UMTP MF 2014

DDDをじっくり語る60分

グロースエクスパートナーズ（株）
ITアーキテクト 和智　右桂

JavaEE勉強会 所属

Yukei Wachi

グロースエクスパートナーズ株式会社 勤務

和智　右桂

ネコ好き

IT アーキテクト

@digitalsoul0124

Digital Romanticism

http://d.hatena.ne.jp/digitalsoul

時々翻訳をしています


ソフトウェア開発の実践
エリック・エヴァンスの
ドメイン駆動設計
ソフトウェアの核心にある複雑さに立ち向かう
Domain-Driven Design:
Tackling Complexity
in the Heart of Software


Continuous Delivery
継続的
デリバリー
信頼できるソフトウェアリリースのための
ビルド・テスト・デプロイメントの自動化
Jez Humble, David Farley 著
いつまで手動で
デプロイして
いるんですか?
ASCII


実践テスト駆動開発
テストに導かれて
オブジェクト指向ソフトウェアを育てる
Steve Freeman・Nat Pryor 著
SE


組織パターン
チームの成長により
アジャイルソフトウェア開発の変革を促す
James O. Coplien・Neil B. Harrison 著
「プロセスが答えではないのなら、
どこに答えがあるのだろう?」
SE

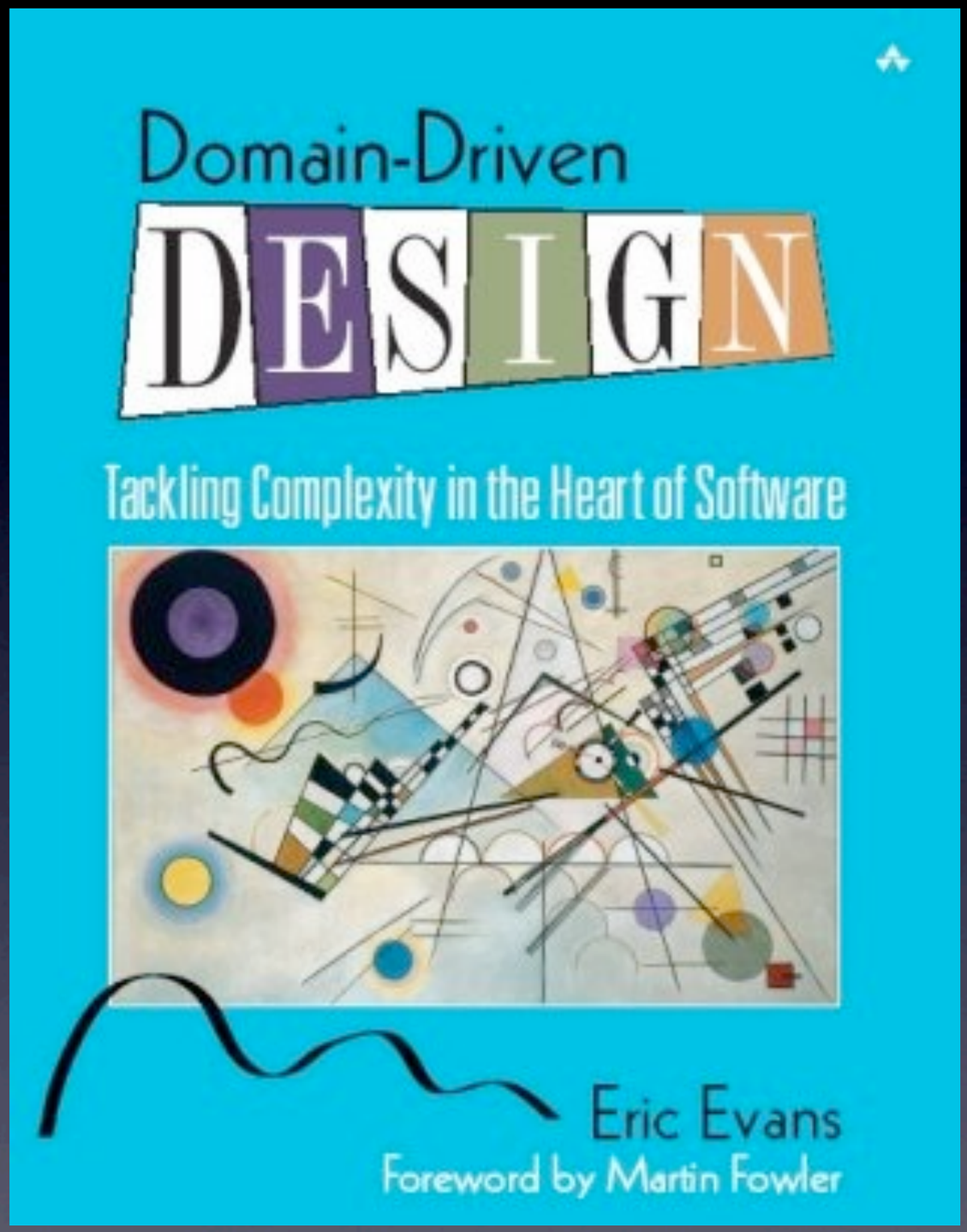
普段の開発経験を元に、
第三部、第四部の実践について
考えていきます。

アジェンダ

- DDDとは
- モデリングとは
- 大規模開発の課題
- ドメイン分割とコンウェイの法則
- コアドメインとしなやかな設計
- まとめ

DDDとは

Domain Driven Design



Eric Evans

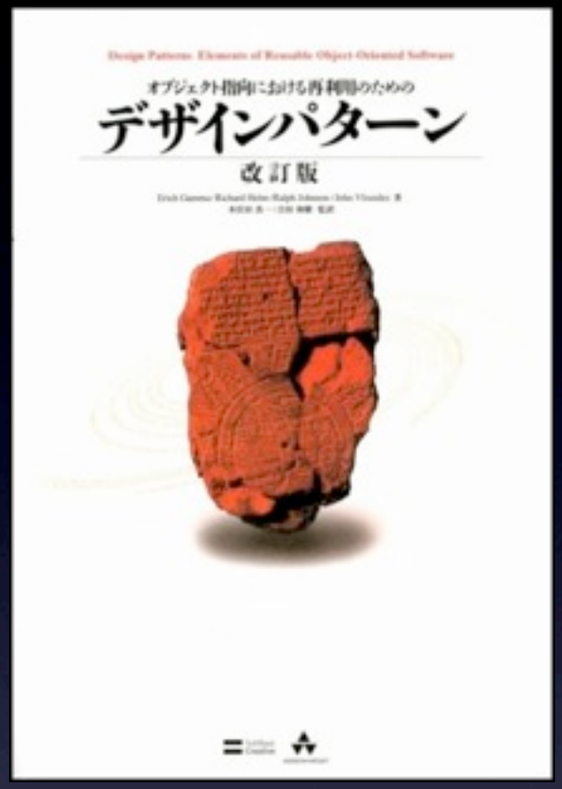
Eric Evans http://www.flickr.com/photos/chrstopher/1447594745/ by Chrstopher

出版は2003年

- 2001年　Windows XP
- 2002年　J2SE 1.4 リリース
- 2003年　Spring Framework リリース
- 2004年　Oracle 10g リリース
- 2005年　StrutsがApacheトップレベル
  プロジェクトに昇格

Chronology http://www.flickr.com/photos/elsie/4607687530/ by Elsie esq.

DDDの主な参考文献


オブジェクト指向における再利用のための
デザインパターン
改訂版

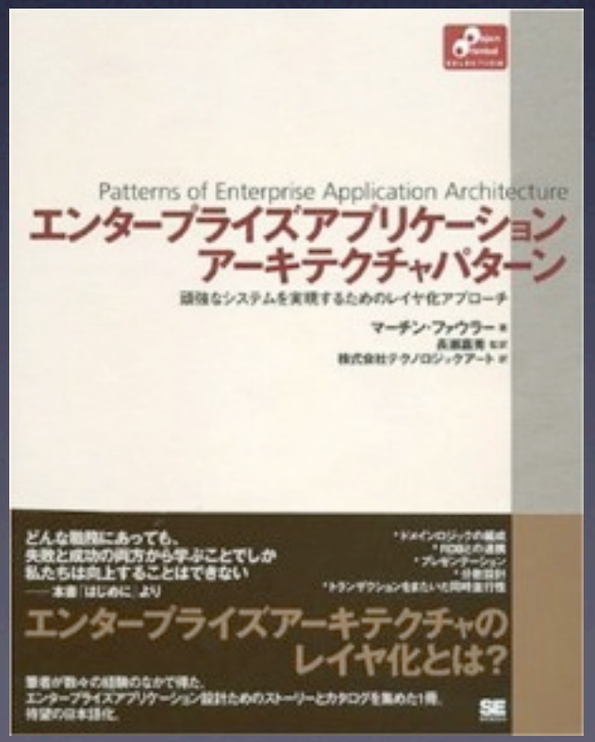

ソフトウェア
アーキテクチャ

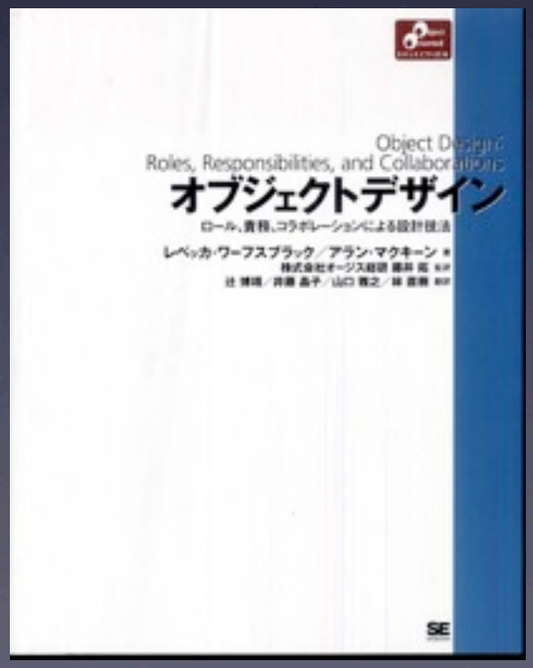

アナリシスパターン
再利用可能なオブジェクトモデル
Analysis Patterns:Reusable Object Models
マーチン・ファウラー 著
BOOCH
JACOBSON
RUMBAUGH


リファクタリング
プログラミングの
体質改善テクニック
Refactoring : Improving The Design of Existing Code
マーチン・ファウラー 著
児玉 公信　友野 晶夫　平澤 章　梅澤 真史 訳
BOOCH
JACOBSON
RUMBAUGH
ピアソン・エデュケーション


実践UML
第3版
オブジェクト指向分析設計と反復型開発入門


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
eXtreme
Programming
explained
ケント・ベック 著
長瀬嘉秀 監訳／永田渉・飯塚麻理香 訳
ピアソン・エデュケーション


Patterns of Enterprise Application Architecture
エンタープライズアプリケーション
アーキテクチャパターン
マーチン・ファウラー 著
どんな組織にあっても、
失敗と成功の例方から学ぶことでしか
私たちは向上することはできない
エンタープライズアーキテクチャの
レイヤ化とは?
SE


Object Design
Roles, Responsibilities, and Collaborations
オブジェクトデザイン
ロール、責務、コラボレーションによる設計技法
SE

エッセンスは？

顧客の**仕事**を理解すること

Domain

View At Work http://www.flickr.com/photos/slightlynorth/2127922888/ by Slightlynorth

顧客の言葉で理解すること

Language

モデルを共有すること

Model

hands wikipedia aussiegall http://www.flickr.com/photos/nojhan/3204073130/ by nojhan

モデルを基に

Model Driven Development

ソフトウェアを作ること

モデリングとは

モデリングとは？

3D Character and Question Mark http://www.flickr.com/photos/crystaljingsr/3914729343/ by 姒儿喵喵

“モデルとは、
知識の表象である”

1979

– Trygve Reenskaug

Tree of Knowledge http://www.flickr.com/photos/knilram/64366434/ by Knilram

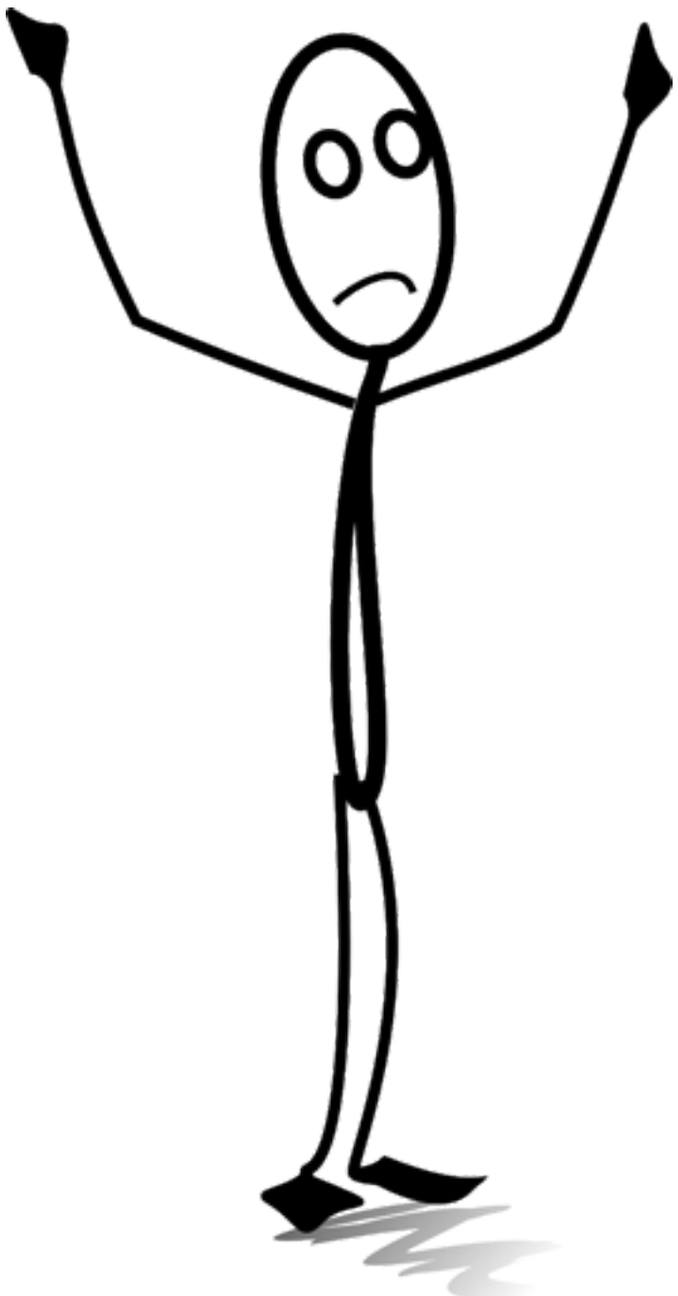
モデリングとは、

知識に基づく

表現の構造化・立体化

である

そう言われましても



お題

新年会幹事の仕事を
新人に教える

手続き型

・お店を選ぶ
・メンバーの出欠を確認する
・お店を予約する
・当日の乾杯指名 / 一本締め
・お金を集める
・支払いをする

モデル型

企画 → 遂行 → 会計

予算感

ジョウゲカンケイ

傾斜

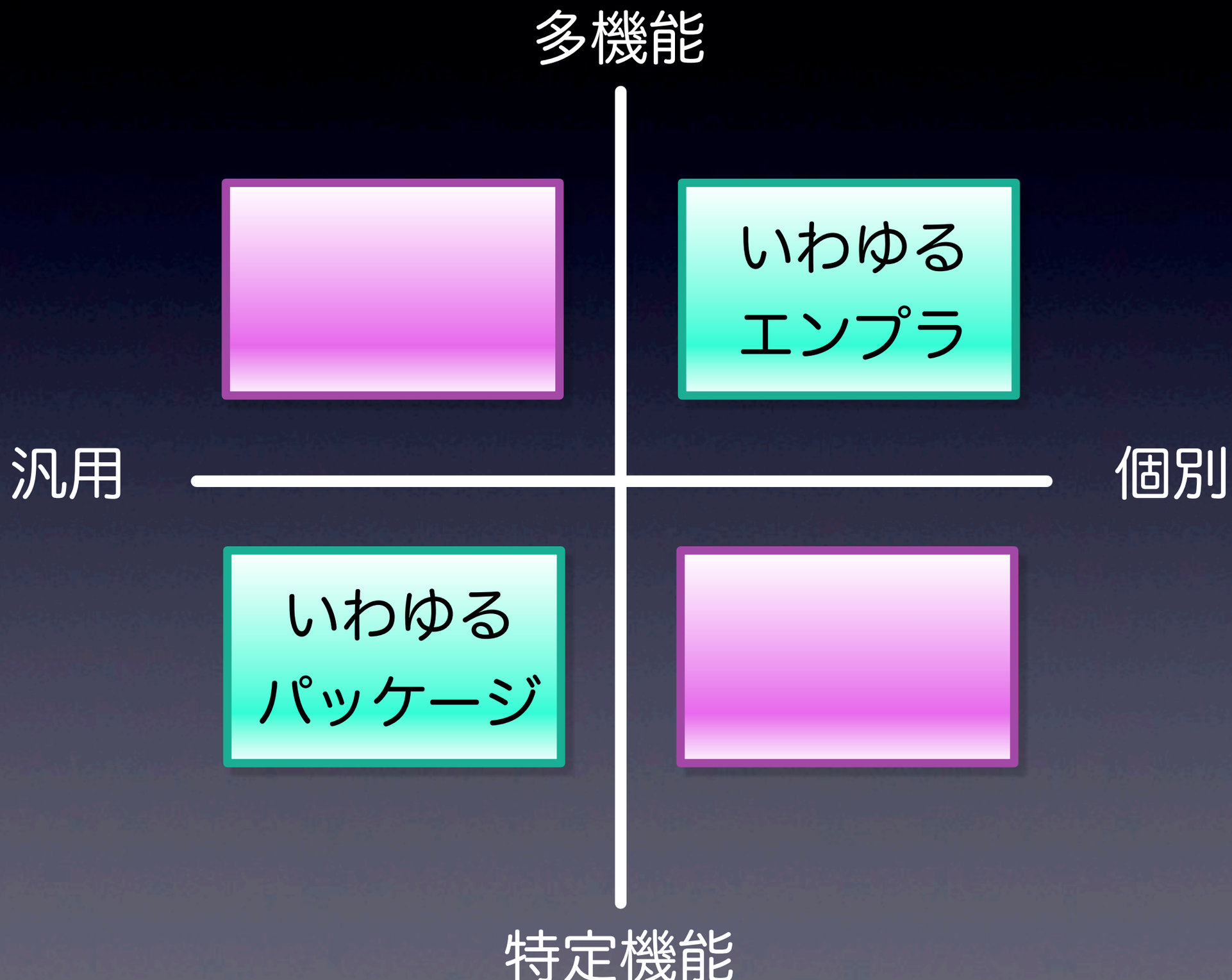
補助金

新年会幹事業務の“モデリング”

システム開発にとって
モデリングとは？

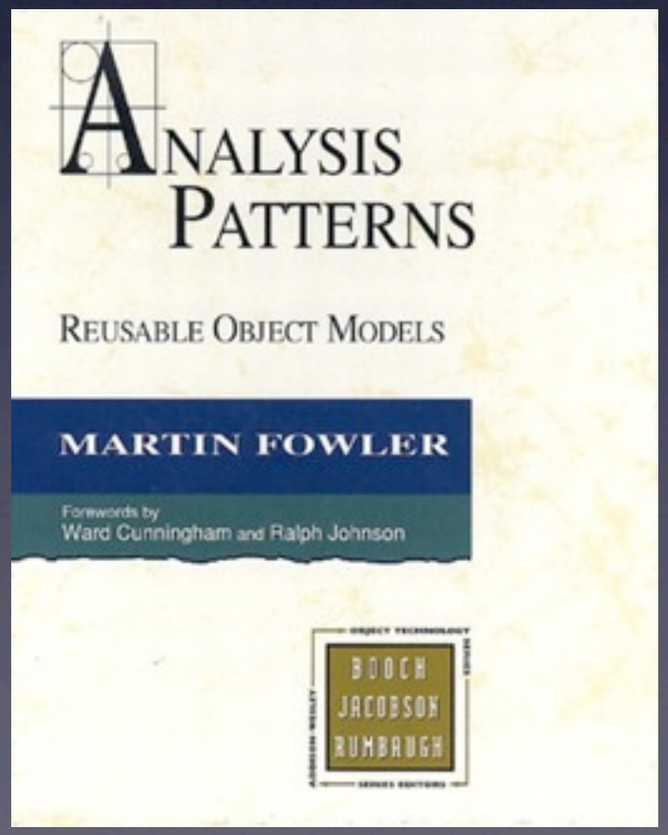
3D Character and Question Mark http://www.flickr.com/photos/crystaljingsr/3914729343/ by 姒儿喵喵

システムの性質による



パッケージ

- 汎用化は抽象化によって実現
- Adaptiveなモデル
- 必然的に難易度が高くなる



エンタープライズ

- 具体的でやっかいな業務
- 機能間難易度の濃淡が激しい

本発表のテーマ

大規模開発の課題

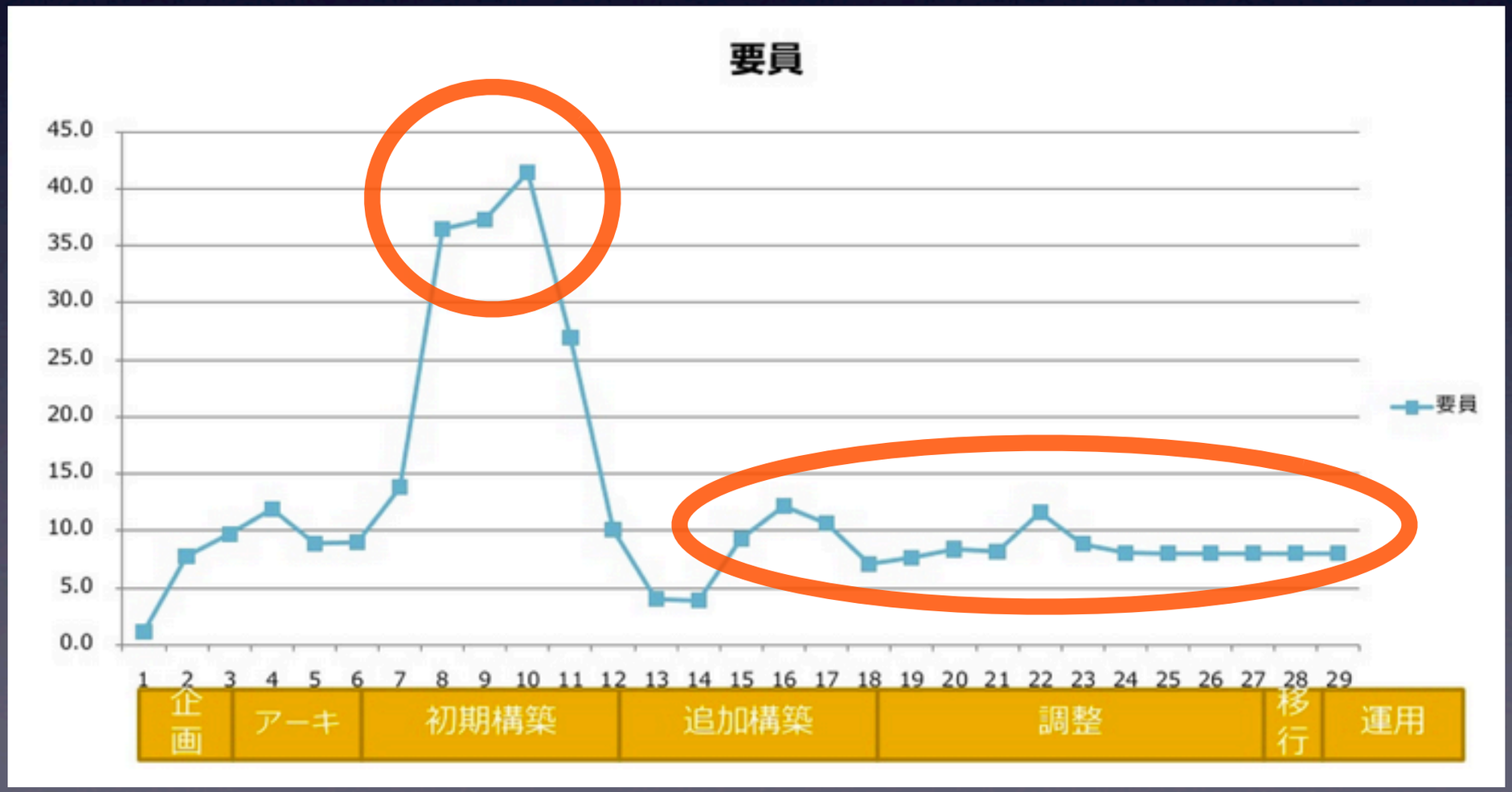
個別 / 多機能 / 大人数開発

http://www.slideshare.net/gxp/sgt2014gxp

課題

・スケールアウトにどう耐えるか

・継続的な変化をどう実現するか



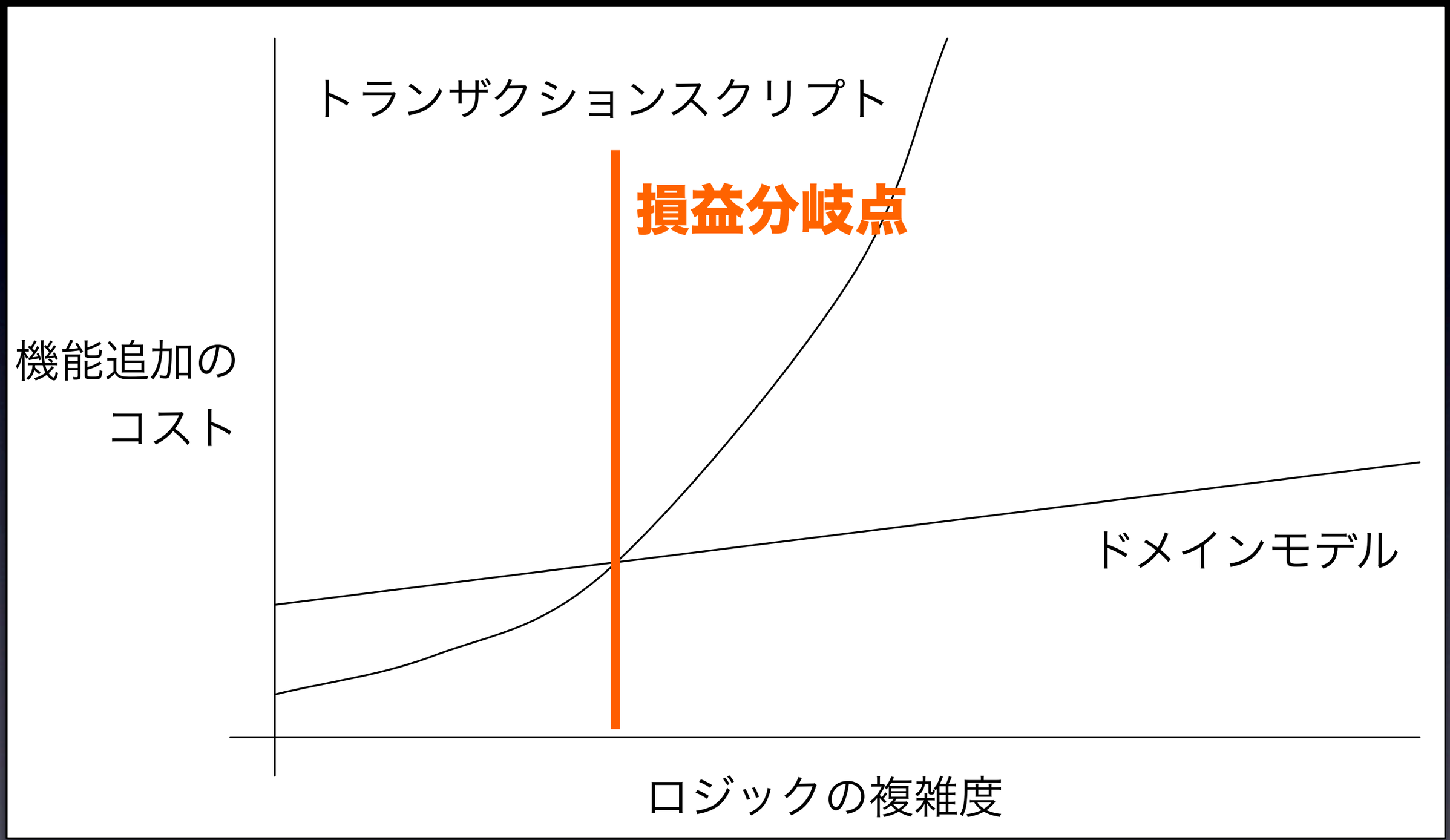
柔軟さと単純さのバランス

- 変化するところはしなやかに
- 変化しないところは硬く

どうバランスをとるの？

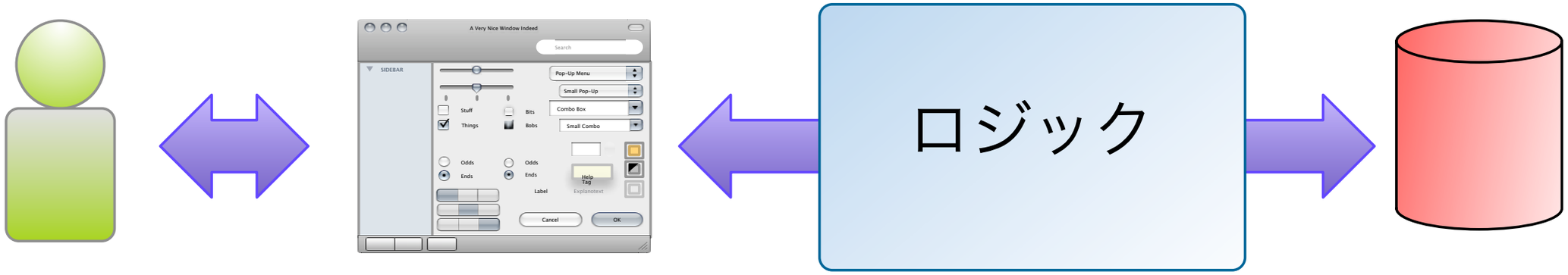
3D Character and Question Mark http://www.flickr.com/photos/crystaljingsr/3914729343/ by 姒儿喵喵

手続き型 vs モデル型



PoEAAより

システムの基本モデル

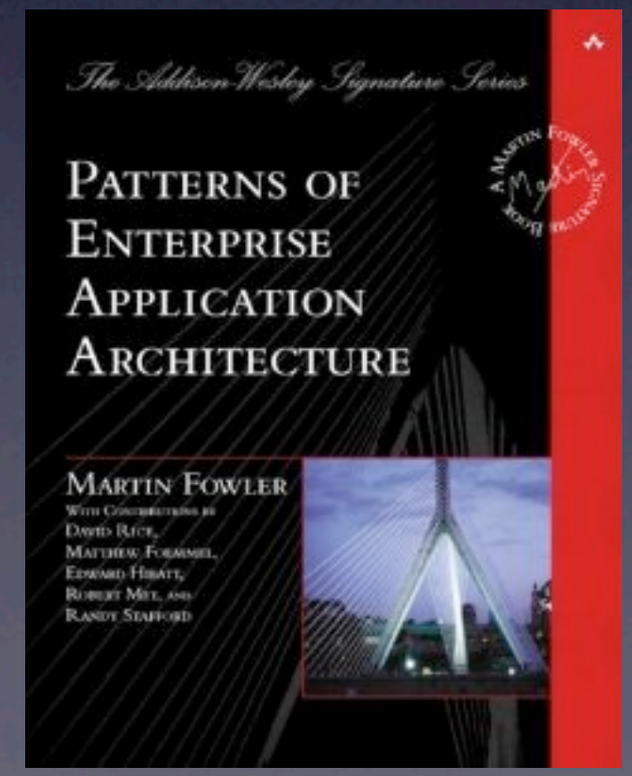


ユーザー UI 永続データ

ユーザは、ユーザインターフェイスを通じて、データにアクセスする。

ロジックの構成方針

- トランザクションスクリプト
  - ユーザーの要求を満たす**手続き**
- ドメインモデル
  - 複雑なロジックを
    オブジェクト指向で解決する



トランザクションスクリプト

UI

入力チェック

```
<script
 var a=
 var xl
 if(xls
```

DBアクセス

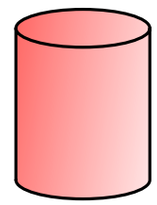
パラメタ

結果セット

SQLテンプレート

```
<script
 var a=
 var xl
 if(xls
```

SQL

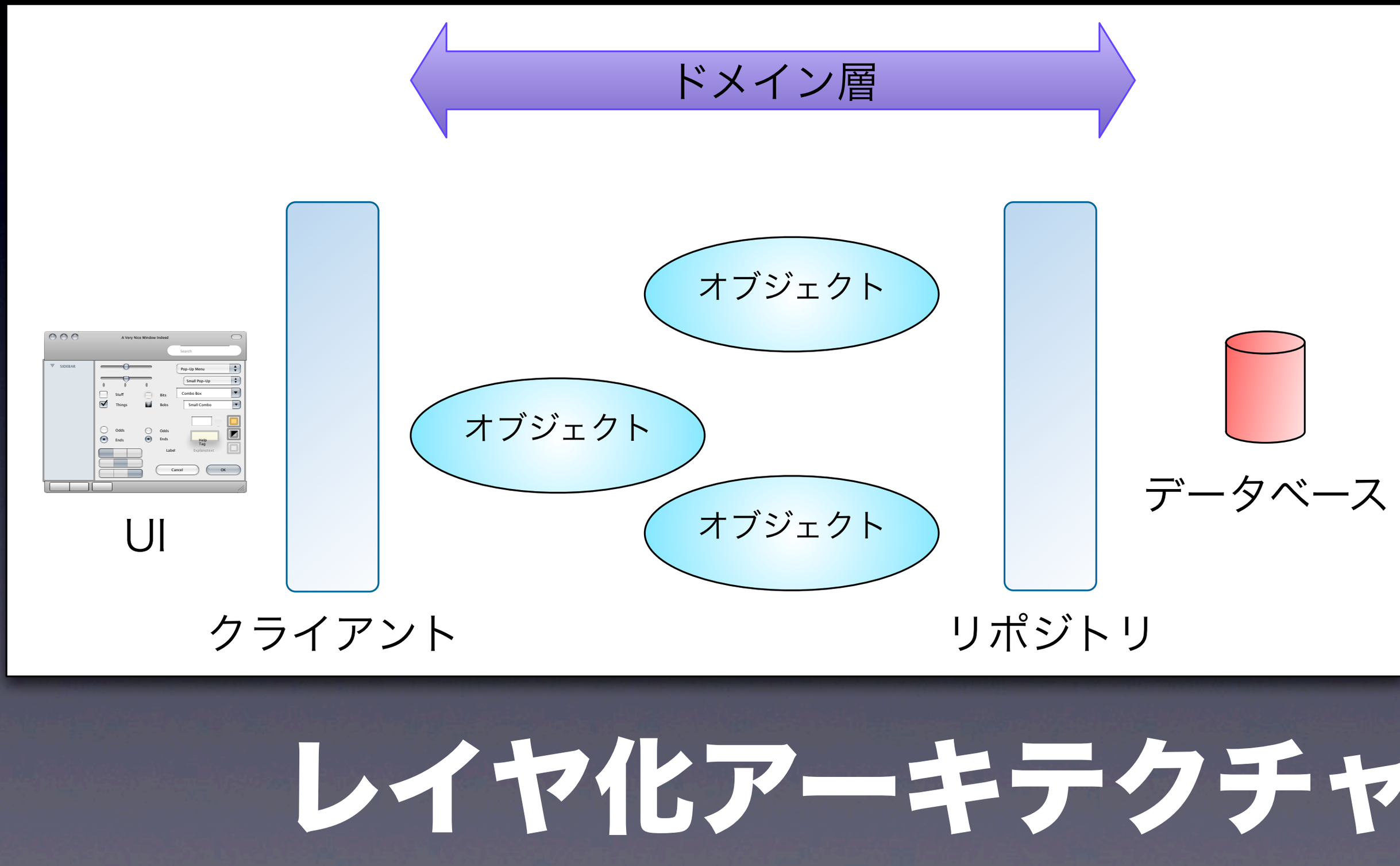


データベース

編集ロジック

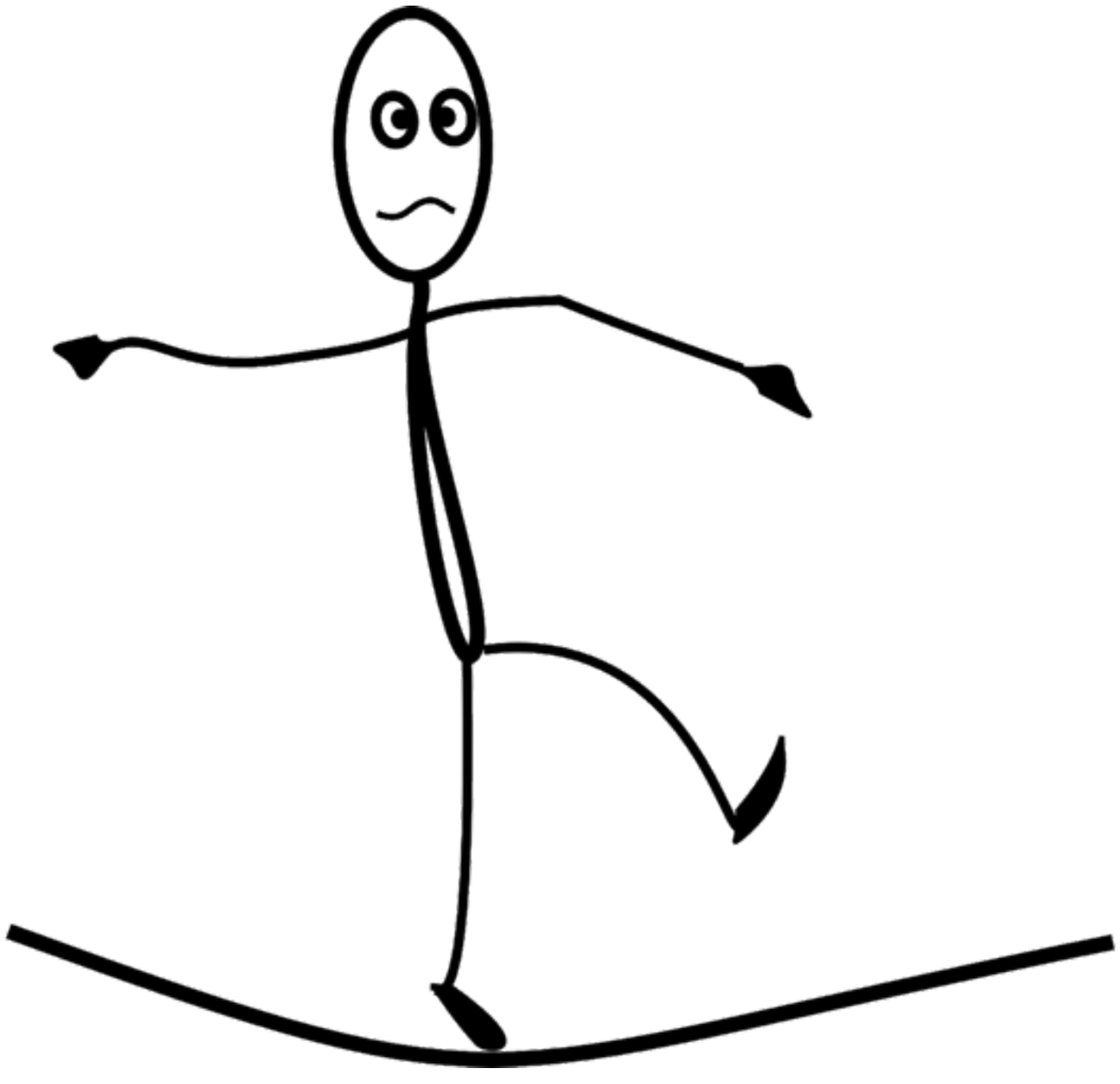
```
<script
 var a=
 var xl
 if(xls
```

ドメインモデル



レイヤ化アーキテクチャ

そう言われましても



ドメイン分割
と
コンウェイの法則

テーマ

トランザクションスクリプトとドメインモデルという対比を軸に、システム全体をどう組み立てるのか

基本方針

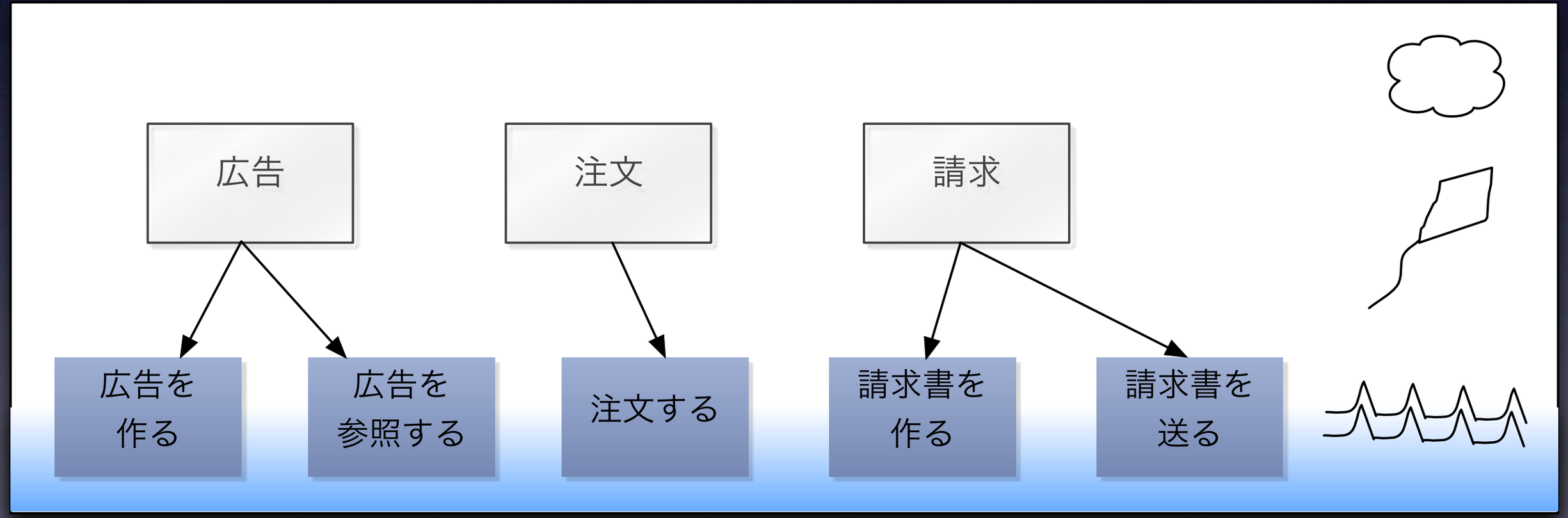
- ドメインの**境界**は明確に
  - ドメイン同士を結びつけるインターフェイスは注意して設計する
- ドメインの分割に際して**体制**を無視しない
  - コンウェイの法則重要

最初のドメイン分割

基本

目的の違い

「サマリーレベル」（雲または凧）のユーザーゴール

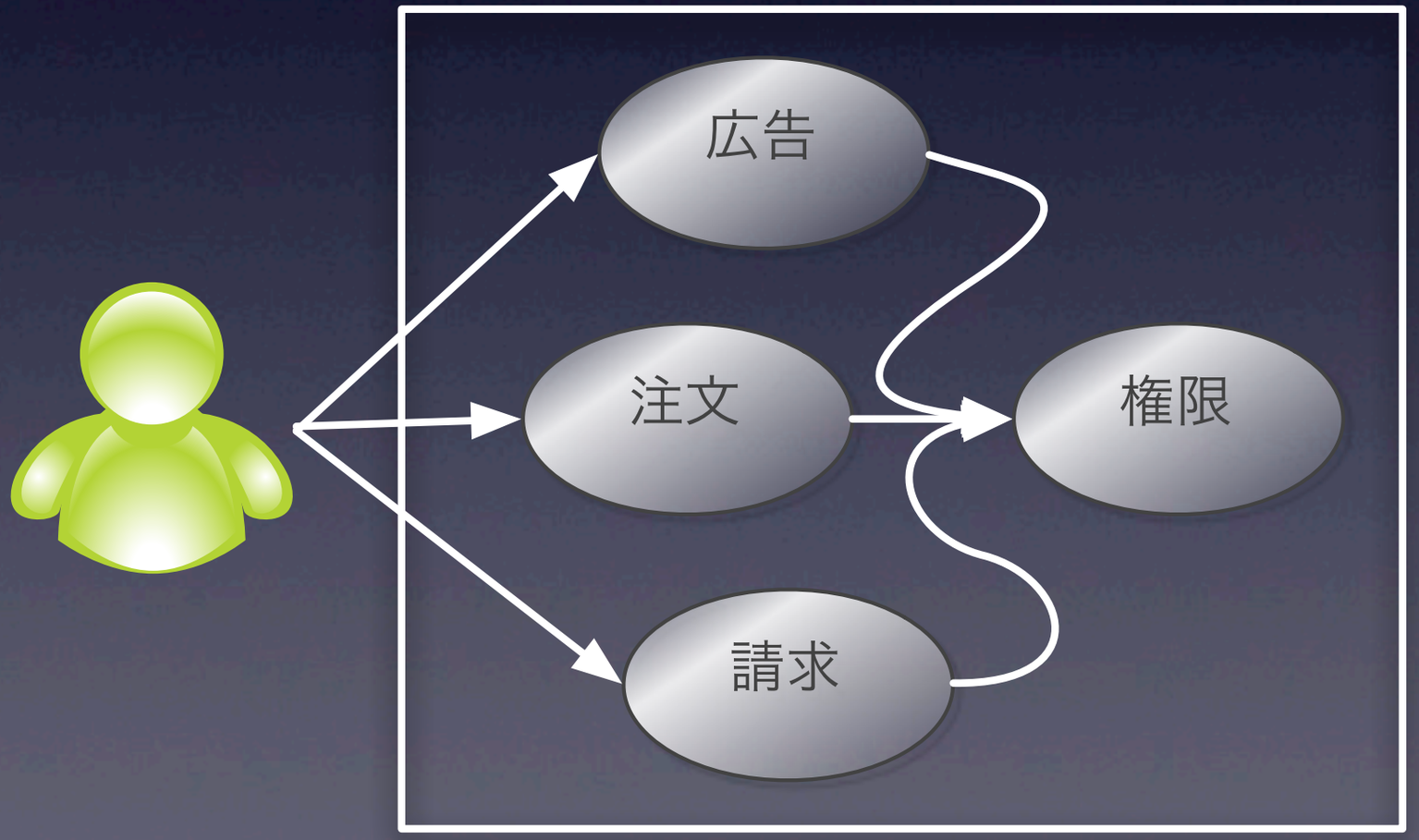


Alister Cockburn “Writing Effective Use Cases” Addison-Wesley 2001 p.62

基本

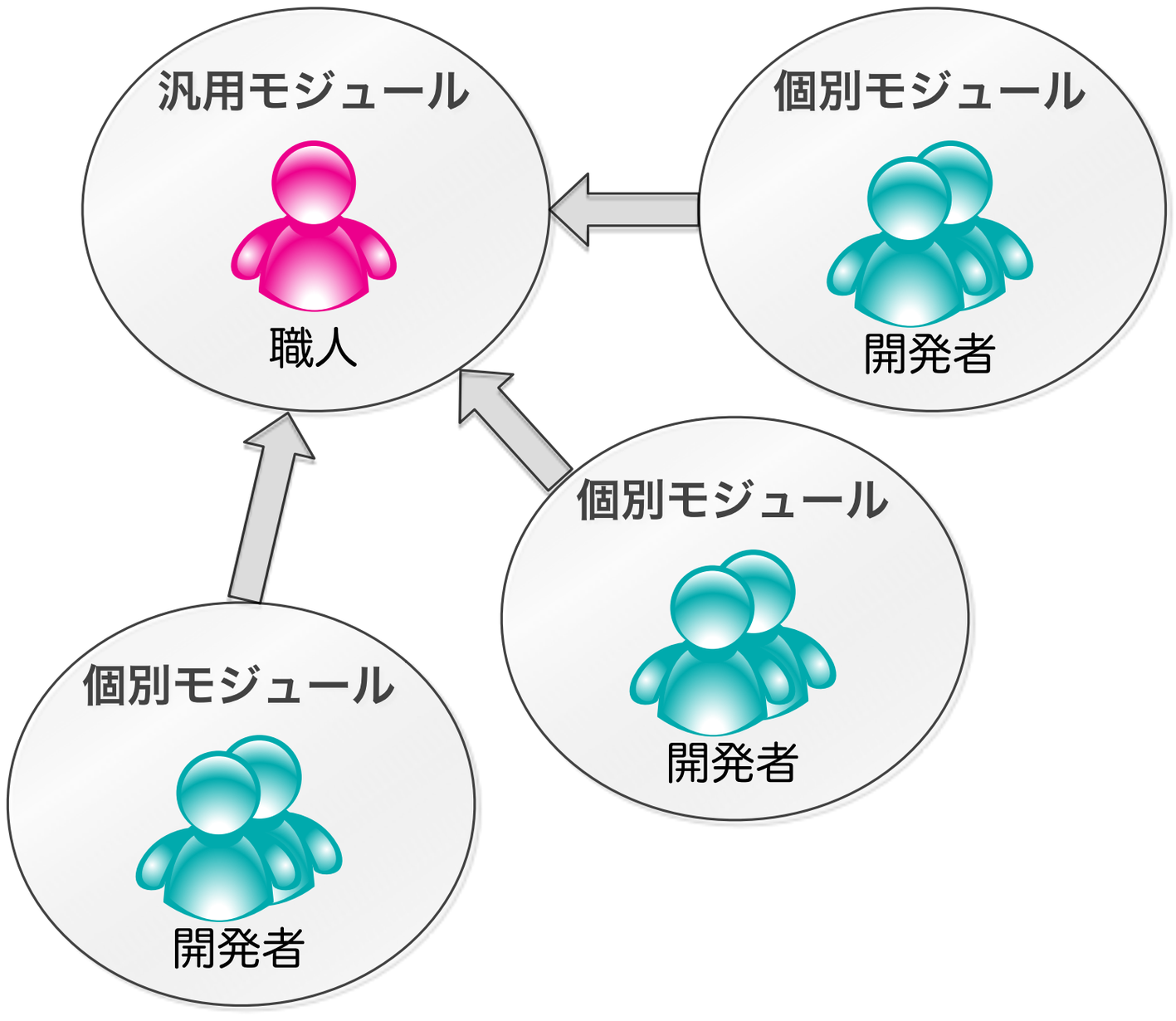
横断的関心

業務的な関心が
複数のドメインにまたがる



コンウェイの法則

組織の構造とプロダクトの構造を
そろえておく



コアドメイン

と

しなやかな設計

テーマ

複雑さを凝集したドメインに
どう立ち向かうか

基本方針

- **レイヤ化アーキテクチャ**により、オブジェクトが動ける空間を作る
- 基本は**オブジェクト指向**の手筋
- 業務的な意味のある**概念**に対して敏感になること

レイヤ化アーキテクチャ

システム的な都合から業務的な概念を分離するための空間

- 「SQLを発行して**結果セット**を取得する」から「リポジトリに問い合わせて**オブジェクト**を取得する」へ

オブジェクト指向

・小さいクラス、小さいメソッドを作ることを怖がらない
・手続きに意味を与える手段になる
・ストラテジーパターン多用

概念の抽出

制約

プロセス

組み合わせがあるルール

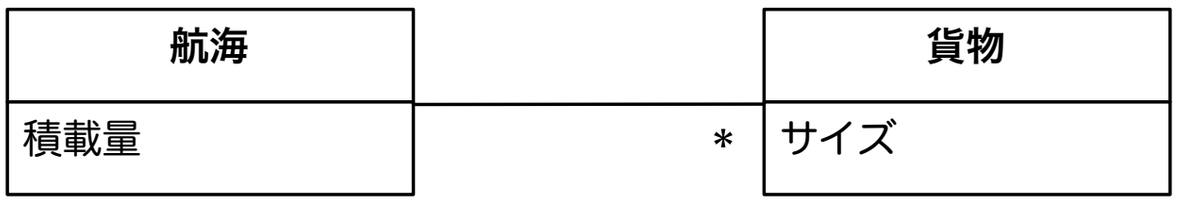
制約が手続きに溶け込む

貨物予約ドメイン

仕様書

10 %のオーバーブッキングを認める

データモデル



ソースコード

```
// 貨物を追加する
int 予約済み貨物量 = …
if(予約済み貨物量 + 貨物.サイズ > 航海.積載量 * 1.1){
   // 予約できない
   return … ;
}
```

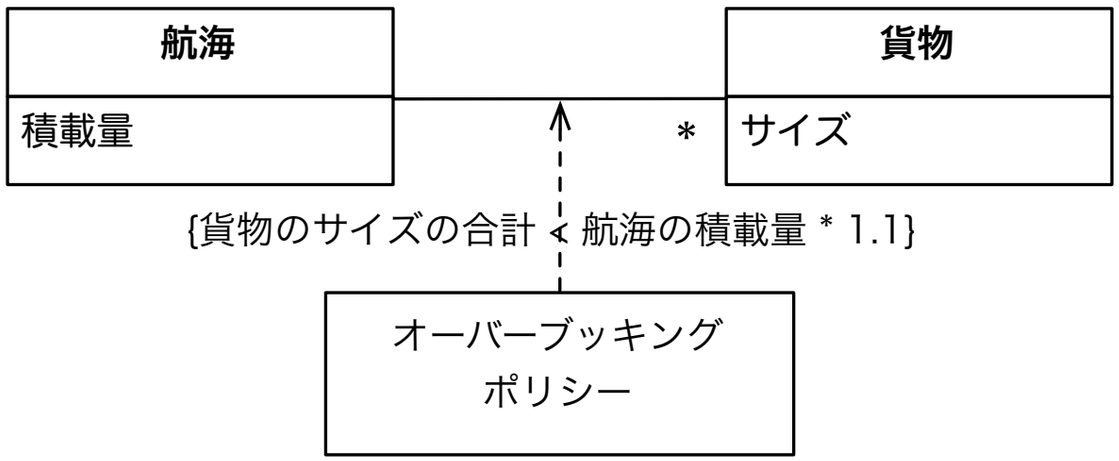
制約が**可視化**される

貨物予約ドメイン

仕様書

10 %のオーバーブッキングを認める

ドメインモデル



ソースコード

```
// 貨物を追加する
int 予約済み貨物量 = …
if(オーバーブッキングポリシー.allows(貨物, 航海)){
    // 予約できない
    return … ;
}
```

まとめ

- DDDはオブジェクト指向の「正しい」使い方を教えてくれる本
- エンタープライズでは、オブジェクト指向の使い所が難しい
- システムの急所を見極めて、しなやかな設計を

ありがとうございました！