



超ビギナー入門

CBDs&XP

NCS CBDs 研究会

NCS XP 研究会

日本コンピューター・システム株式会社

(<http://www.ncs.co.jp/>)

はしがき

この研究会は、オブジェクト指向技術に興味を持った社員が、事業部を超えて全社レベルで集まり、会社の組織でも、会社の指示で行っているものではなく、有志で活動しているものである。パッケージ開発、S Iシステムインテグレーション開発、ソフトウェア受託開発と担当もいろいろであり、所属する事業本部も様々である。若手プログラマから中堅ソフトウェア開発者、開発チームリーダー、プロジェクトマネージャと様々な立場で、昼間は実業務のプロジェクトに属し、勤務終了後、夜な夜な集合する、裏街道プロジェクトである。特徴としては、初心者集団である、ということで、CBDやXPの専門家が集まっているわけではない。本書は、その初心者集団の強みを活かして、書籍などを見てCBDやXPに興味を覚えた素人が実際に取組もうとするとどんな目にあうか、どんな壁に当たったかということだけは自身を持って語れるのである。研究会は、決してすんなり進んで行ったわけではない。メンバー全員が頭を抱え、苦労したどたばた体験記とも言える。取組むに当り、メンバー全員でどんな勉強をしたのか、そこで吹き出した疑問、実際にやってみたこと、そこで感じたこと、どうやって解決したか、について述べることにする。よって、これからCBDやXPに取組もうとされている初心者の方々には、実践しようとするとき必ず疑問に突き当たる項目について参考になるべくバイブルとなれば幸いである。尚、本書を作成している段階では、まだ研究会は活動途中であり、本書のタイトルである超ビギナーという言葉をご理解いただきたい。この次には、超ビギナーという文字をつけなくても良いCBD&XP入門を発行したい。

これからも、研究会活動を行いますので、「我こそは」という方がおられましたら参加していただきたい。

また、壮大な目標ではあるが、社外との情報交換を行い、ソフトウェア開発者のコラボレーションによる新しいコミュニティを広げて行くことを目指している。

本書が多くのソフトウェア開発者にとって良き参考書となれば幸いである。

平成 14 年 4 月 10 日

NCS CBDs研究会・NCS XP研究会一同

目 次

序章 それから、はじめる.....	1
0.1 はじめる.....	1
0.2 CBDs.....	1
0.3 XP	3
0.4 NEXT	5
第一章 CBD概説.....	7
1.1 コンポーネントベース開発の定義	7
1.1.1 背景.....	7
1.1.2 コンポーネントとは.....	7
1.1.3 コンポーネントベースとは.....	7
1.1.4 コンポーネントで重要なこと.....	8
1.1.5 コンポーネントの粒度は大きく.....	8
1.2 コンポーネント開発プロセス.....	10
1.2.1 CBDの手法のひとつ	10
1.2.2 開発プロセス.....	10
1.2.3 要求分析	11
1.2.4 要求分析のポイント.....	11
1.2.5 システム分析.....	12
第二章 CBDS 実践例	13
2.1 実践のテーマ.....	13
2.1.1 課題の設定.....	13
2.1.2 CRCカードとは	13
2.2 実践記.....	14
2.2.1 Getting Started.....	14
2.2.2 ビジネスプロセスを知る.....	14
2.2.3 ユースケースを作成する.....	15
2.2.4 コンポーネントを見つける.....	16
2.3 まとめ	18
第三章 XP概説.....	20
3.1 XP って何.....	20

3.2	XPのコンセプト.....	20
3.2.1	4つの変数.....	21
3.2.2	4つの価値.....	21
3.2.3	12のプラクティス.....	21
3.3	開発プロセス.....	22
3.4	XP研究会～よもやま話～.....	23
3.4.1	XPに興味を持った理由～O氏の発言～.....	23
3.4.2	XP? WindowsXP?.....	23
3.4.3	12のプラクティスを勉強しよう.....	23
3.4.4	シンプル設計って、何?.....	23
3.4.5	リファクタリング.....	24
3.4.6	テストファースト.....	24
3.4.7	ペアプログラミング.....	24
3.4.8	プラクティスの支え合い.....	25
3.4.9	ところで、xUnitで.....	25
第四章	半かじり実践記.....	27
4.1	さあ、実践.....	27
4.1.1	何を作るか.....	27
4.1.3	ストーリーカード管理システム.....	27
4.2	壁との戦い.....	27
4.2.1	ストーリーカードを書く.....	27
4.2.2	タスクカードに分割.....	28
4.2.3	インフラ整備等はタスク.....	29
4.2.4	第3の壁、そして第4の壁.....	30
4.2.5	再度、クラス図を書く.....	30
4.2.6	まずは、テストから～実践したプラクティス.....	31
4.3	まとめ.....	32
4.3.1	XPへの険しい道.....	32
4.3.2	XPを広めるために～プロジェクトに活用しよう.....	32
4.3.3	今振り返って、そして今後の課題.....	33
A	活動報告.....	35
A.1	NCS CBDs研究会.....	35
A.1.2	NCS CBDs 研究会とは.....	35
A.1.3	NCS CBDs 研究会の目的.....	35
A.1.4	活動経過.....	35

超ビギナー入門 CBDs&XP

A.1.5 今後の活動.....	35
A.2 NCS XP 研究会.....	36
A.2.1 NCS XP 研究会とは.....	36
A.2.2 NCS XP 研究会の目的.....	36
A.2.3 活動経過.....	36
A.2.4 今後の活動.....	36
B 参考文献	37
あとがき	38

序章 それから、はじめる

0.1 はじめる

ここでは、技術的な面については述べていない。CBDs (Component Based Developments) や XP (eXtreme Programming) をアプローチするために「知っておかないといけない思い」について述べることにする。

0.2 CBDs

CBDs は Component Based Developments の略である。訳せば、コンポーネントベース開発になる。その、コンポーネントベース開発では、必ず言われる言葉がある。それは、「作るな、使え」という言葉である。コンポーネントベース開発の世界ってどんなものなのか、Sler を例にした簡単な図で示す。

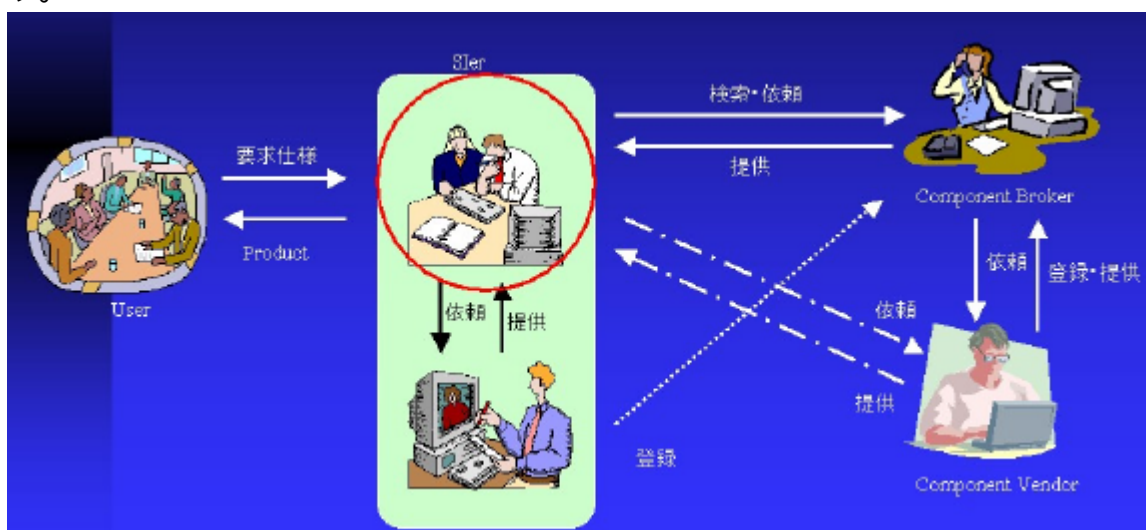


図 1 CBDsの世界

ユーザからの要求仕様にもとづいて、Sler は製品を作って納める。では、コンポーネントベース開発でSlerはどうするのか。コンポーネントブローカーに必要なコンポーネントを検索し、それを購入する。ここで、それぞれの役割について簡単に説明する。

まずは、コンポーネントベンダは、自ら作ったコンポーネントをコンポーネントブローカーに登録し、コンポーネントブローカーや Sler から依頼されたコンポーネントを開発し、

提供する。

コンポーネントブローカーは、Sler から依頼されたコンポーネントを提供する。もしなかった場合は、コンポーネントベンダにコンポーネントの作成を依頼する。

Sler は基本的にはコンポーネントブローカーに必要なコンポーネントを依頼し提供を受けるが、なかった場合は直接コンポーネントベンダに依頼し提供を受けるか、自前でコンポーネントを社内で調達する。Sler がその作ったコンポーネントをコンポーネントブローカーに登録することもある。

では、今、丸で困ったところはどうするのかである。コンポーネントをつなぎあわせて製品をつくるのである。

このとき重要となるのが3つのAである。¹

コンポーネントベース開発で行う場合、組立てて構築できるように考えなければならない。また、コンポーネントを利用するということはコンポーネントとシステムのアーキテクチャ、構成に整合性が必要となる。そして、最後に、コンポーネントは交換可能でなければならないから、プラグアンドプレイではないですが、適応させることも考えていなければならない。

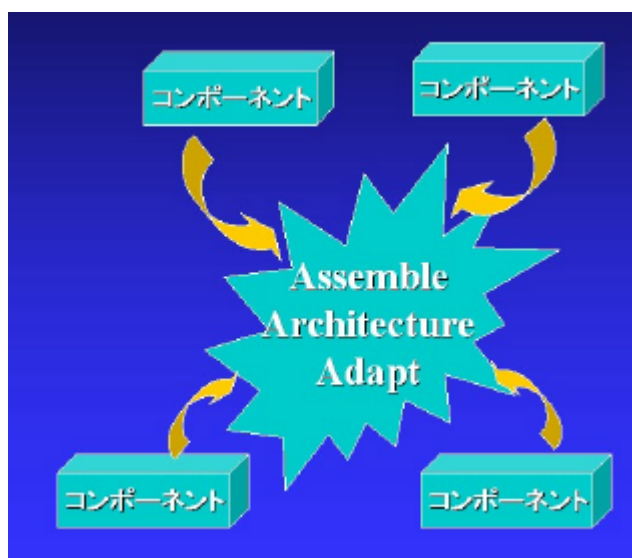


図 2 3つのA

ここからは、再利用について述べることにする。

コンポーネントベース開発も再利用をうたっているが、ソフトウェア開発における「再利用の目的とは何か」である。それは、生産性と品質なのである。しかし、この当たり前のことが、なかなかされずに、いろいろと考えられてきている。

そこで再利用を阻害する最大の要因は何かということである。実は、心理的な要因

¹ アラン・ブラウンとキース・イエガーが 1998 年に「The Future of Enterprise Application Development With Components and Patterns」で書いている。

が大きいと言える。プログラムは、「人の作ったものはいやだ」とよく言う。だがしかし、自分の作ったプログラムは人に渡せないほど陳腐なものなのではないか。他の人から「あなたの作ったプログラム」を同じように言われたいのか。決して、そうではないと言える。プロジェクト管理者は「そんなもの使ったら、責任なんかとれない」と言うが、その前にプロジェクトマネジメントに対する責任をきちんととっているのか。それに比べたら、コンポーネントを使うリスクの方が小さいと言える。ユーザの立場だと、「保証されない」と言う。実際は、利用してもそれ程リスクはないと言える。

そのためには、再利用しやすい環境が大事である。それは、組織や教育も含めてである。教育とは大学における教育をさす。特に、ソフトウェア工学について大切である。ところが、情報工学や経営工学でソフトウェア工学は選択科目になっているところがある。これは大変な問題である。

では、いつまでも再利用できないでいるとどうなるかである。「生産性の向上は？」どうか。それほど画期的に向上させることは難しいと考える。高い品質には高いコストが必要ということになる。ということは、海外生産には太刀打ちできないことになる。それは、日本の半分の価格でプログラムが作成されるからである。そうすると、結果は簡単だ。仕事がなくなるということである。まして、「開発プロセスは改善できるのか」、CMMやSPAって叫んでいるけれど本当に出来るのか。大いに疑問を投げ掛ける。その結果として、顧客満足度は高くない。結局、ユーザが損することになる。

再利用ができるということはどういうことになるか。再利用ができるということは、プロフェッショナルの証明であり、しっかりした開発手法を持っていることになる。さらに、再利用可能なソフトウェアを提供できれば、そしてそのものが、再利用可能なソフトウェアを組み合わせ提供したものであり、かつ、多くの人に使われることになれば非常に高い生産性と品質を持ったものを提供できる技術力があるということになる。

やはり結論は、「作るな、使え」となる。

0.3 XP

XPは「銀の弾丸」²ではない。残念ではあるが。オブジェクト指向技術の書物にはしばしば出てくる「銀の弾丸」ではあるが、情けないことに、この言葉から今も逃れられないのである。まさしく、呪縛と言える。しかし、変わらなければ...いけないのである。

変わるために、知っていなければならない2つの言葉がある。それは、「取り戻そう」と「New 職人」である。

まずは「取り戻そう」と言う言葉から述べる。誰が、取り戻すか。「ユーザが」、「プログラマーが」が取り戻すのである。誰から取り戻すか。えたいの知れない人々から、システ

² フレデリックス・ブリックスがもう20年以上前に書いた「月月の神話」に書かれている有名な言葉。

ムエンジニア」や「システムコンサルタント」と言う人達からである。この人達は何をするのか明確に定義できるか。システムアナリスト・システムアーキテクト・プログラマと明確にそれぞれを行う人がいる。しかし、システムエンジニアはない。一方、コンサルタントは、マネジメントつまり経営からコンサルタントすると言わないといけない。

もう少し詳しく述べることにする。

まずは、「ユーザが取り戻そう」から述べる。要求仕様はユーザのものである。でも、要求仕様だけなのか。もっと大事なものがある。それは、「思い」³である。実は、これこそが大切である。ユーザの「思い」は、ユーザしか書けない。ユーザだけが、「思い」を伝えられるのである。当たり前のことだが、本当にそうなっているのか。

次に、プログラマが取り戻すものは何かである。その前に、まず、はっきりしなければいけないのは、プログラマはコーディングとテストだけが担当ではないということである。システムに対して Plan-Do-See できるのはプログラマだけである。設計からリリースまでをきっちりとできることがプログラマと言える。本当にその責務を果たしているのか。そしてもうひとつ大事なことがある。ユーザの「思い」を直接受け取れるようになることである。

ケント・ベックは「私はプログラマだ」と誇りを持って語る。多くのプログラマも「私はプログラマだ」と誇りを持って言えるようになって欲しい。

再度、この言葉を提言する。取り戻そう「思い」を伝えるユーザに。取り戻そう「思い」を受け取れるプログラマに。

もうひとつの言葉、New 職人について述べる。ここでは、一人工程と匠の世界という点から見てみる。

一人工程が意味するものは、製造工程の考え方が変わるということである。従来の分業・ベルトコンベアからすべてを一人が行う工程へ変わるのである。それは、かんぱん方式に屋台方式が連結したようなものである。そして、その結果、一人当りの生産性は大幅に Up しているのである。でも、従来の方法で製造するなら海外生産でよい。品質は同じ、コストは低いからである。一人工程は、作業者に作ることへのよろこびを与える。そしてそれは製造物への責任から誇りへと変わっていく。同じものを同時に複数作るなら一人で良いわけである。しかし、ソフトウェアは、どうでしょうか。言わなくても分かると考える。

匠の世界、それはどんな世界なのか。それは、創造するということである。Make でなく、Create。そして、常に、新しい技術へ挑戦を行うことなのである。まさに、プロフェッショナルであるということである。自負、誇り、責任が持てることがプロフェッショナルなのだ。匠とは決して、今まで私たちが考えている職人の域ではないのである。それを超えた技術者なのである。そして、今は、誰もが匠の世界に到達することが可能なのだ。

³ 「思い」この言葉は、IT コーディネータが経営とIT(情報技術)を掛け橋するための大事な言葉。

それは、例えば、エリック・ガンマらのデザインパターンを知ることによって、正しい達人のコードが実装できるということである。

ソフトウェアの世界でのNew 職人とは、Super Programmer である。Super Programmer になるには、自らが、それを目指し、その域に到達し、さらに飛躍し続けるという意識を持っていることである。それは、プロがプロの仕事をするということなのだ。

多くのプログラマは、Super Programmer を目指せばよい。ところが、従来の開発方法論や開発プロセスでは行き詰まってしまう。だから、今、新しい開発方法論・開発プロセスが必要なのだ。でも、すべてがうまくいくものは存在しないのか。本当に、手はないのか。」これでは、暗黒の世界をただ漂うだけになる。

「そうではない。」と言える。何故か、XP が今、私達の前にある。XP って、何かを解決してくれるヒントにならないか。私は、多くのヒントがあるとみる。もし、あなたが何かの光を見出せたとしたら、暗黒の世界に一筋の閃光のように、そう、あなたは Super Programmer なのだ。皆さんは Super Programmer として XP からはじめよう。

0.4 Next

CBD や XP はゴールではなく、通過点である。Next は何にか。その答えは、あなたが出して欲しい。XP や CBD の現状に止まらずに、開発手法を進化させよう。そして、それは、New XP かもしれないし、CBDs+XP かもしれない。臆することなく、勇気を持って、新しい開発手法を見出そう。そうすれば、本当に「銀の弾丸」を手に入れるかもしれない。ソフトウェア開発の「狼人間」はそのとき消滅できるだろう。

(特別寄稿 新保 康夫)

CBD 編

第一章 C B D 概説

1.1 コンポーネントベース開発の定義

1.1.1 背景

最近、「コンポーネント指向」という言葉を良く耳にするが、簡単に述べるとコンポーネントを組み合わせて開発するということである。この背景として、EJB (Enterprise Java Beans) や .NET といったインフラ部分でコンポーネントをベースにしたものが普及をはじめているということにある。そして、次に、システム開発の現場では、企画からカットオーバーまでが例えば3ヶ月という短納期が当たり前という点と、変化に対応できる柔軟さ、そして、企業にとってライバルとの競争性を出すために独自の戦略が求められるということにある。そのためには、高い生産性と品質が求められ、それを解決する方策として挙げられるのが、部品の再利用である。今までも、部品化を行い、再利用を促進する、という運動は、ずいぶん昔から何回もいろいろな部署で試みられているが、うまくいったという話はあまり聞かれない。使われない部品が一杯できているという状況である。なぜ部品化による再利用はうまく行かないのか。そういった疑問に対してのひとつの解としてCBD (Component Based Development) という方法がある。

1.1.2 コンポーネントとは

コンポーネントとは何か。その定義は、ひとつまたは複数のオブジェクトをまとめたものであり、明確なインターフェースを持ち、システムに対して物理的に交換可能な部品として、そのサービスを提供するものである。⁴その目的であるが、利用しているアプリケーションにPlug&Playで機能を追加・交換することである。⁵当然のことではあるが、追加・交換するには、インターフェースが必要である。

1.1.3 コンポーネントベースとは

CBDとは何か。Component Based Development の略であり、コンポーネントをベー

⁴ コンピューター・アソシエイツ社, CAのアプリケーション開発支援製品によるCBD / カタリシス概説, 2001.

⁵ 長瀬嘉秀、今野睦 (監修) Fondatoo Inc. (著), コンポーネントモデリングガイド, ピアソン・エデュケーション, 2001.

スにした開発方法論ということである。「コンポーネントをベースにした」とは、コンポーネントを組み合わせることや交換してシステムを構築することを指している。⁶

1.1.4 コンポーネントで重要なこと

コンポーネントで重要なことは、コンポーネント同士のインタフェースで結合させるということである。このようにすることは、互いの依存性を低くし、コンポーネント自体の独立性を高めるといことが可能になる。そうすることにより、コンポーネントを取り替え可能な部品として扱え、部品を組み合わせたり取り替えたりしてシステムを構築することが可能になるということである。コンポーネント自体は独立性を高くし、インタフェースを明確にすることは、再利用を促進するために重要なことである。

1.1.5 コンポーネントの粒度は大きく

いまして、コンポーネントについて考察することにする。

今まで何故部品化による再利用がうまくいかなかったのか、もしくは、どう考えればうまく行くのかということですが、従来の再利用が失敗するパターンの部品化は、小さな粒度の部品の方だと考える。

それは、これまではサブルーチンやマクロ、関数のライブラリ化、といった小さな粒度でのアプローチが部品化であった。

しかし、ここでは、商品管理や会員管理といった大きな粒度の部品化なら再利用の実現性が高いということを指している。

コンポーネントステレオを例にとりて述べることにする。

コンポーネントステレオという言葉は、メーカー毎にステレオシステムが提供されていたものを、音にうるさいマニアが、例えば、アンプはA社で、スピーカはB社、プレーヤはC社を組み合わせコンポーネントステレオ、という世界を作った。既存のものを組み合わせただけなのに、組み合わせは無数にあるので、組みあがったものは、オリジナリティが発揮される。ここに「コンポーネント」という言葉が使われている。これは、各部品が取換可能ということを現わしている。



図 3 真空管アンプ

⁶ 長瀬 嘉秀、久保 雅恵、カタリス入門、JAVA PRESS Vol.22、技術評論社より

小さな粒度の部品とは、ケーブル類やコンデンサ、増幅器、スピーカ部品を現わしている。

図 3 は、自作真空管アンプの例である。各部品とハンダごてを渡されて、さあ組立てなさいと言われても、全体の構造とそれらの組立て順序を知らないと手の出しようがない。マニアの方は、コツコツ作っていく過程も楽しまれて、出来上がったときには、アンプ 2号というような名前までつける。



図 4 コンポーネントステレオ

一方、図 4 は、コンポーネントステレオの例である。スピーカーシステムとアンプ、DVD プレーヤを用意すれば、ケーブルでつなぐだけですぐに使えるものである。この時の前提としては、ケーブルとプラグのインターフェースが一致していることと、各部品としては完成された機能を持っており、テスト・検査済みという点である。この場合は、スピーカだけを取り替えることは簡単である。これがコンポーネントステレオというものである。

例えば、DVD プレーヤを買いに行ったとする。その時、店員に「テレビに D1 端子がありますか」と聞かれた。それがあるとより映像がきれいになるということである。これは、インターフェースによって提供されるサービスが変わるという例である。インターフェースとそれに対応するサービスを明確にするということが重要というわけである。

インターネットでオーディオステレオを検索すると、このコンポーネントステレオであれば家電メーカーのサイトで検索できるが、図 3 のような自作アンプとなると、オーディオマニアの世界に入らないと探せない。佐藤さんのスピーカとか、田中さんのアンプといったようにである。

ところが、システム開発の現場では、逆転するわけである。手作りが普通であり、コンポーネントによる開発はこれからというところである。

アプリケーションを例にとってコンポーネントについて述べる。

例えば、BtoC のマーケットプレイスの開発をする場合、決済方法として、代引き・銀行振込・郵便振替・コンビニ決済・クレジット決済等多種多様である。

それを一から作り出すにはかなりの時間とノウハウが必要になる。でも、決済部分はコンポーネントを利用するということを決めれば、そこについてはあれこれと考えずに、どのようなデータをどのように渡せば良いかを考えることになる。

これがコンポーネントによる開発であり、生産性と品質を上げ、かつ短納期に応えるための方法である。

もうひとつの考え方として、マーケットプレイスを構築するパッケージを利用した方が

もっと早いのではという案がある。先ほどの「コンポーネントステレオ」に見るように、ステレオをセットごと買うのでは、独自性がだせない。

そこで、必要と思われるコンポーネントを組み合わせることでステレオシステムを組み上げるスタイルにすると、既存のアンプやスピーカを使うのに、それを組み合わせるだけで、自分だけのステレオができ、独自性を持てる。これが、競争力を持てるシステムを構築する秘訣なのである。この方法によってこそ企業の個性を出すのである。図 3 では、様々な部品を組み合わせているが、部品自身は独立してサービスを提供していない。前述したコンポーネントの定義からわかるとおり図 4 のように、コンポーネントは独立性が高く、交換可能であることである。そのためには、コンポーネントは大きな粒度で考える方がよいということである。

1.2 コンポーネント開発プロセス

1.2.1 CBDの手法のひとつ

CBDでは、どのようなプロセスとモデリングフェーズで開発を進めるかだが、CBDのひとつとしてカタリシスという手法がある⁷。カタリシスは、コンポーネントモデリング手法と言われ、図 5 は 1 例であるが、UML を拡張した表記を使い、コンポーネントの構造と操作（インタフェース）を表現する。

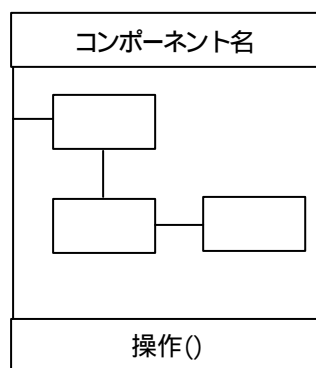


図 5 カタリシス表記例

1.2.2 開発プロセス

図 6 は、カタリシスを例に開発プロセスとモデリングフェーズを表している。

⁷ 提唱者は Alan Wills, Desmond D'Souza である。著書として、Objects, Components, and Frameworks with UML: The Catalysis Approach (Addison-Wesley) がある。

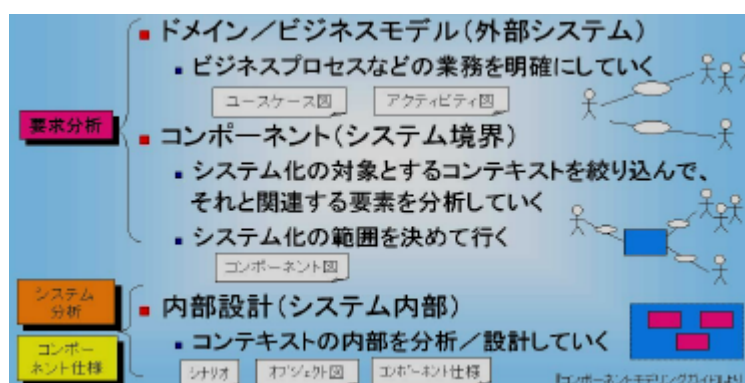


図 6 開発プロセス

左側の色のついた箱が開発プロセスを表す。まず、要求分析を行い、次にシステム分析を行う。そして、コンポーネント仕様を行う。このプロセスに対応して、右側にあるのがモデリングフェーズである。

開発は、まず要求分析から入る。最初に、ドメインモデリング、ビジネスモデリングを行う。そこでの目的は、「ビジネスプロセスなどの業務を明確にしていく」ということである。

1.2.3 要求分析

ステップとしては、まず、ユーザインタビューを行い、外部からのアクションを洗い出し、ドメインの用語の定義を行う。ビジネスフローを作成して、ビジネスのプロセスを把握する。アクタや他システムといった外部とのやりとりや、ビジネスの流れと範囲を明確にするために、UMLのユースケース図やアクティビティ図のようなイメージのものをラフにスケッチする。ユーザと共通イメージを持つことが目的であるので、きっちりUMLを描くというよりは、外部とのやりとりや流れがわかるものであれば形式は問わないということである。

ここでは、システムの境界を明確にするために、アクションをもとにシステムの範囲(スコープ)を決定する。そして、スコープ内で、分類可能なものをビジネスコンポーネントとしていくつかのかたまりに分けるということである。

このプロセスでは、この時点でコンポーネントが現れるということだ。実装を意識せず、上位レベルでコンポーネントに分割してしまうのである。

1.2.4 要求分析のポイント

そのために、次のような視点で考える。大事なことは、機能面に焦点をあてないことであり、ビジネスプロセスの視点からとらえたビジネスコンポーネントを適切に分割する

ことに焦点をあて、エンティティを中心に考えるのではなく、具体的な業務の流れやプログラムの動きを想像することが大切なのである。

ここまでで要求分析は終わる。次は、システム分析になる。

1.2.5 システム分析

モデリングフェーズとしては、内部設計にあたる。スコープとコンポーネントを見出したので、図 6 のように、その範囲に着目し、システム内部のモデル化を行う

ここでは、まず、シナリオを集める。シナリオとは、ある瞬間をとらえたスナップショットと言える。例えば、セミナーの申し込みをする場合、「受講者がセミナーを申し込む」といった抽象的な表現ではなく、「田中さんが、3月23日のCBDひとかじりのセミナーに申し込むためにメールを出した。メールを受けた担当者は、空席状況を確認し、空席の場合は、田中さんを受講者として登録し、その旨のメールを発信する」というような具体的なアクションである。

また、シナリオはセミナーをキャンセルするとか、いくつもあり、それを多く集めてモデルの品質を上げる。そこから、オブジェクト図を作成し、オブジェクト間の関連と属性を明確にする。コラボレーション図を使うなどして、オブジェクト間の相互関係と接続関係が明確にする。

最後のプロセスは、コンポーネント仕様である。システム分析で把握したオブジェクト間の関係により、UMLのクラス図のように、内部モデルを作成し、そこに対して、外部からの操作、インタフェースを設けて、コンポーネント仕様を作成する。

この段階で、分析レベルのモデルが導出されたことになる。これを繰り返し行うことで精度をさらに上げていくことになる。

この先は、内部設計としての実装設計になる。

第二章 CBDs 実践例

2.1 実践のテーマ

2.1.1 課題の設定

本章では、CBDsによる実践について述べることにする。

実際にアプリケーションを構築しなければならないが、何を作るかということである。大切なのは、自らが顧客とならなければならないということである。このため、ソフトウェアを開発するということをビジネスと捉えて、CRCカード管理システムをCBDsの手法を使って開発することにした。

2.1.2 CRCカードとは

最初に CRC カードとは何かについて知る必要がある。CRC カードとは Class Responsibility Collaborators カードの略であり、責任駆動型のオブジェクト抽出の手法である。

もう少し詳細に述べると、CRC カードをオブジェクトにみたと、クラスを識別し、責務を各クラスに記述し、継承関係を決定し、属性を定義していく手法である。

このとき注意することは、責務を明確にすることであり、操作を明確にすることである。

クラス名:	
責務:	協調者:

図 7 CRC カードイメージ

CRC カードに記述することにより、そのクラスは、どんな役割を担い、どのような操作で使われるかということが分かるのである。

2.2 実践記

2.2.1 Getting Started

CRC カード管理システムを構築することにしたが、「何からはじめるか」からも模索する状態であった。開発プロセスの順番どおりに進めることとした。

まずは、要求分析から行う。そこで問題、要求分析って何をするのかである。ここは、オブジェクト指向分析の王道であるビジネスを把握するためにアクティビティ図を作成することにする。

アクティビティ図を書く目的

目的

- 顧客のビジネスプロセスに関する全容を把握するため
- アクティビティ図を書くことの意味
 - どこまでをシステムの範囲としているかが明確になる
 - 役割を担う物が明確になる
 - 役割が明確にされれば、それがコンポーネントとなる

2.2.2 ビジネスプロセスを知る

まずは、CRC カードを使った開発を前提として、あるシステムを企画してリリースするまでのビジネスプロセスをアクティビティ図で表現することにする。

ところが、メンバーでは行き詰まってしまったのである。その理由は以下の要因である。

- どこまで書けば良いのか。
- メンバー個々のイメージがあっていない。
- アクティビティ図を書きだすと、みんなのイメージが違っている。

これを解決するには、ビジネスプロセスとは何かを理解し、その粒度を同じすることである。あまりにも当然のことであるが、実際に作図するとそのことを忘れてしまうのである。

前述のアクティビティを書く目的を理解していれば容易に気がつくのだが、実際上はかなり難しいようである。これは、ビジネスプロセスについての正しい知識を十分に知っているか、もしくは、描けるセンスを養っているかである。

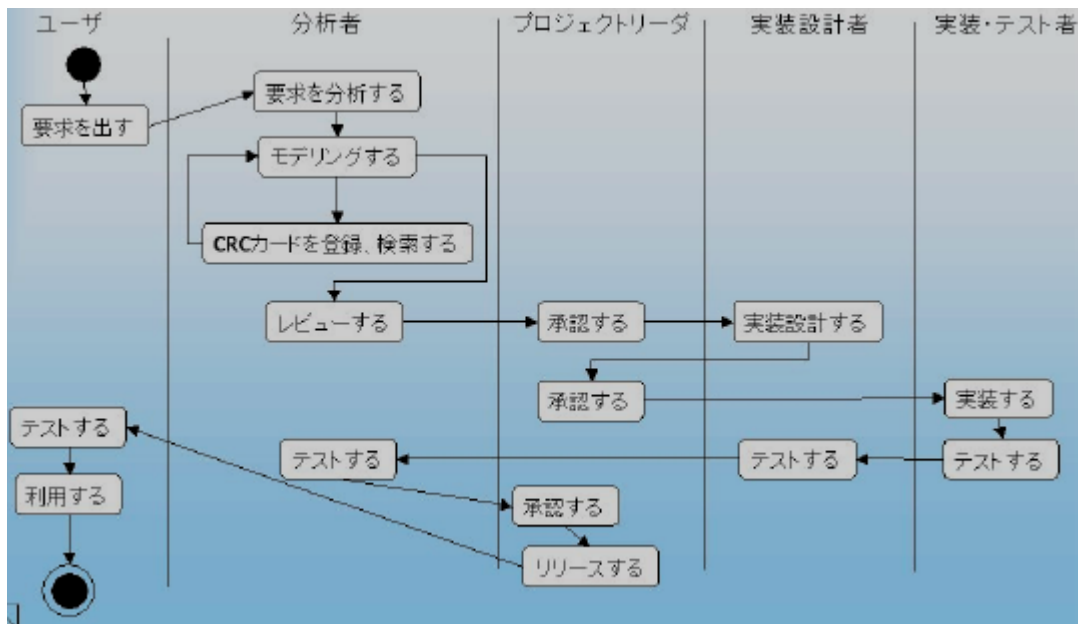


図 8 アクティビティ図

2.2.3 ユースケースを作成する

アクティビティ図が完成できたならば、次に、システムとして受け持つ範囲明確にしなければならない。CRCカード管理システムの役割を割り出すにはどうすればよいのか。これを見出すために、ユースケースで考えることにする。

ユースケースによって顧客がシステムに期待しているサービスを明らかにするためである。このため、ユースケース導出において次の点に着目して行った。

- アクタとの関わりを明確にする。
- システムの境界を明確にする。
- シナリオからアクタと機能の関係を明確にする。

この着目で注意しないとイケないのは、「機能」と言う言葉である。初心者はこの「機能」と言う言葉を従来の設計手法の「機能」と同じと考えるのである。ここでは「振る舞い」と考えるべきであろう。実は、ここから間違いの第一歩を踏み出すのである。

アクティビティ図からシステム境界を見出し、ユースケースダイアグラムを作成する。

ドメインモデリング

- シナリオから名詞と動詞を切出す
- それらの関係を明確にする
- 重複するものはまとめる

実は、このときに第二のミスをしてしまっている。ドメインモデリングをしていることである。アクティビティ図で描かれたアクティビティがユースケースになる。こうしておかなければ一貫性が保たれないのである。従来手法ではひとつのフェーズがすめば言葉は変わっても良いが(実は、これがソフトウェアでの大きな問題なのである)、オブジェクト指向開発では変えてはいけないところである。そのため、ドメインモデリングはもう少し遅らせて行つ方が最初は良い結果になると考える。

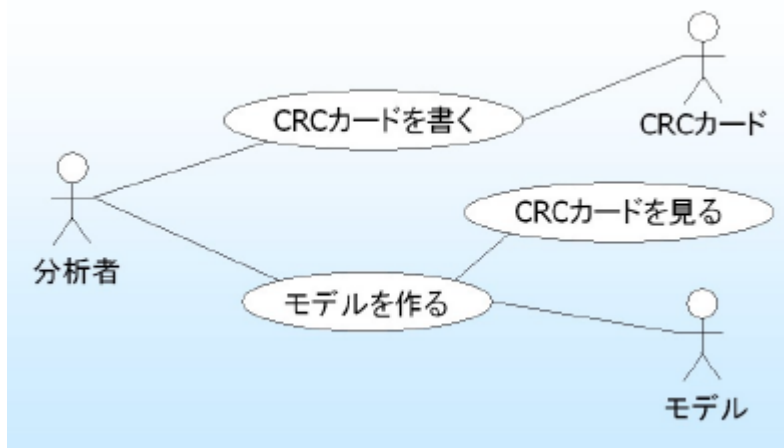


図 9 ユースケース例

図 9 のユースケースの例である。この時点でアクティビティ図と合っていないのが理解できる人は CBDs の達人への道を一步踏み込んだことになる。

そして、そこから機能に着目をしたのである。

- CRCカードを登録する
- CRCカードを検索する
- CRCカードを変更する
- CRCカードを削除する

図 10 ユースケースを機能で見てしまった例

2.2.4 コンポーネントを見つける

図 10 では、何かおかしい点がある。それは、ユースケースの名称がないということである。ここで重要なのは「振る舞い」として捉えることである。そうすると「CRCカードを記録する」もしくは「CRCカードフォルダを管理する」というサブユースケースが出てくるはずである。これが、コンポーネントとなるのである。この発想の根底には、「作るな、使え」というコンポーネントベース開発の基本思想をいかに正しく理解しているかが大切

である。

しかし、「振る舞い」を見出すのであれば、多くのユースケースを作成するのではなく、コンポーネントを直接見出し、表現する方が良い。そのため、図 11 のようなラフなコンポーネントモデルを作成する。

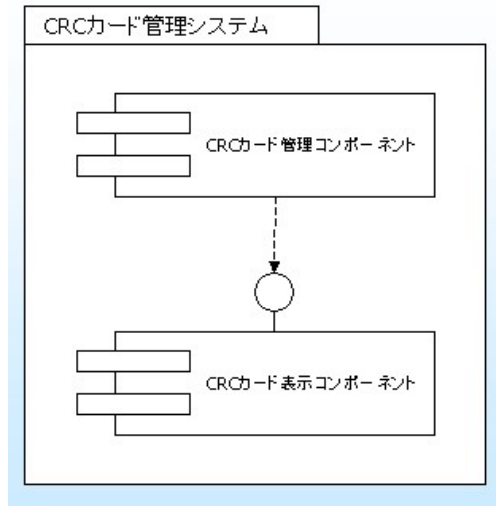


図 11 コンポーネントモデルのラフスケッチ

今回のシステムのレベルであれば、カタリシス手法の全てを駆使して行う必要はない。その手法の表記に従ったコンポーネント図を描けば十分である。このことは、カタリシス手法をどの程度適用させるのかという判断が必要であることを示している。この判断ができれば、CBDs の達人と言える。

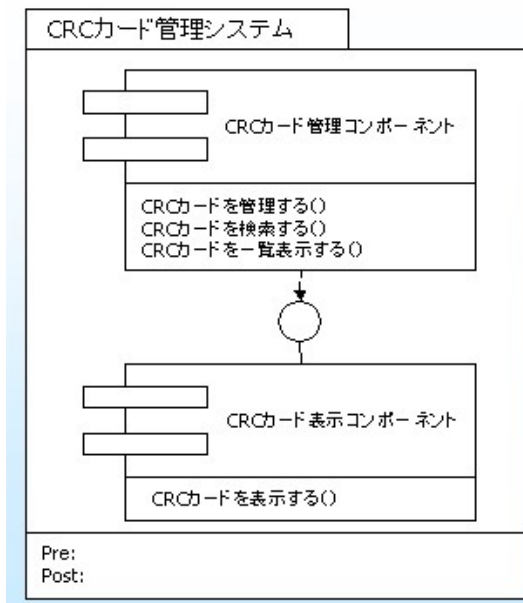


図 12 コンポーネント図

残念ながら、今回の実装はここまでである。ここからは一気に実装モデルの設計へとなる。この次に暗黒の世界に再び引き込まれるのは、いきなりアーキテクチャの世界を考慮した実装モデルへとワープするところである。この暗黒の世界から脱出する一筋の閃光をうまく見出せるかがポイントであろう。

2.3 まとめ

今回の実践からは以下のことが言える。

システムを分析するときは、最初は機能に着目せず、ビジネスフローに着目することである。そして、ユースケース駆動をうまく活かすためには、まずは、機能に着目するのではなく、振る舞いに着目する。

そうすることにより、上流でコンポーネントに分けることができるので、各コンポーネントに関して、設計を進めることができる。例えば、上流で分けたコンポーネントが、そのままEJB等のコンポーネントに対応させることができるということである。

重要なことは、コンポーネントの定義とインタフェースである。このため、コンポーネントの中身の設計は後に回して、コンポーネントの振る舞いとそのインタフェースをきっちり記述することが大切である。

実は、このことこそ再利用を行うための重要な鍵となる。

今回のCRCカード管理システムで言えば、CRCカード管理コンポーネントは既にあることを前提として考えることである。このことにより、機能ではなく、インタフェースに着目することになる。そこから、インタフェースを記述することが重要であることにたどり着くのである。

コンポーネントは取換可能を意味している。つまり、インタフェースさえ一致していれば別のコンポーネントを適用できるということである。

コンポーネントベース開発の極意は、「作るな、使え」である。

XP 編

第三章 XP 概説

3.1 XP って何

XP とらと多くの方が、まずは WindowsXP を思い描くのである。しかし、XP という言葉は、そのときすでに、エクストリーム・プログラミングとして使われていたのである。それもアメリカではソフトウェアエンジニアリングの世界にとってメジャーな言葉として扱われていたのだ。

もし、あなたがソフトウェア開発者や情報処理技術者に属する人であれば、中身はともかくとしてXPを知らないとは言えないだろう。ましてや、WindowsXP とは、それは自らを技術者として否定することであり、他の多くの技術者への冒涇となるからだ。

手厳しい言葉からはいったが、XPの中身は知らない方のために、簡単に述べることにする。

XP 入門の書籍から見ると以下のことが挙げられる。⁸

- 「ライト級」
- 「優れている」
- 「低リスク」
- 「融通が利く」
- 「予期可能」
- 「科学的」
- 「楽しく行なえる」ソフトウェア開発方法！？

また、「Embrace Change！」である「変化を受け入れる、抱擁する。」も言える。⁹

別の言い方をすれば、アジャイル宣言で言われる「私たちは、プロセスとツールよりも個人と対話に、包括的なドキュメントよりも動くソフトウェアに、契約交渉よりも顧客との協調に、計画に沿うことよりも変化に対応することに、価値をおく。」も言える。¹⁰

どうやら、従来とは視点が異なったソフトウェア開発手法のようである。

3.2 XPのコンセプト

XPについては、以下の3つのコンセプトがある。

⁸ XP エクストリーム・プログラミング入門～ソフトウェア開発の究極の手法」より

⁹ 平鍋健児, <http://objectclub.esm.co.jp/eXtremeProgramming/> 言葉を引用。

¹⁰ 原文 <http://AgileManifesto.org/>

和訳 <http://objectclub.esm.co.jp/eXtremeProgramming/WhatXPPresentsUs/> より

- 4つの変数
- 4つの価値
- 12のプラクティス

ここでは、それぞれの項目についての説明は割愛するが、どのような項目があるかを列挙することにする。

3.2.1 4つの変数

4つの変数は、以下のとおりである。

- コスト(予算)
- 時間(納期)
- 品質
- スコープ(開発範囲)

3.2.2 4つの価値

4つの価値は、以下のとおりである。

- コミュニケーション
- フィードバック
- シンプル
- 勇気

3.2.3 12のプラクティス

12のプラクティスは、以下のとおりである。¹¹

- 計画ゲーム
- 短期リリース
- メタファ
- シンプルな設計
- テスティング
- リファクタリング
- ペアプログラミング
- 共同所有

¹¹現在は、開放的作業スペース、日次スキーマ移行の2つが増え、14のプラクティスになっている。(出典:月刊 JavaWorld 2001.9)より

超ビギナー入門 CBDs&XP

- 継続した結合
- 40時間労働
- オンサイトの顧客
- コーディング規約

個々の項目について詳しく知りたい人は、是非、書籍を購入して勉強して欲しい。

3.3 開発プロセス

XP における開発プロセスとはいかなるものかを以下に簡単に示す。

XP では、短いイテレーションを繰り返し、こまめにリリースを実施する。イテレーションのサイズは、1～3週間で同じサイズとし、リリースサイズは、1～3ヶ月とする。複数のイテレーションが集まって1つのリリースとなる。図 13 は、それを表したものである。

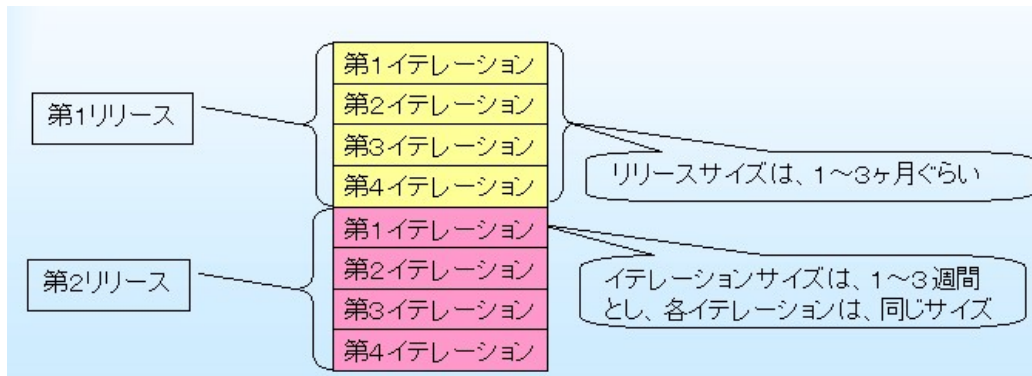


図 13 イテレーションとリリース

これを理解した上で、実際に進める開発プロセスの例を図 14 で示す。

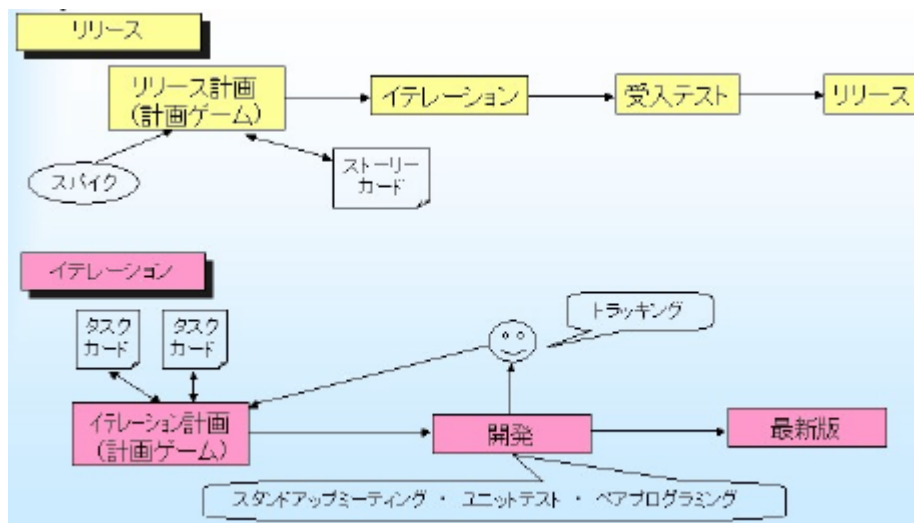


図 14 XP 開発プロセス

このように、XPの開発プロセス自体は複雑なものではなく、むしろシンプルなものと言える。

3.4 XP研究会～よもやま話～

XPの概説はこの辺にして、XP研究会のよもやま話としよう。ひょっとするとこの方がXPを知るのには近道かもしれない。

3.4.1 XPに興味を持った理由～O氏の発言～

O氏曰く、「XPに興味を持った利用は、顧客と一緒にHappyになれるって良いじゃない。ソフトウェア変更コストが抑えられるってことは顧客ともめないし、しかも、家族と過ごす時間を増やせる?のだから。それに、新しい手法ってところが気を引くね。そうそう何か楽しそうだしね。」。

3.4.2 XP? WindowsXP?

どうも、XPを知っているという人は少ない。実態は、社内では認知度ゼロって言える。ましてや、「12のプラクティス」を唱えれば、声がかえってこない。最初は大きな声で言っていたが、社内では、だんだん小声になる。そしてなにやらあやしいXP教という団体に間違われている。「あやしい団体は私達ではない。あなた達だ。」言っても気が付かない。最早、末期的症状。XPの達人曰く、「井の中の蛙は捨てさり 大海に出でて多くの幸とコラボレーションするなり。さすれば、おのずと道は開かれる。」。やっぱり宗教かもしれない。

3.4.3 12のプラクティスを勉強しよう。

XPって、魅力てきだね。シンプルな設計、今動けば良い (YAGNI: You're NOT gonna need it!)、まずテスト、こまめなリリースって言うからね。ところで、これってどういう意味なの。まずは、12のプラクティスを勉強しよう。

3.4.4 シンプル設計って、何?

シンプルな設計って、良いとは思うね。今必要なものが、今動けば良いということだし、スリムなコードになれば良いのだからね。でも、注意しないと「とにかく動きさえすれば良い」とはXPは言っていないよ。じゃどうするの。リファクタリングがヒントかもしれない。

3.4.5 リファクタリング

実装してから設計改善するのがリファクタリングでしょう。」

「そうよ。」

「何かしっくりこないですね。」

「プログラムを実装してから、こうすれば良かったってことあるよね。そしてその方がわかり易くて、シンプルならそうした方が良いよね。」

「でも、そう思うだけでしょ。」

「XPは、それを実践するからすごいと言えるね。そして、ソースに対して、コメントを書くよりも、コメントが不要なコードに変更すべきだとも言っている。」

.....

もうひとつのよもやま話。リファクタリングを辞書で引くと「因数分解」と書かれている。ちょっと、笑えると思う。しかし、何か真理をついているのではないか。

3.4.6 テストファースト

若手は言った。

「テストを書くってどういうことですか？」

「画面があるのに、わざわざどうして？」

O氏曰く、

「テスト熱中症を読んでみたら¹²、それとツールも用意されているのだから。」

「ツールなんかあるのですか。」

「JUnit¹³で言って、Java, VB, C++, VC などのツールが用意されているのだよ。」

3.4.7 ペアプログラミング

「作業効率は2倍になれば、うれしいね。」

(達人同士のペアプログラミングでね。開発のライフサイクルで見ればね。)

「これだけでも是非やりたい。」

(一番魅力あり。)

「この業界は長時間労働が当たり前になっている。」

¹²Kent Beck 著、小野剛 訳、JUnit テスト熱中症：プログラマは、テストを書くのが好きになる、
<http://objectclub.esm.co.jp/eXtremeProgramming/TestInfected-J.html> より

¹³Unit Test/ ツールへのリンク集 <http://objectclub.esm.co.jp/eXtremeProgramming/>

(いつも、早く帰りたいよね。)

「コードの共同所有の第一歩はペアプログラミングから。」

(「このプログラム、Oさんしかわかりません。」って言われなくてすむ。)

3.4.8 プラクティスの支え合い

プラクティスって相互に関連しあっているのだ。

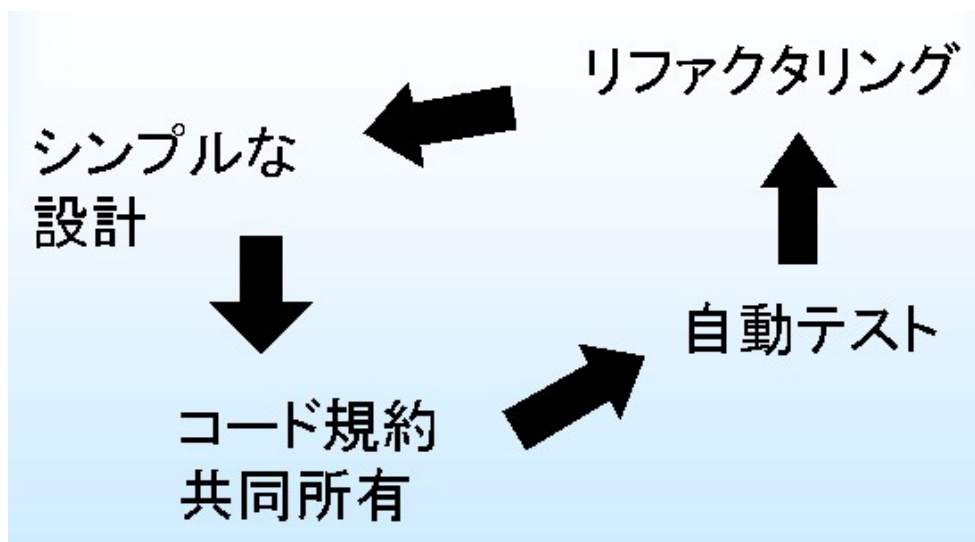


図 15 プラクティスの支え合い

3.4.9 ところで、JUnitで

「JUnitで何。」

勝手にテストしてくれるみたい。」

「ほんまかいな。」

成功したら、緑。失敗したら、赤。」

「それって、本当に役に立つの。」

.....

そして、JUnitに関する英語の文献を読み漁る。

JUnit Primer¹⁴、JUnit CookBook¹⁵の2つに絞ってトライする。

¹⁴ <http://junit.sourceforge.net/doc/cookstour/cookstour.htm> よし

¹⁵ <http://junit.sourceforge.net/doc/cookbook/cookbook.htm> よし

「テストングフレームワーク」を知り、「デザインパターン」の威力を知る。
でも、疑問も出てきた。

画面系やDB系のテストはどうするの。」

サーバサイドJavaのテストはどうするの。」

画面は、HttpUnit¹⁶で、DB系は、JUnit¹⁷だけでか、Mock Object¹⁸で、そして、サーブレットは Cactus(Jakarta)¹⁹で行える。

よもやま話はここで幕を閉じる。

¹⁶ <http://httpunit.sourceforge.net/> 参照。

¹⁷ <http://junit.sourceforge.net/> 参照。

¹⁸ Mock Object については、「XP エキストリームプログラミング実践記～開発現場からのレポート」に紹介されている。

¹⁹ <http://jakarta.apache.org/cactus/> 参照。

第四章 半かじり実践記

4.1 さあ、実践

4.1.1 何を作るか

まずは、オンサイトユーザが必要と考えるが、メンバー構成上、ユーザとなるのが難しい。このため、自分たちがユーザとなるものとした。本来的にはシステム化してはダメであるが、実践するテーマとしてはちょうど良い例と考えた。

それは、ストーリーカードのシステムを作成するというものである。

4.1.3 ストーリーカード管理システム

ストーリーカード管理システムのイメージを図 16 に示す。

ID	ストーリーカード名	優先順位	担当者	備考
1	2002-02-01	1	山田 太郎	開発準備が整った
2	2002-02-01	2	山田 太郎	ストーリーカード管理システムの開発が完了した
3	2002-02-01	3	山田 太郎	ストーリーカード管理システムのテストが完了した

図 16 ストーリーカード管理システム

4.2 壁との戦い

4.2.1 ストーリーカードを書く

さあ、はじめよう。まずは、自分が顧客になったと想定し、ストーリーカードを書いてみる。早くも第一の壁にぶちあたる。その壁とは、書き方がわからない、なかなか書けない、どれぐらいの粒度で書けばよいのかという基本的なことがわからない。

でも、とりあえず書いてみて、みんなでレビューする。なんとかストーリーカードになり次に、ストーリーカードに優先順位をつける。

今回は、ストーリーカードを紙のカードに書くことにした。ここで、紙がこんなに便利なのかと思ひ知らされる結果となる。

Customer Story and Task Card			
DATE:	作成日	TYPE OF ACTIVITY: NEW or FIX or ENHANCE	FUNC. TEST 機能テスト
STORY NUMBER:	ストーリー	PRIORITY: USER or TECH	
PRIOR REFERENCE:	細紙分割	RISK: リスク分割	TECH ESTIMATE: 見積り
TASK DESCRIPTION:	タスクの内容を記述		
NOTES:	<div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; width: 30%;"> A: それなしでは機能しない B: 本質ではないが多くの ビジネス価値を含む。 C: あればよいもの。 </div> <div style="border: 1px solid black; padding: 5px; width: 30%;"> A: 正確に見積もれる B: 合理的には見積もれる C: まったく見積もれない </div> <div style="border: 1px solid black; padding: 5px; width: 30%;"> 理想的なエンジニアリング時間だ！ </div> </div>		
TASK TRACKING:	予期していたことに対しての測定値 何日費やしているか、あと何日必要か)		
Date	Status	To Do	Comments

図 17 ストーリーカードの例²⁰

4.2.2 タスクカードに分割

ストーリーカードの次にタスクカードに分割することになる。ここで、第二の壁にぶちあたる。ストーリーとタスクってどう違うのかわからない。ここで、とてつもなく大きな壁であることを発見する。タスクに落とせるってことは、すでに頭の中にラフなスケッチでクラス図やシステム動作イメージを持っている必要があるということだ。

どうにかタスクカードを作成し、まずは、直感での見積もりを行い、サインアップとする。実は、ここでのラフなスケッチが本当に頭の中で描くことができたかを確認する必要があった。これは、あとで大きな壁となってあらわれた。

²⁰ ケント・ベック著 XP エクストリーム・プログラミング入門~ソフトウェア開発の究極の手法より。

4.2.4 第3の壁、そして第4の壁

タスクカードをレビューし、これでスタートすることにした。そこで、第3の壁が待ち構えていた。タスクカードからどうやって実装すればよいのか。いままで知っていたオブジェクト指向開発の分析 設計手法は役立たない。本当にオブジェクト指向なのか。不安をかきたてる。ユースケース図は書かないの。クラス設計ってどこですの。XPの設計フェーズってどうするの。暗中模索の中、怪しく進めることとなる。そして、懸念が発覚したのだ。チーム間での概念クラス図が合っていない。

概念クラス図を合わせることにする。そして、第4の壁が大きくのしかかってきた。

すべてが悪夢のようである。

「どれをクラスにするのか。」

「えっ、それってわかっていたことじゃないの。」

「ところで、ストーリーカードクラスの役割は。」

「何を、今さら。」

「登録メソッドは管理クラスに持たすの。」

「おい、おい、おい……。」

要するに、モデリング能力が無いことを痛感させられる結果となる。まさに、モデリング技術の重要性について身をもって知るはめとなる。

4.2.5 再度、クラス図を書く

概念クラス図から再作成することにする。はやくも、リファクタリングとなる。

概念モデルを再度描いて、再確認することになる。

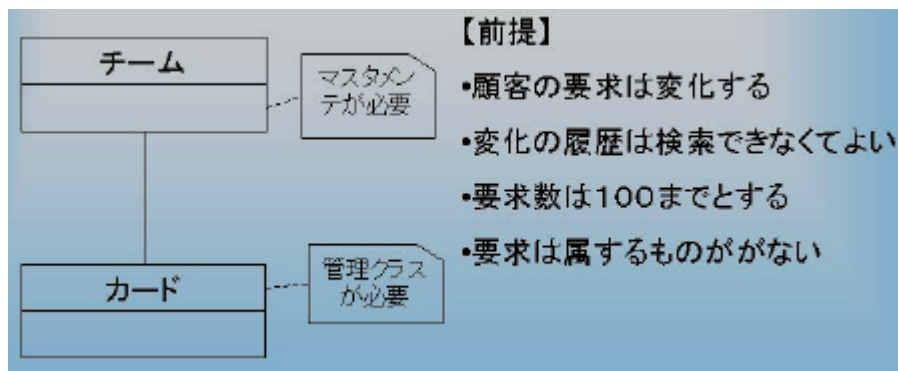


図 20 概念モデル

実は、これではメタファにならないのだ。これは、オブジェクト指向分析 開発を知っているものにとっては信じられないことである。

本来は、XP では邪道とされているアーキテクチャ設計まで踏み込まないとメタファにならないため、設計のクラス図をモデルとした。

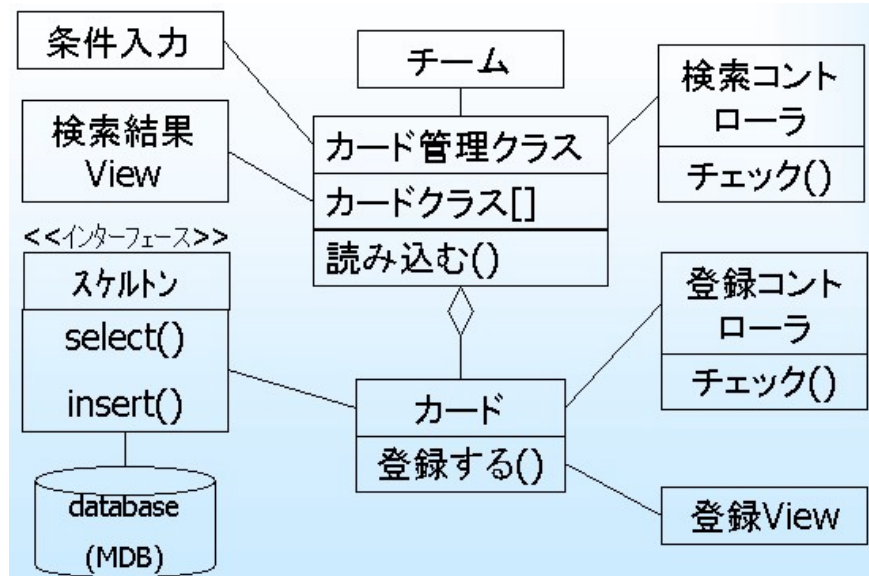


図 21 アーキテクチャ設計モデル

これは、MVC(Model-View-Controller)フレームワークを知っていれば、メタファができるのである。ここまできると、オブジェクト指向技術が根底にあることを思い知らされる。

4.2.6 まずは、テストから～実践したプラクティス

XPを行うのであったら、テストファーストの実践を試みたかった。まずは、テストファーストの実践を行う。もちろんペアプログラミングで実装する。JUnit を使ってテストを実践、緑が出た。テストは成功だ。

ここまでの道のりはかなり厳しいものであった。すべてのプラクティスを実践したかったが、図22の結果となっている。

■ 計画ゲーム	(○)	■ ペアプログラミング	(○)
■ 短期リリース	(×)	■ 共同所有	(×)
■ メタファ	(×)	■ 継続した結合	(×)
■ シンプルな設計	(×)	■ 40時間労働	(?)
■ テスティング	(○)	■ オンサイトの顧客	(?)
■ リファクタリング	(×)	■ コーディング規約	(×)

(○)は、今回実践できたプラクティス

図 22 実践できたプラクティス

4.3 まとめ

4.3.1 XP への険しい道

現行の開発スタイルは、オープン系ではスパイラルもあるが、ウォーター・フォールがまだまだ基本となっている。作業中心となり、納品用ドキュメントを作ることが開発の最終工程となっていることもある。なによりも、仕様変更は悪というイメージが強い。まずは、開発計画書をきちんと書くことからしないといけない。

一方、今、開発現場にXPを実施にはためらいもある。これは、XPの効果を実例もまだまだ少ないことがある。そのような現状では、上司や顧客、開発チームメンバが受け入れてくれる可能性は低い。まずは、上司への説得が課題となる。

XPのプラクティスであるテストファーストについても未経験者から見れば、手間がかかりそうに見え、短納期では避けたいと考えられてしまう。本当は、短納期だからこそ必要であるのだが、ハードルが高く見えるのである。これは、オブジェクト指向を前提としていることが起因していると考えられる。そのわりには、XPのライトウェイトなイメージが先行しているように見られる。もっと、地に足をつけて挑むものであり、しっかりとした足取りであれば素晴らしい効果は期待できるものである。

4.3.2 XPを広めるために～プロジェクトご活用しよう～

従来手法しか知らない人にとっては、XPの説明は、反発や議論を招く結果となりやすい。これは、彼らの無知さをあまりにもストレートに露呈させてしまうからであろう。

このためには、実践をして実績を作ることが大切であるが、まず、最初はXPと言わずに、ペアプログラミング、短期リリース、そしてリファクタリングを勧めるのが良いと考える。また、プロジェクトについては、コミュニケーションの重視を適用し、スタンドアップミーティングやタスクカードを利用していくのも良い。

また、プロジェクトについて、それぞれの視点で見ればXPにおけるプラクティスは非常に有用である。

プロジェクトマネージャから見れば、共同所有と標準化は例えば要員のローテーションを行うことができる。さらに、コミュニケーションが活性化される効用としてメンバーがコードは個人ではなくチームの所有物と言う意識を持つことになる。そして、やらされているのではなくて自らがやっているというモチベーションの向上をさせることができるのである。

プロジェクトリーダーから見れば、標準化、リファクタリング、そして、ペアプログラミングは引継ぎ容易にしてくれる。テストの自動化、YAGNIそして、リファクタリングは修正

のコスト削減してくれる。ペアプログラミングは一人で悩むことによる時間コストを低減させる。テストの自動化は品質向上に役立ってくれる。

このように、個々のプラクティスを導入することはメリットがある。XP に対する偏見や抵抗勢力が強いところでは、使えるプラクティスから導入し、時間が経つと、実はXP を実践していましたということになればよいであろう。

4.3.3 今振り返って、そして今後の課題

今、振り返って見ると以下のことが言える。

- がんばってみます。」から やります。」へ
- 紙やホワイトボードは役立つ。
- やっぱりコミュニケーションは大切。
- 意外と出来るものである。
- 共通語 (UML、デザインパターン等)が必要である。

そして、今後の課題としては、受託契約でのXPが実践できる環境へ持っていくことと、ケント・ベックと同じようにはできないことを痛切にかんじたので自分にあったXPの手法を確立することである。

最後に、私の思いを込めて。

始めてみませんか。観客ではなく、実践者に。新しいIT世紀にふさわしく。」

付録

A 活動報告

A.1 NCS CBDs研究会

A.1.2 NCS CBDs 研究会とは

NCS CBDs 研究会は、コンポーネントを利用した開発方法に興味を持ったメンバーで構成されており、CBD の実践を体験してみる場となっている。

A.1.3 NCS CBDs 研究会の目的

コンポーネントベース開発の方法論を知見し、語れるソフトウェア開発者になることをゴールとしている。

A.1.4 活動経過

NCS CBDs 研究会の活動経過は以下のようになっている。

- 2001年4月に発足。
- 4月から6月までは Component Based Development and Object Modeling を教材とし、翻訳しながらCBD について理解を深める。
- 10月から3月は、CRC カードの管理システムを題材として、CBD を実践。

A.1.5 今後の活動

内部設計を行い、実装フェーズを体験する。

A.2 NCS XP 研究会

A.2.1 NCS XP 研究会とは

NCS XP 研究会は、XP (エクストリームプログラミング)に興味を持った、NCS 社内有志の集まりであり XP の実践を体験する場である。

A.2.2 NCS XP 研究会の目的

オブジェクト指向の最新技術で、リーディングソフトウェア開発者になろうということを目的としている。

A.2.3 活動経過

NCS XP 研究会の活動経過は以下のようになっている。

- 2001 年 4 月に発足。
- 4 月から 6 月は準備勉強の期間として、XP の 12 のプラクティスやデザインパターンについて理解する。(12のプラクティスについては、社内ホームページにて成果を公表。)
- 6 月から 7 月は、エクストリームプログラミングテスト技法の本の執筆に参画。(この本は、昨年 11 月に翔泳社より出版。)
- 7 月から 9 月は、実践のための準備勉強を実施。
- 10 月から、ストーリーカード管理システムを題材とした、実践を行う

A.2.4 今後の活動

実践中であるシステムの 1 回目のリリースを実施することと、2 回目のイテレーションにおいて、機能強化やリファクタリングを行う。

B 参考文献

1. Fondatao Inc. 著, 長瀬 嘉秀、今野 睦 監修, コンポーネントモデリングガイド, ピアソン・エデュケーション, 2001.
2. Doug Rosenberg and Kendall Scott, *Use Case Driven Object Modeling with UML: A Practical Approach*, Addison Wesley, 1999.
邦訳) 株式会社テクノロジックアート 訳、長瀬 嘉秀、今野 睦 監訳, ユースケース入門, ピアソン・エデュケーション, 2001.
3. 長瀬 嘉秀、久保 雅恵 著, カタリシス入門, JAVA PRESS Vol.22, 技術評論社, 2002.
4. CAのアプリケーション開発支援製品によるCBD / カタリシス概説, コンピュータ・アソシエイツ株式会社, 2001.
5. Kent Beck, *Extreme Programming Explained : Embrace Change*, Addison Wesley Publishing Company, 1999.
邦訳) 長瀬嘉秀 監訳、永田 渉、飯塚麻理香 訳, XP エクストリーム・プログラミング入門~ソフトウェア開発の究極の手法, ピアソン・エデュケーション, 2000.
6. Kent Beck, Martin Fowler, *Planning Extreme Programming*, Addison Wesley Publishing Company, 2000.
邦訳) 長瀬嘉秀 監訳, 飯塚麻理香 訳, XP エクストリーム・プログラミング実行計画, ピアソン・エデュケーション, 2001.
7. James W. Newkirk, Robert C. Martin, *Extreme Programming in Practice*, Addison Wesley Publishing Company, 2001.
邦訳) 比嘉 康雄、平鍋 健児、高嶋 優子 訳, XP エクストリーム・プログラミング実践記~開発現場からのレポート, ピアソン・エデュケーション, 2001.
8. 長瀬 嘉秀 監修, 日本 XP ユーザグループ 著, eXtremeProgramming テスト技法~xUnit ではじまる実践 XP プログラミング~, 翔泳社, 2001.

あとがき

ちょうど一年前に「これらの研究会をしませんか」と私から声をかけたところ、NCS社内からメンバーが集まってくれた。私が幹事役をするのではなく、メンバーの中から幹事役を選び、私自身は顧問という立場で行うこととした。それは、メンバーが自らの力で進んで欲しいという願いがあったからだ。

NHKで放送されている「プロジェクトX」という番組を知っていると言う人は多いと思う。なぜ、この番組を例にあげたかと言うとこの研究会へ参加したメンバーが「プロジェクトX」の番組のように何年後かに評価されてほしいという私の「思い」を言いたかったからである。

研究会に参加したメンバーもまた「プロジェクトX」である。そして、メンバーは今、小さな第一歩を踏み出した。でも、それは大きな価値を見出す一歩であろう。

今書かれている内容については、ちょうど誰もがぶち当たる最初の難関であり、そして、どうしても乗り越えなければならない難関なのである。このことを、体験した言葉で書くことにより、これから遭遇するであろう多くの人達の乗り越えなければならないハードルに対して、少しはお役に立つと言える。同じ苦しみをいつも誰かが繰り返しては良い技術は伝わっていかないからだ。

最後に、私が、研究会のメンバーに問いかけている「思い」である言葉を書きたいと思う。この言葉は、ベストセラーとなった「ザ・ゴール」から引用したものである。

「あなたは何に変化しますか。どう変化しますか。」

多くのソフトウェア開発者に。

平成 14 年 4 月 10 日

新保 康夫

超ビギナー入門CBDs & XP

初版 平成 14 年 4 月 10 日 発行

著者：NCS CBDs 研究会・NCS XP 研究会

高森 正延

岡田 将

大中 敏行

東 成樹

山中 美智子

(特別寄稿・監修

新保 康夫)

@非売品

本書は、いかなる方法やいかなる理由においても無断でその一部または全てを複写、流用もしくは転載を禁ずる。

Copyright © 2002 By NCS CBDs 研究会・NCS XP 研究会





NCS 日本コンピューター・システム株式会社