UMLとオブジェクト指向と開発プロセスについて

本テキストは、2003年にMCT社で行われた
「UML　勉強会　第9回」での資料を改訂したものです。

ソフトウエア開発でのUML活用法

講師 池田公平

なぜUMLが作られたのか

UMLが作られた経緯をおさらいします。

UMLを活用するにあたり、

なぜUMLが必要になったかを知り、

UMLを使う理由を理解しましょう。

意外に浅いUMLの歴史

C++ができたのは1983年

Javaができたのが1994年

UML1.0は1997年

UML2.0は2004年
UML2.1は2007年

UMLはまだまだ発展途上である！

方法論の統合からUMLが誕生

90年代初頭に、

BoochさんのBooch法（モデリング言語）、
JacobsonさんのOOSE法（ユースケース分析）、
RumbauthさんのOMT法（方法論）、

などが入り乱れ、
オブジェクト指向の方法論や表現法の統一が
なかなか達成できない状況になる。
コマッタ…

とりあえず書式だけでも決めよう!

ということで合意した3人が、

UMLという統一表記法を作り出しました。

めでたし！　めでたし！

抜け殻になったUML

統合に難儀していた方法論については、
UMLに盛り込まれていません。
UMLはあくまでも表現方法の統一を行っただけです。

したがって、UMLの知識だけでは
実践で役に立ちません！

UMLって大したこと無い？
覚えるのは時間の無駄でしょうか？

UMLはすべての方法論の基礎

肝心な方法論を除外してしまったUMLですが、
オブジェクト指向における分析、
設計手法の基本的なところは押さえてあります。

UMLは多くの方法論を表現し、
実践するためのベースとなっています。

さらに、クライアント、設計者、開発者の各所で共通の表現手段を持つとうことは、プロジェクトをスムーズに進行するために、非常に重要です。

さまざまな開発手法や方法論を学ぶにあたって、
UMLの知識は必須です。
つまり、UMLは最初の1歩なのです。

UMLはそれほど複雑でありませんし、
最初にすべてを知る必要もありません。

UMLと開発手法のコンビが必要

目的を達成するために、多くの開発手法があります。

UMLは方法論（開発手法）を含んでいません。

UMLは多くの実践的な開発手法との
コンビネーションで威力を発揮します。

開発手法とはどのようなものがあるでしょうか？

主な開発手法

ウォーターフォール型

ソフトウエア工学

反復型

UP(Unified Process)

ドメイン駆動設計(Domain Driven Design)

テスト駆動型開発（TDD)

アジャイル型（反復型の一種）

アジャイルソフトウエア開発

SCRUM

XP(エクストリームプログラミング)

その他

ペアプログラミング

ソフトウエア職人気質

ウオーターフォール型開発手法とは

古典的なソフトウエア工学に基づいている。
「要求」→「仕様」→「分析」→「設計」→「実装」→「テスト」
という手順を踏み、各ステージを完全に消化してから次のステージに進む。原則的に逆もどりはしない。

長所

- 達成率の管理が容易
- 完璧なドキュメントが残る
- 製品の品質を向上できる

短所

- 開発途中での仕様変更ができない(設計からやりなおし)
- 思わぬ障害がでたときに破綻する(リスク管理が脆弱)
- 工数見積もりを誤ると大幅に遅延する(遅延なしは奇跡に近い)

反復型の開発手法とは

1日から6週間ぐらいの短いサイクルでプロジェクトを区切り、そのサイクルを繰り返してプロジェクトを進めていく手法。

長所

仕様変更に柔軟に対応できる
早い時期にリスクを減らせる
プロジェクトの見通しが良い

短所

設計、開発の要求スキルが高い

デスマーチ

プロジェクトにとって、最悪の事態は
「デスマーチ(日本語訳:八甲田山)」に陥ることです。

デスマーチとは、仕様バグ、実装バグが入り乱れ、1つバグを取ると2つのバグが発生し、昨日と今日では仕様が異なり、開発が収束する気配がまったくないといった泥沼の状態です。
（よく聞きますね）

こうなると、スケジュールは大幅に遅れ、予算は超過し、クライアントは怒り狂い、プロジェクトの責任者は更迭されるかクビになることでしょう。

デスマーチにならないために

細かなリスク管理
仕様変更に対応する柔軟性
正確な工数見積もり
↓
プロトタイプによる検証(リスク駆動型)
定期的なリファクタリングと仕様の見直し
優れた開発手法により進捗の透過性を確保

もはやWaterfall型では不可能です！

某プロジェクトの成功事例の紹介

RUP(Rational Unified Process)を導入

ユースケースによるアスペクト指向開発

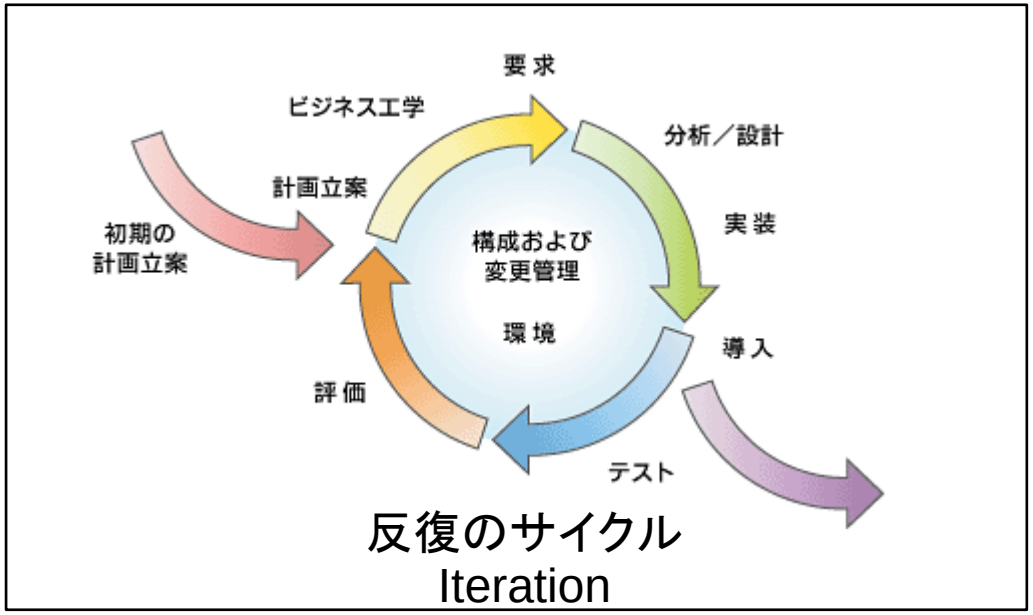
IJCによるコンサルティング

RUPとは

- ✓ Rational社が開発した開発手法。
- ✓ Unified Processをベースに拡張。
- ✓ 最新のフィーチャーを取りれている。

- ✓ 反復型の開発
- ✓ ユースケース駆動
- ✓ リスク駆動
- ✓ 品質管理

反復の内容



反復のサイクル
Iteration

方向付け
Inception

→

推敲(すいこう)
Elaboration

→

作成
Construction

→

移行
Transition

iteration　I1　E1　E2　C1　C2　C3　T1　T2

△ ライフサイクル目標のマイルストーン

△ ライフサイクルアーキテクチャのマイルストーン

△ 初期運用能力のマイルストーン

△ 製品リリースのマイルストーン

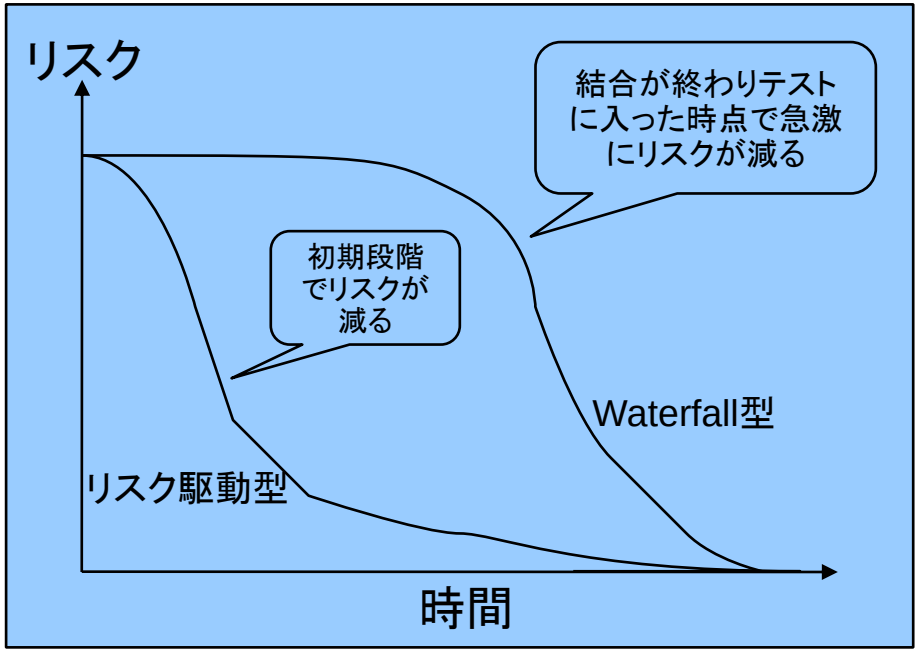
リスク駆動型

リスクの種類

（1）要求リスク　システムに対する要求に関する誤解など
（2）技術リスク　安定性、設計技術、運用と干渉、製品サポート、将来性など
（3）要員リスク　適切な人材の確保や教育、チームワークや適性など
（4）政治リスク　プロジェクトの利害関係者の圧力や対立からの干渉など

リスクの大きなものから積極的に解決していく、リスク駆動型の開発。

プロジェクトの初期段階で大きなリスクは減少する。



アスペクト指向とは

オブジェクト指向に取って代わるものではない

オブジェクト指向で表現しにくいものを実現

ユースケースとの相性は良い

言語レベルでのサポートはこれから

アスペクト指向でできること

すべてのコンポーネント、クラス、メソッドに、
横断的に作用する機能を実現する。

- セキュリティ管理
- ログ機能
- メモリ管理
- ファイル操作の最適化
- 例外処理
- 広範囲にわたる横断的な拡張

などを抽象化して設計できます。

ユースケース駆動

要求仕様や実装方法など、

さまざまな問題を

ユースケースを使って分析します。

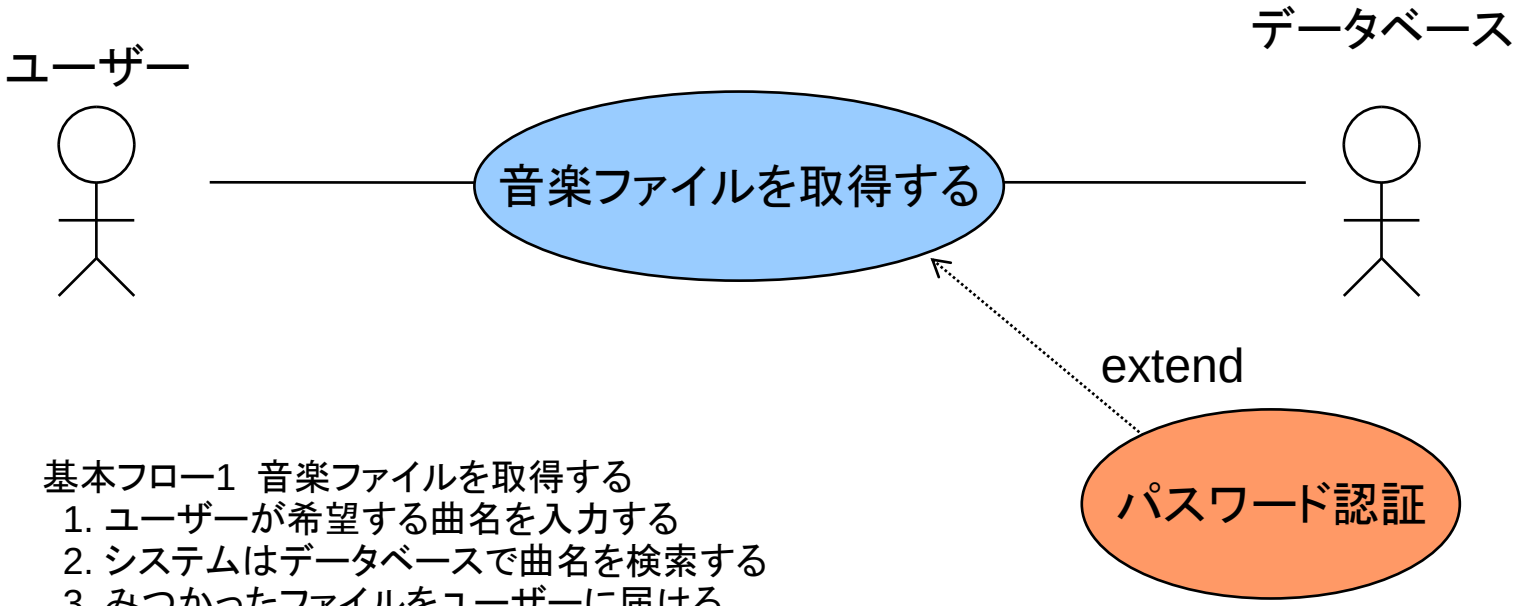
ユースケース分析

最初に、必要最小限なデザインを行います。
(Minimal design)

それに基づき、基本的なフローを作成します。

たとえば、Webでの音楽データ配信サービスを
例に考えて見ます。

音楽配信システムのユースケース分析



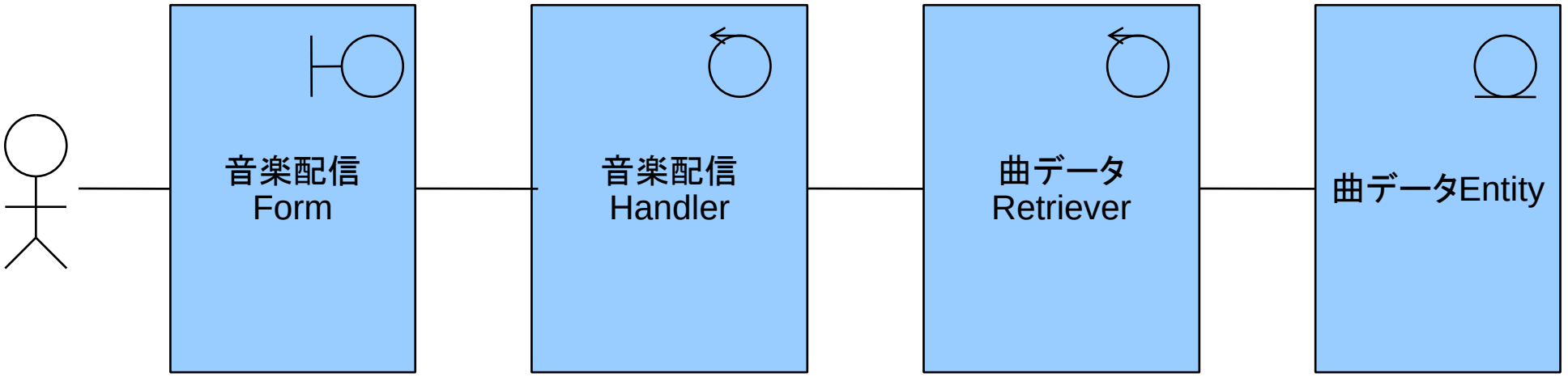
基本フロー1　音楽ファイルを取得する
1. ユーザーが希望する曲名を入力する
2. システムはデータベースで曲名を検索する
3. みつかったファイルをユーザーに届ける

代替フロー1　曲名が見つからない
1. システムはエラーメッセージをユーザーに示す

拡張フロー1　ユーザーIDの認証
1. システムはユーザーにIDとパスワードを問い合わせる
2.ユーザーはIDとパスワードをシステムに通知する
3. IDとパスワードを検証し合格したら次のステップへ進む

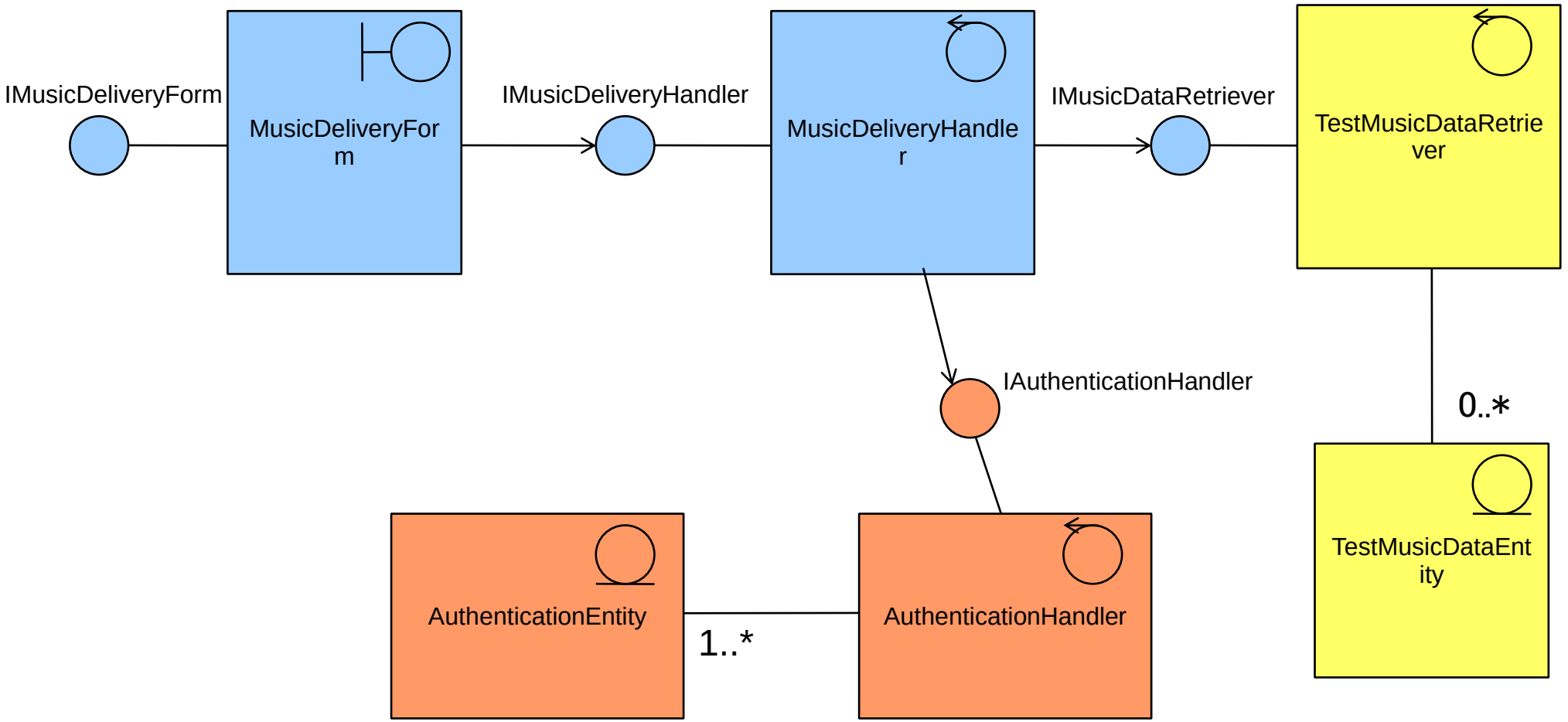
代替フロー2　IDとパスワードが不一致
1. システムはエラーメッセージをユーザーに示す
2. システムは再度IDとパスワードの入力を促す

音楽配信システムの分析モデル



※ UML図はPowerPointで書きやすくするために簡略化しています。

音楽配信システムのデザインモデル



基本フローをまず作ってみる

ここで重要なのは、まず基本フローのみ動作するプログラムを作ってみることです。

エラー処理、パスワード認証、ネットワークのアクセスは後回しで構いません。

動作するものを作ることにより、原理的な検証とクライアントが望むイメージを確認することができます。

代替部、拡張部を順次作っていく

基本部分の検証が終えたら、順次エラー処理や代替部、拡張部を作っていきます。

先ほどのユースケースには、ネットワークアクセスについては記載されていません。ここでは、ネットワークのことは考えずに、ローカルのファイルにアクセスできればよいのです。

どんどん拡張していくと出来上がり

最初のユースケースには無かった、ネットワークアクセス、課金システム、ユーザー情報管理システムなどのユースケースを追加してゆき、システムを徐々に拡大していきます。

ここでも、基本フローの検証から代替、拡張という手順を踏みます。

すると、いつのまにか立派なネットワーク音楽配信システムが出来上がっているのです。

柔軟な拡張性が肝心

このような、基本機能を拡張しながら作成していく手法は、拡張性に優れたプログラムでなければ実現できません。

オブジェクト指向に則ってきちんと書かれたプログラムならば、拡張は容易なはずです。

オブジェクト指向に基づいた設計は、UMLでの記述が適している。

拡張性と柔軟性に優れたプログラムを作るには、設計が大切です。設計の段階で、オブジェクトの抽象化をしっかりと行うことが重要です。

UMLによるクラス設計図は、洗練された抽象化オブジェクトや綺麗なプログラムを設計し、表現するのに大いに役立つことでしょう。

IJCによるコンサルティング

IJCとは、Iver Jacobson Consulting の略で、オブジェクト指向のスリーアミーゴの一人、イバー・ヤコブソン博士が経営するソフトウエア開発のコンサルティング会社です。

残念ながら、日本での営業活動は行っていませんが、私は幸運なことにIJCの全面的なコンサルティングを受けたプロジェクトを経験し、その後IJCのスタッフの一員としてお手伝いする機会がありました。

ヤコブソンさんが講演などでいつも最後に言っている言葉です。
意味は皆さんで考えてみてください。

Good enough is better than best!

参考文献

ユースケースによるアスペクト指向ソフトウエア開発
イヴァー・ヤコブソン／パンウェイ・ング著
ISBN4-7981-0896-0

独習UML第3版
㈱テクノロジックアート著
ISBN4-7981-0763-8

ラショナル統一プロセス入門
フィリップ・クルーシュテン著
(出版元　ピアソンエドケーション）