム全体に対して作成した方が良いと考えています．そしてそこで得られたクラスは，サブシステム間で共有される部品になるのではないかと予測しています．ただし，まだこれはやってみていませんので，わかりません．次のプロジェクトで確認しようと思っています．

4.3 既存の開発方法にどう取り入れるか

これは 1997 年になってから考え始めたことなのですが，既存の開発方法論のどのような段階からどのように取り入れるべきかという問題があると思います．

その会社，あるいはプロジェクト，人によって，いろいろな方法論を持っていると思います．その方法論に取り入れる際，かなり慎重にやらないと結局使われなかったり，役に立たなかったりすると考えます．

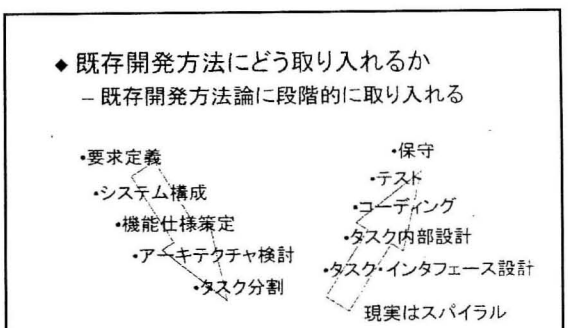

図 1. 既存開発方法論との統合

1994 年のケースでは，要求定義に近い辺りから取り入れてしまったのですが，最初は機能使用の洗い出しが終って，シナリオのネタになるようなことがしっかり出てきた段階で取り入れるのが良いのではないかと考えています．

CRC の考え方が会社全体に行き渡ってからであれば，もっと上流から利用できるとも思います．

4.4 CRC あるいはオブジェクト指向になっているのか.

結局，自分たちは Meyer の５つの基準を満たせたのか疑問です．今 CRC でやったとしても，まだこの基準は満たせないように思います．これは単純に力量不足なのだろうとも思っています．

どうやったらこれができるのかが現在のテーマです．

これにははっきりした回答はありません．「これは５つの基準を満たしているか」を常に意識してやっていくくらいしかないと思っています．

また，やってきたことが CRC になっていないという思いもあります．

現場にとり入れようとすると，いろいろと時間的，メンバ的な問題が存在します．このため，今後は専任のシナリオライターを設け，そのシナリオを叩き台にして皆で検討する形をとりたいと考えています．

それから，CRC でやる場合，システム分析にとどめておくべきで設計や実装まで意識するのは間違いなのだろうかと思い始めています．場面場面での適用を今後はもっと考えていきたいと思います．

5. 質疑応答

1) CRC のセッションは，どの程度の詳細なフェーズまで行なったのでしょうか.

セッションでいうと，期待関係を結んで，システムがそのシナリオに沿って動くところまでは確認しました．

設計の詳細な部分まで CRC で確認しました．

2) RDD とは何でしょうか.

塩谷：Responsibility Driven Design のことで，基本的には CRC の拡張です．Rebecca のグループが CRC に影響を受けてもっとわかりやすく書いたものといって良いでしょう．先ほど武田さんが出したカードの拡張の例は RDD からとったのではないかと思います．

最初の CRC は，OOPSLA のペーパですが，非常に簡単な教育用だけのものでした．そこで，それを実際に設計段階まで使用していくとすると少々力不足であることから，クラス構造なども書けるような拡張が施されました．

また，お二人の話の中にもグループ化の話が出てきましたが，グループ化のポリシーについても述べられています．例えば，こういう傾向が見られたらサブシステムに定義するだとか，こういうところはインタフェースとして特化しようだとかいうようなものです．つまり，オブジェクト把握と関係の把握だけでなく，さらにもう少し詳細に踏み込んで記述ができるような拡張になっています．

ですから，オブジェクト把握くらいまででとどめるのであれば CRC で十分ですが，設計まで踏み込もうとすると RDD を考えた方が良いことになりそうです．

3) オブジェクト指向になっていなかったというような話がありましたが，これは準備とかの問題ではなく，そもそも対象システムがリアルタイム系であるため，オブジェクト指向に馴染めなかったということはないのでしょうか.

その面もあるとは思いますが，データ抽象化がされておらず，Meyer の５つの基準に照らしても適合していないために，そう考えました．

4) サブシステム化する時に CRC を使うと良いと考えているのでしょうか．

サブシステム化する場合に，機能で分ける場合と，役割で分ける場合があると思っています．そして，サブシステム化も役割で分ければ良いと考えています．

5) オブジェクトモデルは組み難いし，自然ではないと話がありましたが，これはどういうことでしょうか．

オブジェクトモデルで全体構造は見えるのですが，それをシステムを実現する際に，モジュールなどにうまく落ちないと感じています．

6) オブジェクトモデルの見方はそれなりに有効だと思いますが，その見方と CRC での分析結果と矛盾することはないのでしょうか．

そこまで詳しく検証していないので，今のところ問題には感じていません．

オブジェクトの切り方などは変わりますが，オブジェクトモデルで出てきたオブジェクトはシナリオに沿って出てきたものではなく，構造から出てきたものだと思います．そして，構造に沿って出てきたオブジェクトはパーツ化しやすいと思いますので，それをシナリオの中でも使えると考えています．

オブジェクトモデルで出てきたオブジェクトでシナリオが影響を受けることもありそうですが，まだそこまでやっていないのでわかりません．

7) CRC とオブジェクトモデルは違う見方なので，違う側面が見つかっているはずだと考えます．オブジェクトモデルで見つかったオブジェクトを CRC のモデルの中にうまくマッピングできるのでしょうか．

オブジェクトの結び付きには，例えば「飛行機は滑走路に降りる」というようなものと，「プロジェクタは電球と鏡からできている」というようなものとがあります．

前者の機能寄りの関係は使えそうに思います．後者の構造的な関係は使いづらいように現在のところは考えています．

（記録：田中慎一郎）

CRC と開発プロセス

(株) アドバンテスト
武田　泰英

1. はじめに

ここでは CRC カード手法を適用した開発のプロセスについて整理していきます．また，このプロセスを支援するために一部をツール化しましたので，それについてもお話します．

1.1 CRC をどのように使うか

1) インフォーマルからフォーマルへ

現在はごくインフォーマルな形で適宜使用しています．今後は，できるだけフォーマルに近付けて使用したいと考えています．

2) まず要求分析フェーズから

現在は要求分析フェーズで重点的に使っています．今後は，できるだけ多くの局面で積極的に使っていきたいと思います．

1.2 どのようなモデルを作成するか

1) シナリオ

シナリオを動的な側面を中心に見るためのモデルとして考えます．

2) CRC カード

CRC カードを静的な側面を見るためのモデルとして捉えます．

3) システム構成モデル

サブシステムも含めたシステム構成モデルを記述します．

4) クラスのパブリックインタフェース

シグネチャとしてクラスのパブリックなインタフェースを記述します．
カードの中の責任は荒いレベルで記述されます．シグネチャを具体的で実装部分に近いものとして扱います．記述形式については，後ほど別に説明します．

1.3 トレーニング

トレーニングに関しては，先ほど「中規模開発での CRC 使用事例」で説明しましたように，CRC の簡単な説明と，オブジェクト指向の勉強会，簡単なセッションを行ないました．

2. プロセス

では，実施した開発プロセスにしたがってポイントを整理します．

2.1 実施方針

1) プロセス実行順序は定義しない．

まず，開発プロセスをどのような順番でやるかについては，定義するのをやめました．定義したところでその通りにはいかないと判断しました．

2) 生成モデルを重視する．

そして，開発全体を通して出力されるモデルを中心に，どのような内容を盛り込めば良いか，それをどのように見つけるか，見つけたものをどのように整理していくかを考えることにしました．

特に，セッションをやっていると，本来注目していたシナリオ以外の話題がかなり出てきます．後々それも検討しなければいけませんので，これをうまく記録することを日頃から気をつけました．ブレインストーミングの応用です．

2.2 要求分析フェーズ

1) セッション前に事前調査をする．

事前調査として，できるだけ要求を洗いだし，理解することを行ないました．

2) シナリオを作成する．

シナリオを作る際の要点は，そのシナリオの入力／出力は何なのか，あるいはシナリオ実行の前後の状態／遷移はどうなっているのか，としました．

もともとは「何々の時，何が起こるか」の形でシナリオを書くように本などには紹介されているのですが，むしろ，入力／出力，あるいは前と後の状態を指定することで，それをシナリオとする形に変えました．

このこと，特に終了条件を明確にすることにより，セッションがおかしな方向に行ってしまうことをかなり防げたと感じています．

2.3 セッション実施

1) 要求の所在を確認する.

セッションを開始する時に，要求としての知識がどこに存在するかをはっきりさせることが重要です．要求を決められる人は誰かとか，どの資料を見れば良いのかという情報を，まず確認しました．

2) セッションルールを確認する.

これは，カードの上げ下げはきちんとやりましょうとか，曖昧で答が出せないようならセッションをやめましょう，などということです．

3) セッションを記録する.

いろいろな局面で検討事項などがたくさんでてきますので，黒板，できればコピーのとれるホワイトボードが必要だと思います．また，できればセッションには直接関係なく，出てきたことをどんどん書き留めていく書記役がいると便利だと思います．

現在，ホワイトボードのコピーをイントラネットの Web で見られるようにしています．それまでは個別にコピーをとったり，どこかに1冊共有のファイルを作ったりしていましたが，現在はみんなでいつでも自由に見られるので，非常に便利です．

72 dpi 程度の保存ですが，ディスクのスペースは結構使います．しかし以前ほど費用はかかりませんので，足りなくなったら買い足せば良いと考えています．

2.4 責任の導出

1) 責任はセッションを通して出てきたもののみ.

カードに書くものはセッションを通して出てきたものだけとしました．

初め，要求からクラス候補を導出する時に，あらかじめ必要がありそうな責任をいくつかカードに書いていました．しかし，結局誰も使わないものもあり，開発の後の段階になればなるほど，それらを本当に消してしまって良いのかの判断が難しくなってしまいます．このため，予想される責任も，先ほどの議事録などにメモ書きするにとどめて，セッションを通して出てきたものだけを正式にとり扱うことに決めました．

2.5 結果の整理および管理

1) 記録をオンライン化する.

セッションが終ったら結果をきちんと管理する必要があります．カードのまま持っていたのでは，バラバラになったり，一度に一人の人間しか見られなかったりします．全員が同時にいつでも見られるように，議事録やカードは基本的にオンライン化します．

2) 結果を整理／分析する.

記録するだけでなく，それを分析することが必要です．

主にやることは，サブシステムを見つけ出すことです．セッション中はクラス中心に動いていますので，意外にサブシステムは見えてきません．コラボレーショングラフを描くことは非常に有効です．

サブシステムの見つかり方には，上から下と，下から上の２つがあります．最初に実行したシナリオが荒い場合には，出てきたクラスは将来サブシステムになる可能性があります．逆にクラスの協調関係を見ながら，ここはサブシステムとして括れるという場合もあります．

やはり，セッションで導出されるクラスの粒度はシナリオの粒度に大きく依存します．

2.6 設計フェーズ

分析が終って，では設計はどうするかですが，これはまだ実践できていません．現在計画していることをお話します．

1) 責任を関数インタフェースに落す．

各クラスの責任を見て，対応する関数インタフェースに落します．ここでどのような手法を用いるかについては，「頭で良く考える」くらいです．

2) 後は従来手法を適用する．

これが決まれば従来通りの形で進められると考えています．従来通りといっても，特に決まった手法があるわけではありません．

3) モデルの検証をこまめに行なう．

責任に対応して関数インタフェースが明確になってきますので，それを作り込んだり，検証するために必要な回りのスタブ化などは比較的楽にできるようになるのではないかと考えています．

そういう意味で，ライブソフトでの検証なども楽にできそうに思いますので，こまめにやれそうに考えています．

3. 支援ツールについて

CRC を実施するに当たって，いくつかツールを準備しました．その説明を致します．

(1) シナリオ管理／検索

要求とシナリオは同じものだという立場に立っています．

図１はシナリオの一覧です．要求を一つ一つ ID をつけてリストアップしています．% の後ろがその要求の ID で，その後ろが親の要求です．

メニューにはトレースの機能を用意しました．

例えば，画面上の ID 「op11」を選択して，メニューの「RecDownTrace」で「サブシステムまで」を選ぶと，その ID に関連したサブシステムの一覧が表示されます．

現在のカードベースのドキュメント管理ですと，小さな単位でデータがたくさん発生しますので，トレース機能がないと何がどこにあるか全くわからなくなってしまいます．トレース機

能は絶対に必要だと思います．現在はまだダウントレースしか作っていませんし，ID を追っているだけのものです．上側へのトレースも本当は必要だと思っています．

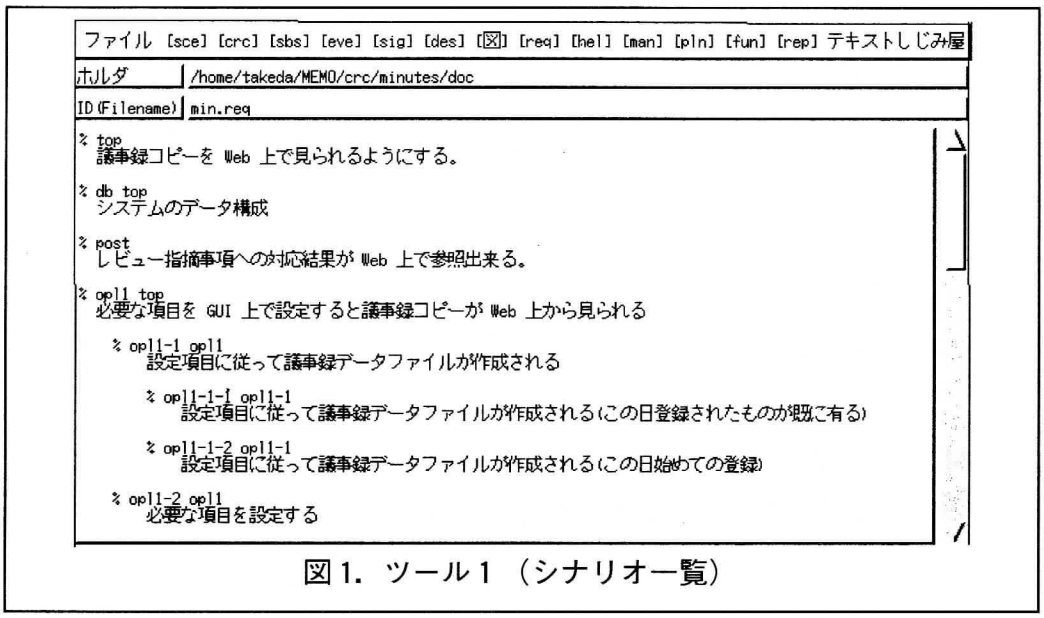

図 1. ツール 1 （シナリオ一覧）

(2) シナリオ設定

図 2 はシナリオです．それぞれのシナリオについて少し細かい内容を記入しています．特に前提条件だとか終了条件などをきちんと定義します．

内容がクラスの協調関係にまで達しているものは，Nancy の本に書いてあったように，誰が誰に対して何を頼むかというメッセージングのシーケンス・フローを書き込んでいます．

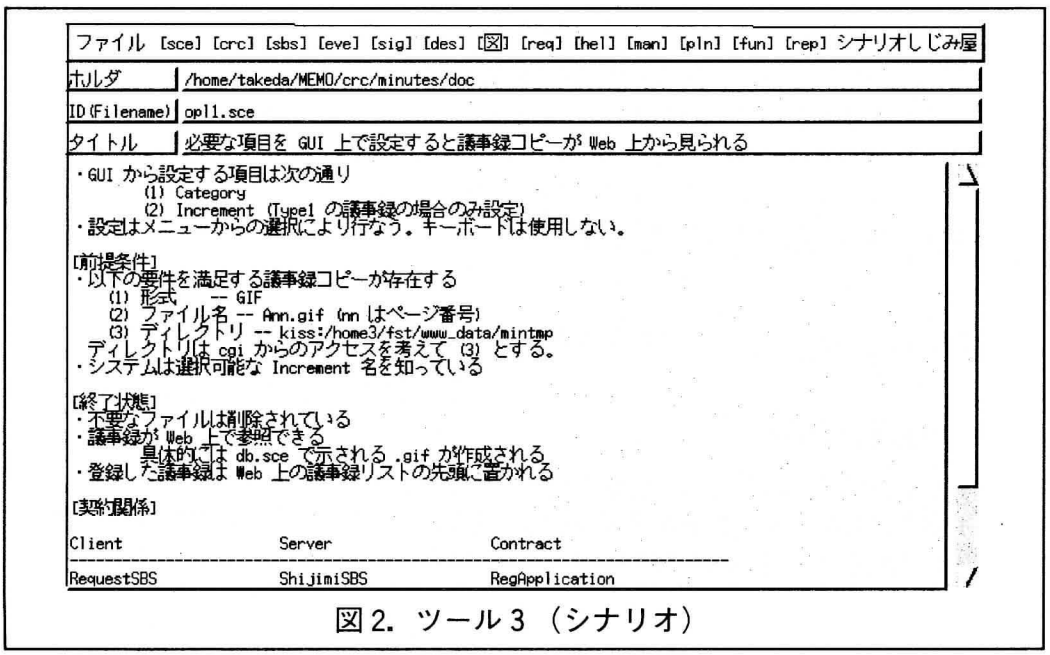

図 2. ツール 3 （シナリオ）

(3) CRC カード管理

図3と図4が CRC カードで，いろいろ拡張してありますが，拡張部分はあまり使われていません．

責任と協力者を書きますが，オンラインデータ上には，責任の下にインデントして対応する関数インタフェースのシグニチャも書けるようになっています．この他，サブクラススーパークラスを書く欄も設けましたが，今のところ，継承関係などはあまり考慮していないのでほとんど使われていません．また，このクラスが属するサブシステムも記入できるようになっています．

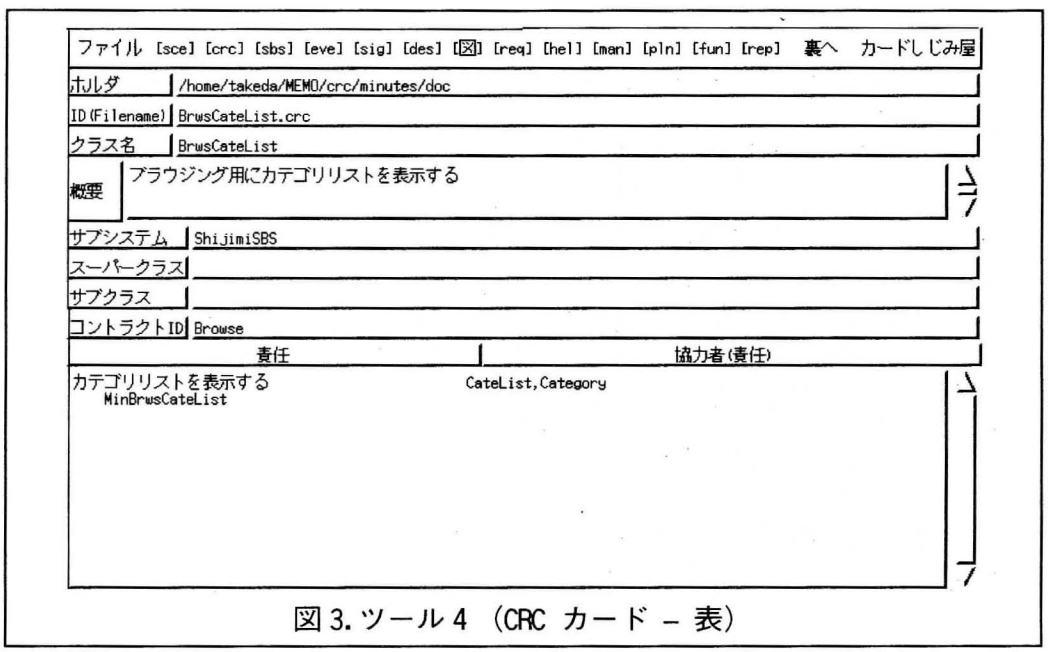

図3. ツール4 (CRC カード - 表)

裏には属性や責任の入出力などの補足を書くようになっていますが，ここを使うのは実装レベルにもう少し近付いてからだと考えています．メニューバー部分にボタンがあって，表裏に変えられます．

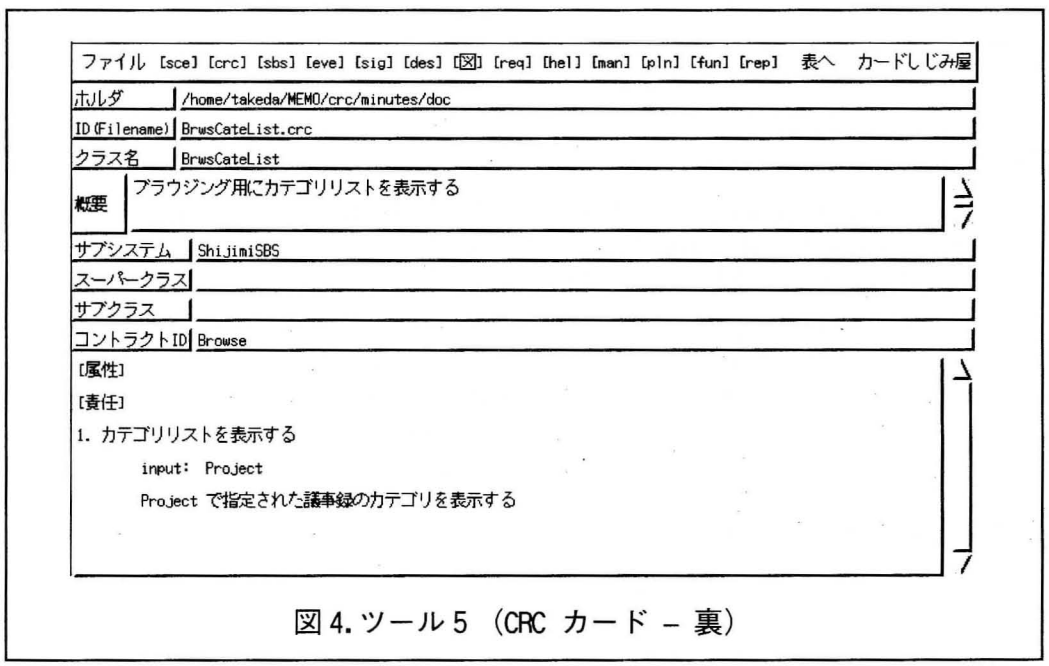

図4. ツール5 (CRC カード - 裏)

(4) サブシステム管理

一つのサブシステムが複数の仕事を持っています．その一つ一つにコントラクト ID をつけています．ID の隣にある←→で次のコントラクトなどが見られるようになっています．

(5) 共通機能

共通機能として検索機能があります．
検索したい ID を選択して検索ボタンを押すと，その内容が出てくるようになっています．

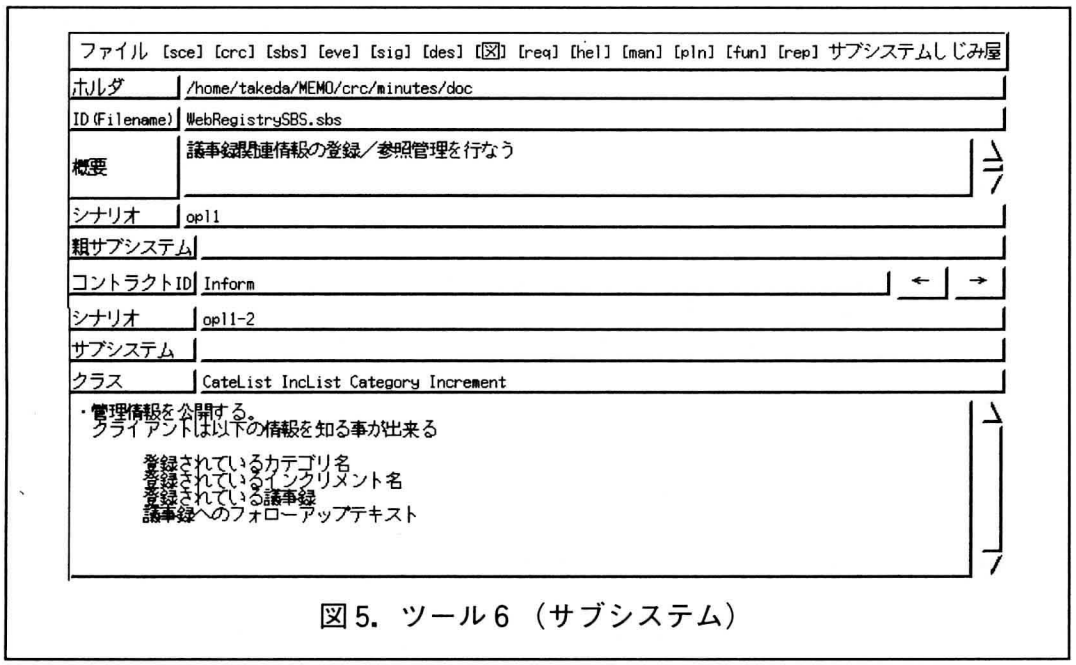

図5. ツール6 （サブシステム）

4. まとめ

(1) プロセス

以上のようなことをやっていけば，要求分析のフェーズは「要求」「シナリオ」「セッション」のループで形成されるように思います．

(2) ドキュメント

このツールを使用すると，ばらばらにたくさんのファイルができてしまうので，トレーサビリティが重要です．また，複数の人が見られるという意味で，オンライン化は重要です．

(3) 従来プロセスとの親和性

従来プロセスとの親和性，特にドキュメンテーションの親和性ですが，設計段階から後は従来手法を用いることにしていますが，特に問題は出ていません．

(4) オブジェクト指向的な考え方の自然な導入

このようなプロセスを踏むことで，まあまあできているのではないかと考えています．

5. 質疑応答

1) セッションの記録方法について，書記がいると便利という話がありましたが，記録のためにはテープレコーダやビデオは使えないのでしょうか．

武田：これまで考えていませんでしたが，テープレコーダだと，図を描いた場合に記録ができないので難しいと思います．ビデオは検討してみたいと思います．

田中：記録を読める形に起こす必要がありますので，単純な記録のためには有能な書記にはかなわないでしょう．

塩谷：３人ないし４人のチームで書記を一人つけるのも現実的ではないので，その場合にはビデオも良いかなと考えます．ただし，後で整理するという面では確かに大変だとは思います．

渡邉：逆に後で客観的に見られるというメリットがあるのではないでしょうか．

田中：後で忘れてしまったことを思い出すために，そこだけ頭出しして皆で見るなどの使い方を考えるのであればとても有効だと思いますが，単純な記録をそれから起こすためにビデオをとるのは，数倍の工数がかかってしまうので無駄だと思います．

ビデオを回しておくのなら，話題やカードの登場毎にこまめに時間やテープカウンタの数字を記録しておくという作業が重要でしょう．

2) セッションが終った後，各自が自分の持ちカードを入力するのでしょうか？ それとも，セッション中にその場で入力するのでしょうか？

セッション中はカードでやります．終ってから入力します．

3) シナリオを事前に整備しておくと言われましたが，入出力条件などがそれほど簡単に決まるのでしょうか．特に状態とか状態遷移が簡単に決まるとは思えないのですが，いかがでしょうか．

どこまで細かく書くかということにもよりますが，きちんと決められないことも当然あります．その場合は，「これをきちんと決めないといけない」ということを事前に確認しておくわけです．その決定はセッション中に試してみることもあれば，知っている人に聞くなり，決めてもらったりすることもあります．

荒く決めるにしても，最終的な落しどころを決めておかないと，メンバが勝手にそれぞれのゴールを決めてしまって，それがバラバラなので収束しないことがあります．これが良いことなのかどうかはわからないのですが，議論が収束しないということから，事前に決めておこうということになりました．メンバ間の認識がずれているままの状態が一番困ると思っています．

4) その場合，過度に実装を意識してシナリオを決めてしまうことが起こらないのでしょうか．

シナリオのレベルでは実装の話はできるだけ入らないようにはしています．ただ，要求として明確なこともありますし，実装を全く意識しないことが必ずしも良いとは限らない気もします．

5) 結果の整理／ドキュメント入力はこのツールを使って直接入力するのでしょうか．

ツールからもできますし，テンプレートにしたがって好きなエディタでデータを作成することもできます．

6) サブシステムを見い出す段階では，コラボレーショングラフを書くなどで行なうようですが，これは専門の担当がいて一人とかでやるのでしょうか，それとも個々の担当がそれぞれ自分のところを中心にやるのでしょうか．

基本的には個々の担当がそれぞれやります．まだあまり，積極的にそれを進めるようにはなっていません．気がついた者が気がついた時点で，普通に伝えられれば一番良いと思います．

7) コラボレーショングラフを描くツールはあるのでしょうか

できれば良いのですが，今はありません．お絵描きツールのようなもので，それぞれ描いています．

全体のコラボレーション関係を見たいというのもありますが，一つのシナリオに関連するコラボレーションを見たいことの方が多いと思います．それを考えると，ツール化することがとても大変そうなので，やっていません．

8) できるだけフォーマルに近付けたいという話をもう少し詳しく説明して下さい．

ここでいっているフォーマルに近付けるというのは，あまりおおげさなことではありません．モデルとして出力するところに CRC を位置付けて定義したということと，ある程度パターンが得られたような責任の書き方をあらかじめ決めておくといった程度のことを今のところ意味しています．まだ，CRC カードに書く時の書き方をフォーマルにすることを指しているわけではありません．

それから，責任などを日本語で書く時に結構誤解を受けることが多いと感じています．英語だとすべて命令形で書かれているなどわかりやすいのですが，日本語になると主体と客体がはっきりしなくなってしまいます．もしかすると日本語はこういうものに向いていないのかも知れないとも思います．ではどうするかというと，英語で書くとかいうくらいしか思いつくことがなく，非常に難しい問題です．

9) このツール自体は何で書かれているのでしょうか．

tcl/tk で，プラットホームは UNIX です．

10) このツールからさらに紙のドキュメントを生成することは考えていないのでしょうか．

現在は画面で見るためのものとして考えています．いわゆる仕様書の作成などは，まだ提出の必要性がないのでやっていないのですが，このままの形のデータではバラバラ過ぎて何らかの工夫を加えないとわからないものにしかならないと思います．ただし，できるだけここにあるデータを元に生成したいと思っています．

中心になるのは，シナリオがどういう形でデータ化されているかですので，その体系を目次的にすっきり見えるようにできれば，残りのクラスなどはまとめておかれてしまっていても，かなり見通しが良くなると考えています．

11) 実装の段階までいってからトラブルが発生した場合，バグの原因などをトレースして探せるようなものになっているのでしょうか．

難しいと思います．実際にはソース・デバッグを行なうことになるでしょう．また，その変更をこれらのモデルにどうやって反映させるかも問題になりそうです．

これは CRC を使ったからではなく，我々の従来手法がもともとそういうもので，ドキュメントは補助資料にしかなっていません．CRC でも同じことが起こりそうな気がしています．

12) CRC をやって，カードが結構できてしまったので，それを整理するためにこのツールを作ったのでしょうか．それとも最初から計画していたのでしょうか．

セッションが終って席に戻った後，みんながカードを見たがるのでオンライン化した方が良いというのが最初の動機です．

その後，先ほどお話したように，「これもいるね」という感じで拡張していきました．

午前中の話で，準備段階で作成したドキュメントツールというのは CRC の部分だけです．

13) 最終的に要求がこれで良いと確定させるのはどのようにするのでしょうか．

現在は会社としての認可というシステムはありません..作ったモデルについて関係部所を集めてレビューなどは行ないますが，開発者が良いと思えばそれでいくということです．

14) 検査部門の検査のしやすさは上がりそうですか．

モデルを見て，検査する項目が見つけられるかという意味からいえば，カードではなく，シナリオ見ていくことになると思います．シナリオの入出力条件などが使えるので，検査項目を出すという意味では多少楽になるかなと感じています．

15) 「シナリオ」というと機能の他に動きが含まれる印象がありますが，どうでしょうか．

武田：「シナリオ」というと確かにそういう印象が強いですが，動作以外のこともシナリオとして書くようにしています．例えば，データ構造などもシナリオのドキュメントの中に書いています．あくまでも要求事項を整理したものとして捉えています．

古閑：シナリオには，システム分析のフェーズで用いるものと，実装のフェーズで用いるものがあるという印象を持っています．実際には一緒になってしまいがちですが，頭ではそのように考えています．

ですから，目的にしたがって，それぞれ別のシナリオがあるべきでしょう．

システム分析の時には，矛盾点がないかとか，これで良いかとか，違うケースがないかとかを洗い出すのが主目的であまり細かいことは扱うべきではないと思います．一方実装の段階では，それをさらに詳細化して，実現できるのかとか，処理速度が満たされるかとかが主目的になるべきだと思います．

16) シナリオは何種類もあるのでしょうか．階層的な問題ではなく，性格の違うシナリオが複数用意されるのでしょうか．

武田：トップのシナリオでは，「こういうものを作る」ということがそのままの形で書かれます．その下に，サブシステム的な要求項目が並ぶ形でブレイクダウンします．この辺りまではセッションではなく，要求を噛み砕く形で別にやります．

古閑：私のやり方だと，シナリオができたら詳細設計が終っているという印象もあります．

本来，理想的にはシステム分析の形で抽象的なレベルで扱い，分からなくなったら一旦詳細化して，解決したら再び上がってくるというようにやりたいとは思っています．

お客さんにシステムの説明をする時にも，実はシナリオベースにやっています．そのシナリオがシステム開発の場に降りてきますので，ギャップが少ないとは言えます．

この意味で，最初にやりたかった「ギャップを小さくする」というのはある程度実現できたと思っています．

17) シナリオ・ベースの手法とはどのようなものなのでしょうか．

塩谷：「シナリオ＝要求」と言い切ってしまった明解さには良い意味でたまげたんですが，シナリオはやはり仕様ではなく，仕様の動的側面だと思います．シナリオという言葉を使うことで，イメージがはっきりして分析する人がやりやすくなるとか，お客さんに説明しやすくなるなどがあり，物理世界にマッピングしやすい便利な言葉だと思います．

しかし，シナリオとは芝居の台本ですから，シナリオ・ベースでいくという本来のやり方は，まず芝居のアイデアがあって，それに役割をつけて，演技指導して,.. というようにアイデアを膨らませて，要求を明確にしていくことなのではないでしょうか．

お二人のケースは，初めから機能仕様がかなり明確になっているので，むしろ実装よりの段階での適用になると思いますので，一般的な適用場面とは多少事情が違うように感じます．

しかし，要求分析のフェーズで適用する場合，完成シナリオに近いものを用意して始めるやり方だと，洩れが残ったまま気がつかないと言う心配があります．洩れが検証できないので，別のアプローチと併用する必要を感じます．

小林：シナリオは最初から完全なセットを用意してセッションを始めるのではなく，まずはもっとも単純なシナリオをいくつか想定して始め，クラスとか責任とかの枠がいくつかできてきたところで，「ではもう少し特殊な例を考えてみましょう」と進んでいくものだと理解しています．

そうすると，一体いつ，分析のフェーズは終ったと言えるのか疑問です．

シナリオを実行するたびに新しいクラスや責任の概念が登場し，新しいクラスや責任が出てくるので，では今度はこのシナリオを実行してみようとなり，堂々めぐりになるか，もしくはやらなければいけないのにそれ以上思いつけなくなって止まってしまうことになりそうな気がするのです．

おそらくは止まってしまうのだと思うのですが，そうすると，そこで「一旦分析は終了して，設計をしてみよう」となり，設計で分からないことが出てくるとまた分析に戻る形になるのか，そういうことが良く把握できていません．

武田：洩れがないほどたくさんのシナリオをカバーして，クラスも責任も洗い出して，さあ実装をしましょうというのは難しいと思います．ある程度の分析が済んだ段階で，まずそれを動かしてみて,.. とぐるぐる回すということをこれからやってみようと思っています．

これではいつ分析が終りになるのかという答にはならないのですが，答を得やすい状況に持っていける条件が揃ってくるのではないかと考えています．CRC を使うかどうかによらず，ここで分析を終了したとしなければならない場面があるとするならば，それは CRC でも他の手法をとった場合と同じ基準で判断することになると考えます．

小林：シナリオを思いつきで取り上げていく形でセッションをやっていて，ある時突然全体の見通しが立って，「ここまでやれば一旦，シナリオに洩れがないかリストアップできるのではないか」と思える時期が来ることはないのでしょうか．

武田：まず，いくつかシナリオをやっていくと「ここから先はさっきのシナリオと同じだな」と分かることはあります．その場合は，「シナリオ XX と同じ」などと記述しておきます．

また，いくつか「状態変数」が見つかってきますので，これを用いて洩れがないかのリストアップができます．具体的に言うと，

a) いくつかのシナリオをシミュレーションした結果，いくつかの状態変数が見つかる．
b) 各状態変数の取りうる値とその組み合わせを考えて，新たにシナリオを起こす．(ある状態変数の値が“＋”になるシナリオ A があるなら，値が“－”になるシナリオもあるはず)．
c) 一通りの組み合わせの分析が終われば，大きな洩れはないと考えられる．

という具合です．

18) 現状システムを理解するフェーズは必要ないのでしょうか．

当然そのフェーズは必要ですし，やっています．システムに対する理解がある程度事前にできていないと，セッションがうまく転がりません．システム全体を全員で考えることに意味があるのですが，理解が浅いものがいるとうまくいかないのです．

全員で考えているかの一つのバロメータとして，カードの上げ下げがきちんとされているかどうかがあります．アクションをとらせることはとても効果的なことのようで，「だまされたと思ってやれ」と言ってでもやらせることが重要です．

また，別の見方として，CRC の一つの利点として本当は良く分かっていないところが見つけやすいというものがありますから，それは「調べなければいけないこと」と書き出しておく使い方もできると思います．

19) １回のセッションに適当な時間はどのくらいでしょうか．開催時間は考慮して行なっているのでしょうか．

集中度が高いので，２時間はちょっときついようです．１時間から１時間半くらいで「休憩したい」との声が上がります．

休憩をとると，休憩の後，またウォームアップ時間が必要なので，非効率な面もありますし，休憩時間の間に忘れてしまうというひどいこともあります．

シナリオを作成する時に時間に合わせた粒度を考えておくのも必要かも知れません．経験上は考えても実際に行なうとあまりあてにならないのなのですが,..

ウォームアップ時間を短縮する意味では，カードだけを使っているとやはり時間がかかります．そこで，コラボレーショングラフのようなものをホワイトボードなどに書いておくと，戻ってきた時に見ることで思いだし易いと言うことがあります．

20) １つのシナリオを構成するクラスの数はどのくらいでしょうか．

分担してクラスを持ってセッションをやるわけですが，全体の流れが理解しやすいという意味では，一人２～３のクラスが適当に感じます．５個も１０個も一人が持っていると，どれを上げているのか分からなくなって，誤解や混乱が多くなるように思います．

しかし，ではカードが増えたら人数が増やせるかというとそれは非常に難しい問題です．

（記録：田中慎一郎）

CRC ワークショップ全体討論

進行：(株) SRA 田中　慎一郎

事例に関する質問を続けていては発表者にメリットが少ないため，一旦討論を打ち切って，視点を変えて討論を行なうことにしました．その経緯を以下に記録します．

1. 討論テーマ候補について

討論テーマの候補に関しては，開催前の準備段階では以下のようなものを想定していました．

- 「分析フェーズの罠」にかからないためにはどうすれば良いか．
- CRC 手法を適用する場合の効果的なドキュメンテーションはどのようなものか．
- 開発規模の違いが CRC 手法にどのような影響を与えるか．
- 他の手法／ツールとの統合は可能か．どのようにすれば良いか．
- CRC 手法を導入するにはどのような前提条件／事前準備が必要か．

また，ご両名の発表からキーワードを拾うと以下のようなものが考えられそうでした．

- モデレータ／書記，あるいはシナリオ演出家／まとめ役の重要性
- シナリオ神聖視問題
- カードの表記法，記録方法，図の利用（ドキュメンテーション）
- 責任の命名方法（日本語の曖昧さ？）
- 効果的な CRC の適用場面，あるいはタスクの分割や実装のフェーズへのつなぎ方

そこで，これらの候補テーマを参考に討論セッションをどのように使用するかを最初に簡単に全体で議論しました．

1.1 テーマ選定のためのフリートーク

田中：セッションの進行のさせ方について興味があります．モデレータあるいは書記の役割や，シナリオ作成の責任者の役割などについて良いアイデアがあればお聞きたい．ガイドラインみたいなものができれば役に立つような気がします．

しかし，「セッションが混乱したらその時点で問題点を確認してあっさり止める」というのをルール化すれば良いという結論で済むかも知れないとも思います．

古閑：システム分析から実装に進む部分が良く分からないので，その辺りに関する皆さんの意見が聞きたいと思っています．特に，RDD についてとか，CRC をシステム分析に用いてその後は他の手法を用いるべきだとか，そういう点について関心があります．

小林：開発規模が大きくなった場合にどうなるかという点に関心があります．
一人辺り数枚のカードで，数人でのセッションが適当だということですが，それでは２０～３０クラス程度のシステムしか作れないことになってしまいます．大きくなった場合には適用不可能な手法なのか，うまい方法があるのか．特に，機能分割と全体のマネージメントのような問題については本には触れられていないので，それについて考えてみたいと思います．

小林：もう一つ，分析と設計のフェーズは，本でも章が分かれていて，セッションのやり方も違うように説明されています．
実際に，分析，設計，実装と進む場合に，どのような発想の転換が必要かについても考えてみたいと思います．

塩谷：1989 年に OOPSLA で Kent Bewck の発表を直接に聞いて，非常に良いと思い，SIGENV で紹介しました．分析が集団でできる点が最大の利点だと思います．
これまで分析は集団では行ない難く，個人が集中して，泥沼にはまりながらやらなければならないのが普通でした．みんなでやれば恐くないではないですが，皆で辛い道を一緒に歩むのは大変良い点だと思います．
これはカードを使ったブレインストーミングであるからこそできることだと思います．しかし一方で，ご両名の発表にもありましたように，ブレインストーミングになり過ぎると収集がつかなくなるので，方向性を与える工夫をする必要があるでしょう．かといって，今度はそれが制約の強い「ガイドライン」になってしまうと，本来出てくるはずのもっと良い発想が潰されてしまう可能性もあります．
結局，セッションの運営の仕方がとても重要だと考えるので，その辺りを議論してみたいと思います．

1.2 テーマ選定

おおざっぱに整理すると，主な関心は以下のようになると思われました．

1) 分析をブレインストーミングで複数人でやる手法と捉え，その運営方法について．
2) 分析から設計／実装のフェーズに進むにはどのような発想の転換が必要か．
3) 大規模のシステム開発に適用するにはどうするか

このため，下の方の問題を念頭に置きつつ，一番上から始めることとしました．

2. 討論

(1) セッション実施時の工夫について

塩谷：OO ブレインストーミングが CRC の本質だと考えています．Kent Beck 自身が元の論文の中で，「Macintosh の Hyper Card でプロトタイプを作成してみたが，うまくいかなかった．これはコンピュータ化しないでカードでやった方が良い．しかし，ツールの整備は重要なので，その内には何とかしたい」と書いています．

00 ブレインストーミングと考えた場合，やはりある程度の指針は必要だと思います．これがないと発散し過ぎてしまうでしょう．ただ，自由な発想が阻害されないようにはしたいと考えます．そういう意味では，あまり厳しい指針は示さない方が良いでしょう．
武田さんたちはすでにそういう工夫をいろいろされていて，

- 混乱して来たら打ち切る．
- スーパーバイサや書記を導入する．

などを上げています．スーパーバイザや書記の導入は非常に重要だと思いますが，今日この場で言われるまでは明確には意識していませんでした．
以下に私が考え付いている工夫を挙げてみましょう．

1) 変わったことは色は変えて書き込む．

坂北のワークショップの時に行ったのですが，カードに書き込んだことは消さないで，変更は色を変えて書き込みました．これはカードに書き込む情報量を増やすことができます．

2) カードに色を塗る．

これは一人が扱うカード数を増やす助けになりそうな気がします．

3) 顧客を交えてセッションを行なう．

知識のありかに関して，セッションに入る前に前提知識を用意したり，知識を持った人のスケジュールを調整したりする話がありました．その意味で，お客さんなどを交えるのは大変有効だと思います．
しかし，その人は完全にオブザーバに徹していてもらって，セッションメンバが「わからない」となった時に教えてくれる形式が最高だと思います．

4) 書記はできるだけ欲しい．

書記は，やはりできるだけ用意したいですね．また，書記がオンラインでダイレクトで入れられるようなツールがあると一番良いと思います．
書記が用意できない場合は，ビデオをとっておいて分からなくなったら戻って確認するという使い方が良いと考えます．毎回それから全記録を起こすのは現実的ではないと思いますが，分からなくなった時のためなどには非常に有効でしょう．

5) ツールによる支援を充実させる．

後はうまいツールの利用ができればやはりうれしいですね．メッセージグラフにしてもコラボレーショングラフにしても，見たい時にすぐ見られるようにできれば，これは大変価値が高いでしょう．
セッションの間は，カード情報の整理よりも，メッセージ関係やコラボレーション関係の方が重要ですし，それだけなら入力もしやすいと思います．

武田： ブレインストーミングには，「集まって思いついたことをどんどん出す」と，「出たものを分析する」との２つの段階があると考えています．CRC セッションでは，カードに書かれたものが結果として残るわけですが，この時に，セッション中に出たものを出た言葉のまま板書しておいて，でき上がったカードと見比べて何が起こったのか考えると面白いと思います．
逆にそういうことをしたい場合に，ビデオだと可能なのかは疑問です．

塩谷：モデレータに関しては，武田さん方式で打ち切ってしまうのが良いように思います．モデレータがいちいち説明するのでは，それで余計なバイアスがかかってしまうし，打ち切って頭を冷やすのが良いでしょう．

田中：私もそういう取り決めにしておけば良いと考えます．後は，どこで打ち切るかの判断だけでしょう．全員が慣れてくれば必要ないでしょうが，導入当初は「混乱してきたら打ち切り提案をする」ことをいつも気にかけている役割の人は必要な気がします

塩谷：書記がモデレータを兼任することはできると考えますか．

武田：打ち切るためには原因がどこにあるのかを判断できないといけないでしょう．
結構良くある混乱は，言葉の使い方がおかしいケースです．言葉の解釈がそれぞれ違うと，無用な混乱をします．言葉の使い方がずれている時と，解がないのにもめている時は中止すべきだと思います．しかし，その判断は対象システムの知識がかなりないと難しいと思います．

田中：書記を板書能力だけでなく，ある程度対象システム知識があるものに依頼するという方法はつかえるでしょうか．
板書といっても，ある部分自分で解釈して書き留めているはずなので，意味がずれていたり，同じことを言いあったりしていて，うまく板書がとれなくなることが一つの判断基準になりそうに思います．

武田：モデレータは誰がやっても良いと思いますが，セッションに参加している当事者は，どうしても熱中するので，そういう気が回らなくて向かない部分があると思います．そういう意味で，セッションに参加していない書記がその判断をするのも良いケースがあるかも知れません．

渡邉：セッションでは，カードは全員が見えるように全部並べたのでしょうか．それとも担当が抱えてしまうのでしょうか．以前に試行した時には，全部机に並べて全員が同じものを見られるようにして行ない，整理などのためにホワイトボードに書き出したりしていました．

古閑：最初はカードを個人別に持っていて，ある段階でホワイトボードにそれを張り付け，丸で囲んで大きな関係を示しました．ホワイトボードにはすべてのカードを張れませんので，途中からはクラス名だけをボードに書いて関係図を作りました．

武田：机も何もない畳の部屋に車座になってやったりすると良いのかも知れませんが，実際には古閑さんと同じような形で行ないました．

渡邉：コラボレーショングラフのようなものがある程度全員の頭の中に見えてくるようにならないと，分析が出来たと言うことにならないのではないでしょうか．

武田：グラフ化がツールで出来ると大変有効だと思います．しかし，おそらく無理でしょう．

塩谷：停滞／混乱の話のところで，「同じ言葉を違う意味で使うことがあるため」というのがありました．それは，結局，カードの書き方が不適切であるということではないでしょうか．
これは日本語の問題ではなく，単に訓練されていないだけのことでしょう．明確に表現する訓練がされていないので，明確に書けない．これが日本における要求分析を難しくしている点だと思います．
要は「正しい日本語で明確／簡潔に書く」ことに尽きるのですが，それだけではあんまりなので，一つの方法として DoD がやっているらしい「語彙と表現形式を限定する」というのを採り入れるのも良いと考えます．
最終的には相手はコンピュータなのですから，，シンタックスを限定してしまうのは大丈夫でしょう．語彙の方は，発生時点で辞書ツールに登録するなどの方法で，かなり明確にできると考えます．

田中：言葉の修飾関係などが分かり難いという問題は避けられないと思いますが，どうでしょう．

武田：言葉の指し示している意味範囲の捉え方の違いという問題もあるでしょう．

塩谷：それらは定義を明確にすることによって解決できる問題と思います．

(2) カードや責務の整理方法について

小林：責務が増えていった場合，いつどのように分割するかについての指針はあるでしょうか．

渡邉：メタクラスというか，抽象クラスをうまく導入することになると考えます．

塩谷：この問題は，分析から設計にいつどうやって移るかに関してと同じでしょう．分析の結果，クラスの構造や依存関係が偏ったりするのは良くあることです．これを是正するのがこの場合の設計ではないかと考えます．

小林：責任が増えて偏った場合に是正する話が本にもでていますね．これは図書館システムの司書の話です．借り手と司書の間のやりとりをトランザクションという概念として捉えてしまう方法です．

トランザクションのような概念は現実社会にはないものです．例えば，今何冊借りているかは本来なら借り手に聞くべきことですが，コンピュータの世界では借り手は「リストを示す」というような責任に変えてしまうことが可能なわけです．

こういうことを方法論として知っているか知らないかで大部違うような気がします．

塩谷：CRC や RDD はデザインパターンの１つだと考えられています．オブジェクト指向の設計法といっても，図だけはいろいろ出てきていますが，中身はあまりない．しかし，図や記法がいろいろあるので，たいしたことをやっているように見えてしまう．それに比べると CRC などは，図もないし，安っぽい手法に見られがちですね．

他の手法と同じで，実際にたいしたことはないのだから仕方がないのですが，それを他の分野のアイデアを借りて，もうちょっと格好良く見せようというのがデザインパターンだと思います．

設計の時はクラスを整理したり，抽象化クラスを作ったりすることが大事な作業になると思われます．しかし，CRC に限らず，はっきりした指針はないでしょう．あくまでもケースの羅列をするのがこの業界の現状だと思います．

協力関係をうまく整理して図視することにより，依存関係の局所化が分かります．それをうまくグループ化することにより，抽象化クラスを見い出すことが出来るでしょう．これがかなり独立したインタフェースを持つようなものであれば，それをサブシステムと呼ぶことも出来る．その程度のものではないでしょうか．

コラボレーショングラフがきれいに整理された形で出来ていれば，良い設計になっていると言えると思います．スパゲッティになっているのは，あまり良い設計とは言えないでしょう．指針はないとしか言えませんが，コラボレーショングラフが一つの目安になるとは言えるかも知れません．

——休憩

塩谷：休憩時間に廊下で話した結論は，骨董屋とおんなじで良いものをたくさん見るしかないということでした．

小林：問題は，いいデザインがまだそんなにストックされていない世界だということでしょう．

(3) 大規模システムへの適用に関して

小林：セッションメンバは５人程度が適当で，一人の持ちカードは５枚くらいが混乱がないとすると，せいぜい３０クラス程度のシステムしか構築できないことになってしまいます．
現実のシステムでは，少し大きくなるととてもこのくらいでは収まらないので，セッションがやりつづけられなくなって挫折するケースが出てしまいそうに思います．こういう場合にどういう形をとれば良いのかをここで検討したいのですが，いかがでしょう．

武田：一度に見なくてはいけないクラス数をどうやって制限するかを考えないといけないでしょう．現在のシナリオに直接関係のない部分はサブクラスに押し込むなどの工夫が必要ではないでしょうか．

小林：サブシステムに分割した後で，それぞれ別のチームが並行してセッションを行なうことは可能だと考えますか．

武田：可能だと思います．
気をつけなければいけないのは，サブシステムを見つけたとたんに，それに関わるシナリオだけを一所懸命考え出しかねないことです．それでは外側から見たシナリオとはかけはなれてしまう可能性があります．

古閑：システムが大きい場合，最初からサブシステムに分割してしまうと言うことは無理なのでしょうか．

武田：特に意識しなくても，シナリオの粒度が揃わないので，自然と分割されていくようになると思います．
最初に粒の大きなシナリオをいくつかザッとやってみるのも効果的なのかもしれません．

渡邉：別々のグループで同じようなオブジェクトを別々に作ってしまうことなどがありそうですが，どうでしょう．

塩谷：複数グループが並行して走るなら，そのコーディネーションがどうしても必要になるでしょうね．それぞれのグループのコーディネータと全体のコーディネータの両方が必要でしょう．

渡邉：しかし，途中であまり情報が洩れるのも困るようにも思います「あちらで良さそうなのを作っているからそれを借りよう」などと勝手に決めてやっていたら，向こうではそのクラスはなくなってしまったなどということがありうるでしょう．

塩谷：そういう本質的な問題もありそうですが，本来，大規模システムの場合，わざわざ分割を考える必要はないのが普通です．機能的，政治的に，元々分割されていることが多いでしょう．

渡邉：１回のセッションには１時間半くらいが適当であるという話があったと思います．
１日に何回かのセッションを行なっている内に，一人辺りのクラスの数が１０とか２０とかになってしまった場合には，これは設計が悪かったとしてまたやり直すのでしょうか．

武田：一つのシナリオでたくさんのクラスをカバーしなくてはいけない場合は，現在は，シナリオを分割しています．例えば，ＡをやってＢをやってＣをやるというシナリオなら，Ａ，Ｂ，Ｃの３つに分けるのです．

また，ある程度セッションをやっていくとクラスの数は安定し，新しいクラスはそれほど出てこなくなります．ある程度クラスの数が安定した段階では，セッションは全員でやらなくても良いように思います．

例えば，ある部分はシナリオ検証のために一人でやってみるなどです．

古閑：クラスの数よりも関係の複雑さの方が後で問題になると考えます．やっている時はクラスの数自体が多いのはあまり気になりません．

失敗だったのは，責任駆動であるべきものが機能駆動になってしまったことだと考えています．途中でクラスのみ直しをすべきだったように思います．

塩谷：表面的なチェックとして，グラフの冗長さを示してくれるようなツールを利用する方法があるかも知れません．

藤尾：クラスの再利用に関してはどう考えていますか．

武田：今は考えていません．出来た暁には，使えるものがあったら使うかも知れませんが，積極的に再利用を目的には考えていいません．

そもそも，再利用を考えてクラスを作ることはあまり簡単なことではないし，必ずしもコスト的に得なものではなさそうに思います．

塩谷：再利用と言うのはソフトウェアの世界だけで，普通の工学では利用としか言いません．きちんとした部品化は今のところ出来ようがないので，オブジェクト指向だから再利用がしやすい」というのは単なる幻想にしか過ぎないと思います．

共通部品を生み出すのは大変な苦労があって，それはいろいろなケースを設計時からすべて想定するなどの苦労で，それを押してまで部品化するメリットは通常はないと思います．

3. まとめ

大変 SEA のワークショップらしい，とてもまとまったとは言えない展開になりました．CRC は我々環境分科会の持ちテーマの１つですので，今回のワークショップもできるだけ記録として残し，今後につなげたいと思います．

古閑さんと武田さんにはいろいろな意味でずいぶん無理をかけてしまいました．最後に参加者を代表して感謝の意を表します．今後の成果もまたご報告いただきたいと考えます．どうもありがとうございました．

（記録：田中慎一郎）

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.iijnet.or.jp/sea