



# ソフトウェアアーキテクチャ講座シリーズ アスペクト指向

## 第1回 全体構成 鄭



# 本講座の構成

- 第1回 全体構成
- 第2回 オブジェクト指向プログラミングの限界
- 第3回 アスペクト指向プログラミング
- 第4回 AspectJ・振る舞いへの作用
- 第5回 AspectJ・構造への作用
- 第6回 アスペクト指向分析/設計とユースケース
- 第7回 実践・ユースケースによるアスペクト指向開発
- 第8回 実践・ユースケースによるアスペクト指向開発(つづき): 拡張
- 第9回 Web開発におけるオブジェクト指向の限界
- 第10回 DIコンテナとSpring
- 第11回 SpringAOP
- 第12回 JavaScriptAOPフレームワークの実装と利用
- 第13回 AOJSの併用
- 第14回 事例
- 第15回 まとめ



## 第1回 アスペクト指向概論



## 講義内容

- アスペクト指向とは
- OOPの限界
- AOPの概念
- AOSDの概念
- 事例
- まとめ

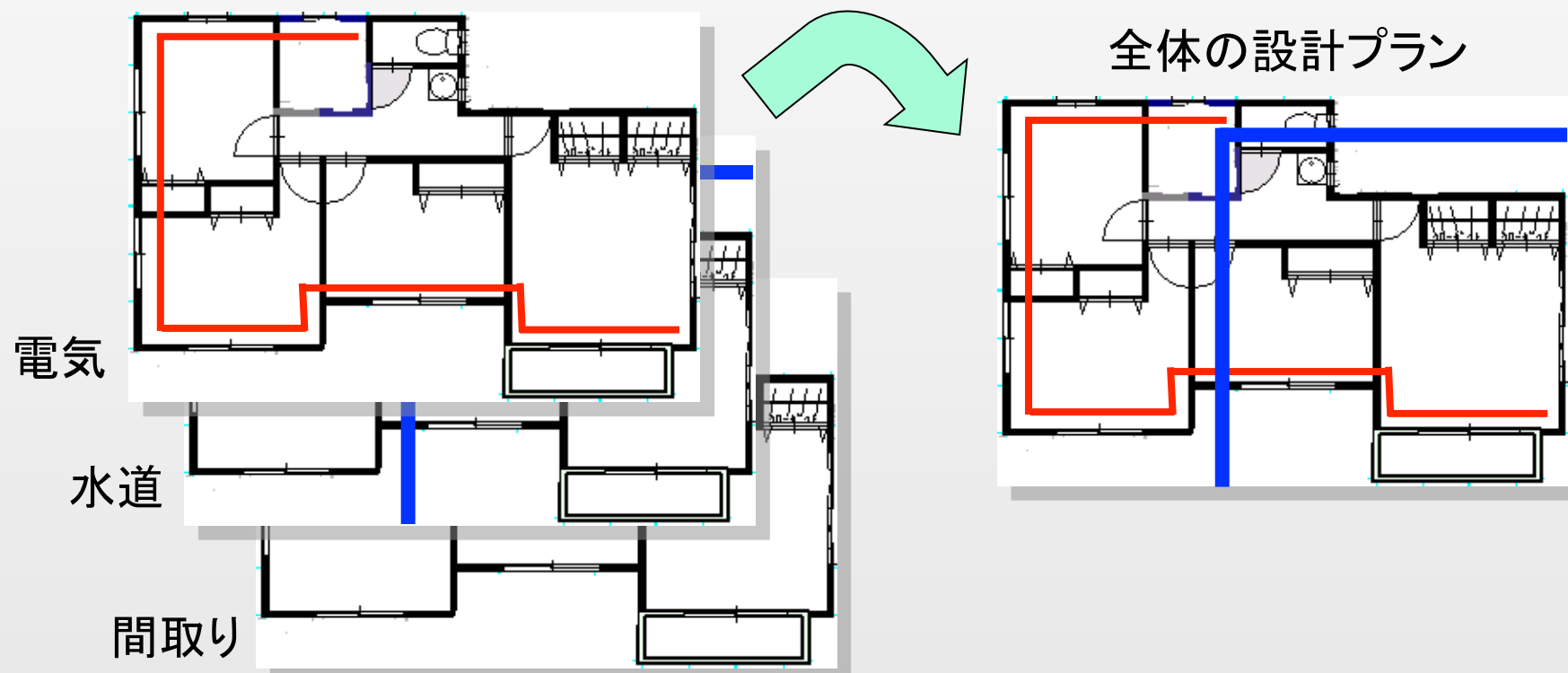


## アスペクト指向とは

# 建築: 設備は部屋を横断する

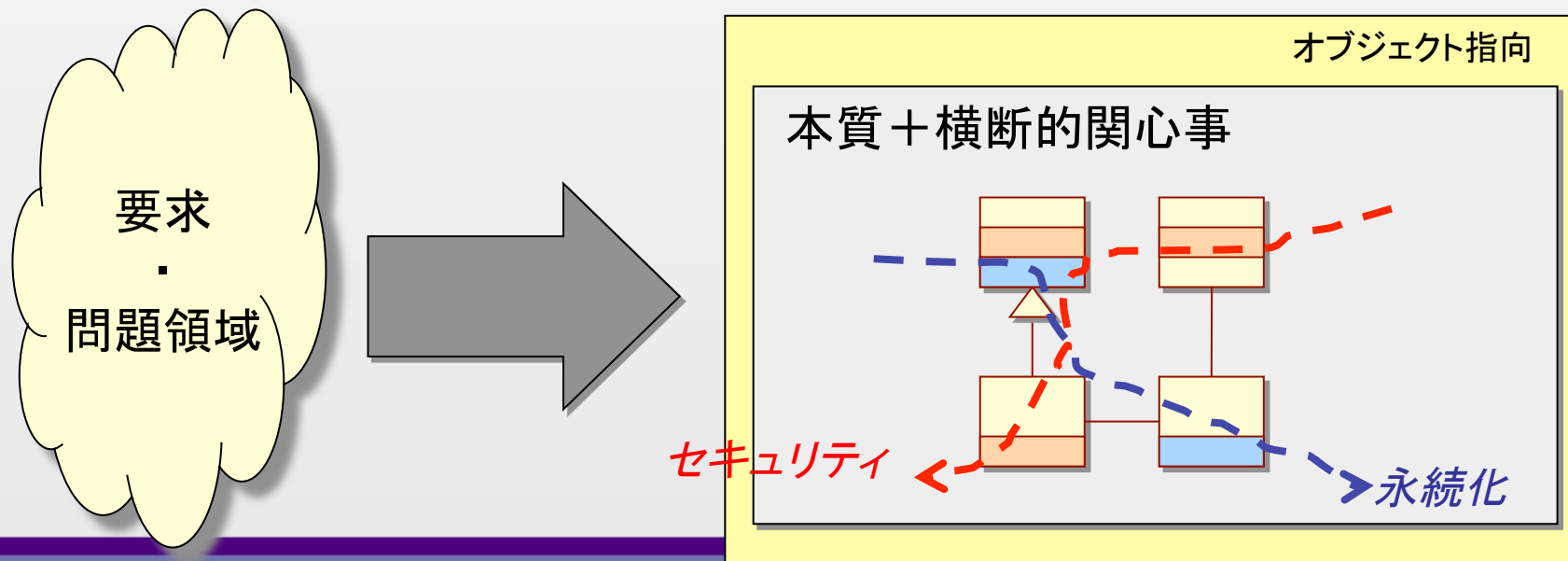
設備をOHPシートに分けて書けば・・・

- 業者ごとの個別作業
- 一貫した全体プラン



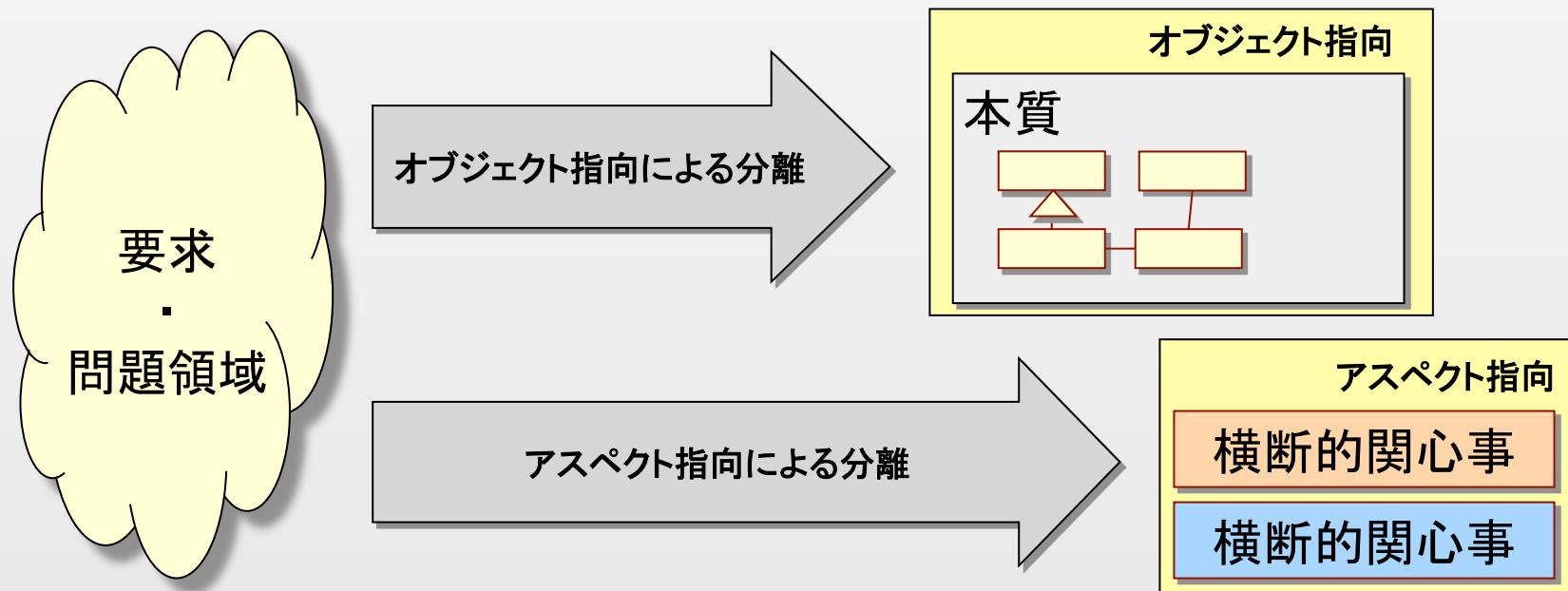
# オブジェクト指向とは?

- 抽象化: カプセル化、拡張・継承、ポリモーフィズム、コンポジション
- モジュール化: クラス
- モジュール構造を「横断」する事柄の存在



# アスペクト指向とは？

- オブジェクト指向による関心事の分離  
→ 低レベルの分離は不可      横断的関心事(Crosscutting Concerns)
- アスペクト指向
  - オブジェクト指向の限界を補い、さらに発展させるための技術





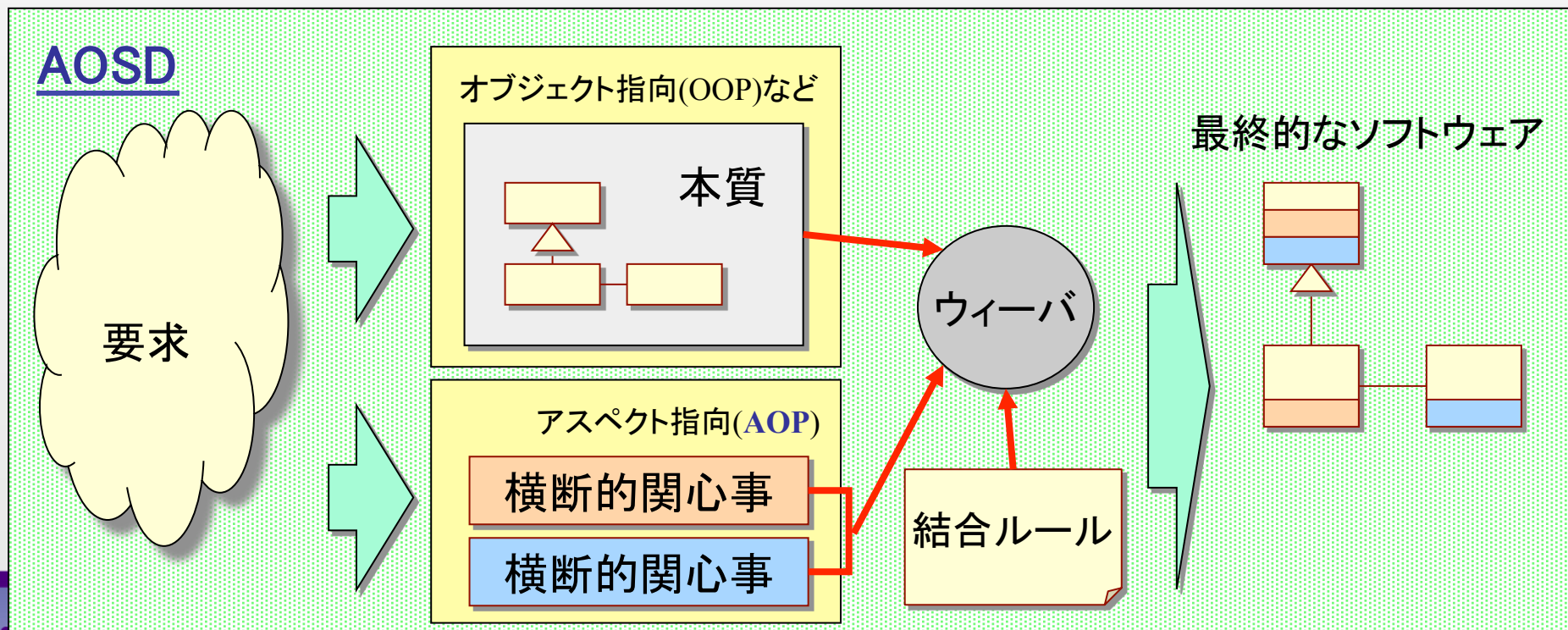
# ソフトウェア: 要求は構造を横断する

## ■ AOP(Aspect Oriented Programming)アスペクト指向プログラミング

本質と横断的関心事の集合として分離して捉え、結合ルールによって1つのプログラムへと纏め上げるプログラミング手法

## ■ AOSD(Aspect Oriented Software Development)アスペクト指向ソフトウェア開発

AOPにおける仕組みを上流工程へ流用





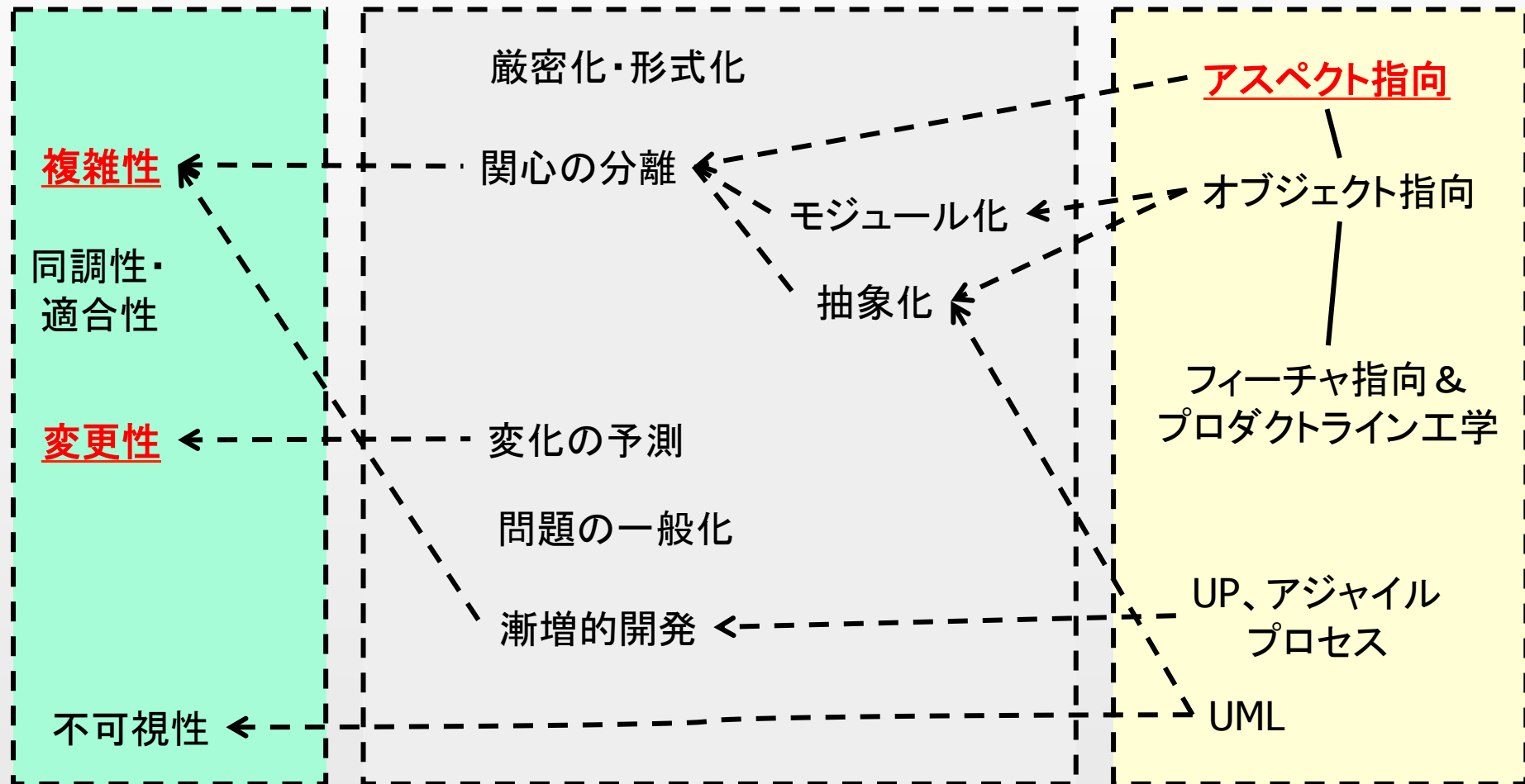
## OOPの限界



# 進化の方向性: ソフトウェア開発の難しさ と工学上の原則[Brooks75]

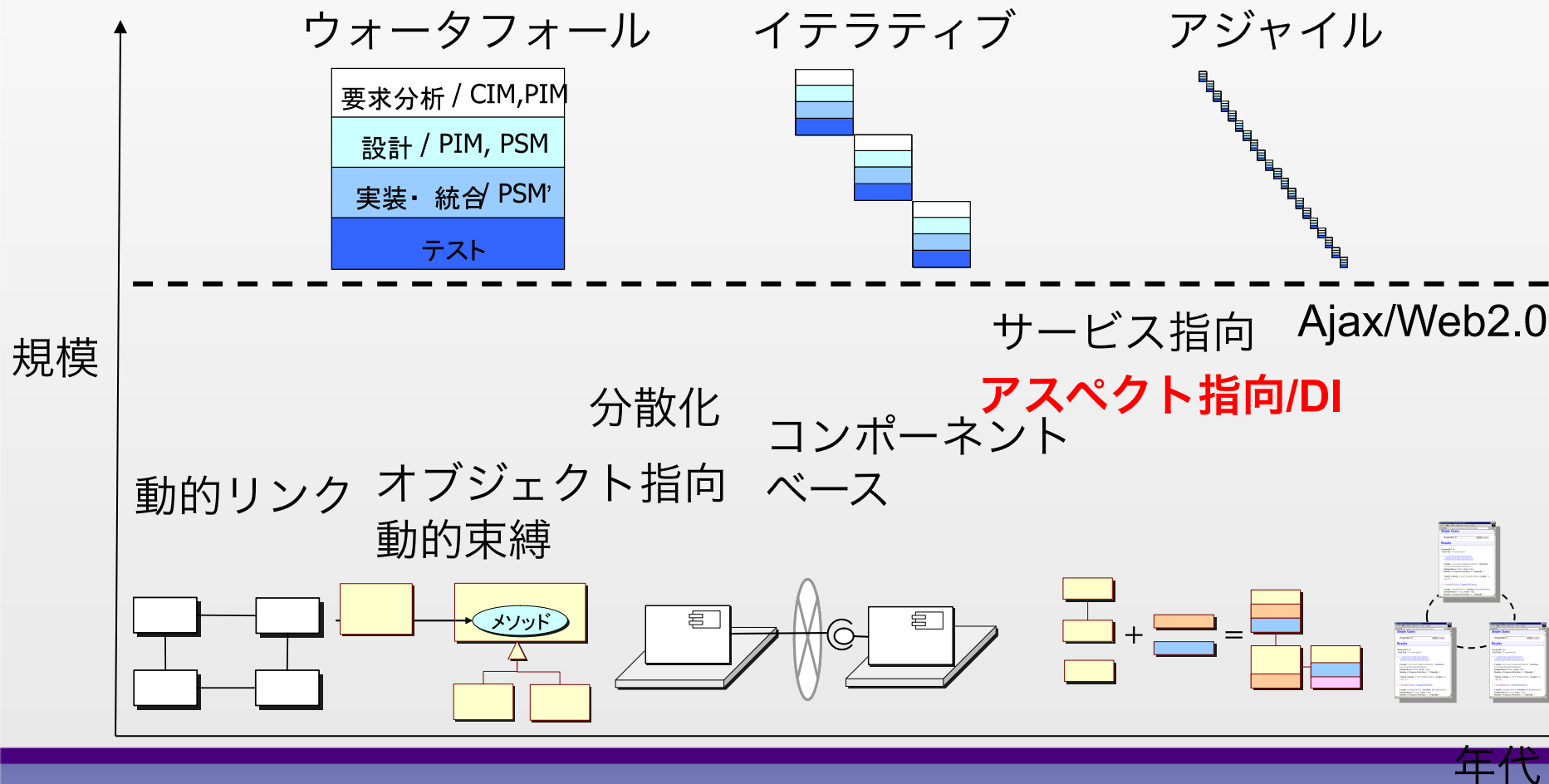
- ソフトウェア開発の難しさ
  - 複雑性 (Complexity)
  - 同調性・適合性 (Conformity)
  - 変更性 (Changeability)
  - 不可視性 (Invisibility)
- 原則
  - 厳密化・形式化 (Rigor and Formality)
  - 関心の分離 (Separation of Concerns)
  - モジュール化 (Modularity)
  - 抽象化 (Abstraction)
  - 変化の予測 (Anticipation of Change)
  - 問題の一般化 (Generality)
  - 漸増的開発 (Incrementality)

# 進化の方向性: 難しさと原則・技術 [Aye04]



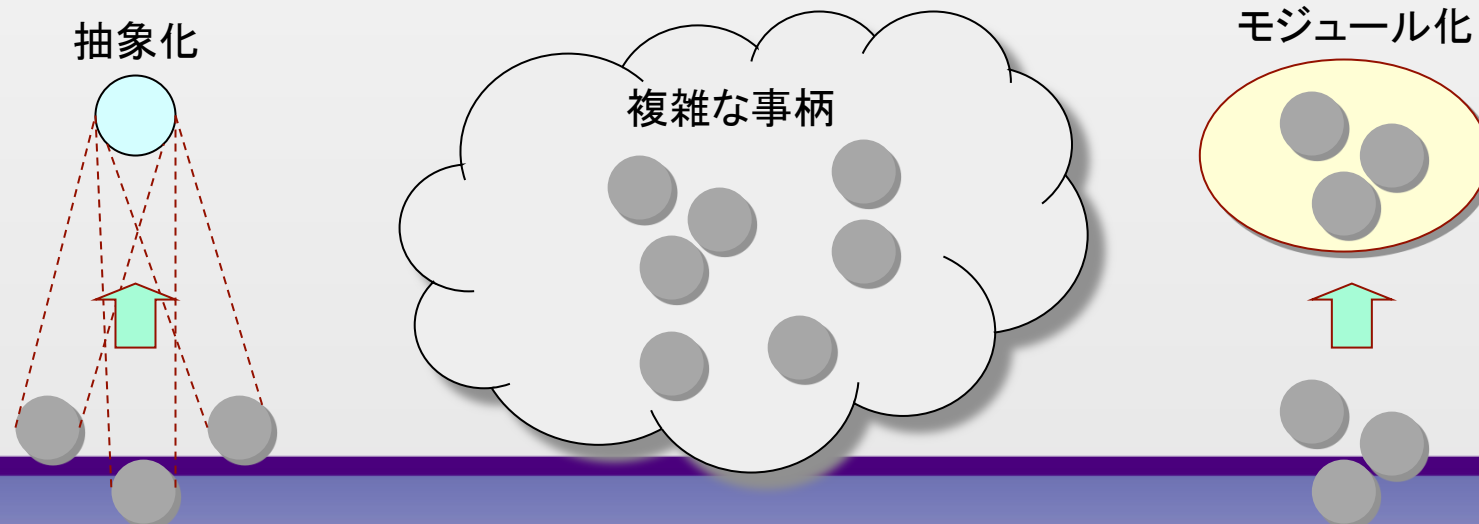
# 変更性への挑戦

- 昔: 最初に計画したとおりに作る
- 今: 変化に対応できるように作る



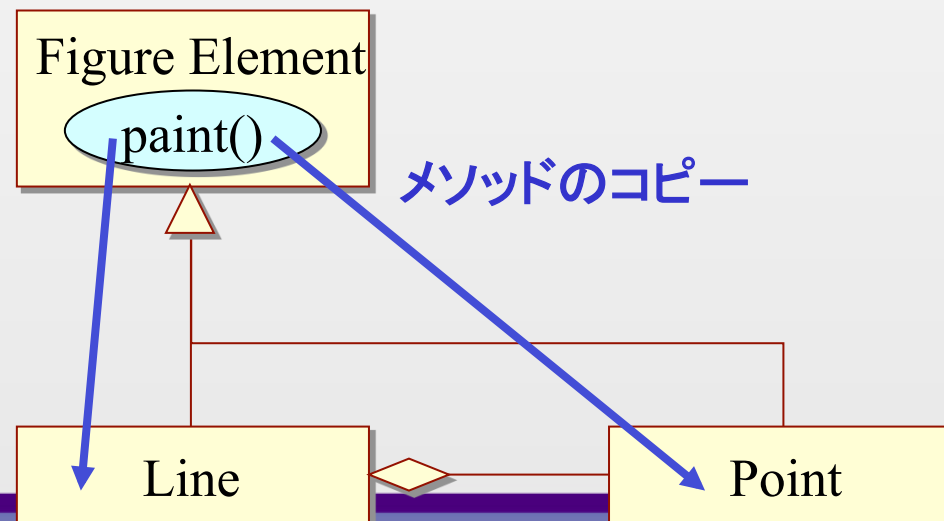
# 複雑性への挑戦

- 複雑な事柄を、高品質なプログラムとして効率よく表し管理したい
- 手段(1): 抽象化
  - 複雑な対象から詳細部分を排除して、本質的・共通なものに単純化して扱うこと
- 手段(2): モジュール化
  - 特定の事柄を達成するために必要な部分をまとめあげて扱うこと



# OOPによる複雑さへの挑戦

- 抽象化
  - カプセル化: クラスの特徴／責任の抽象化
  - 継承: 共通部分のくり出し
  - ポリモーフィズム: 利用方法の抽象化
  - コンポジション: 全体構成の抽象化
- モジュール化
  - アプリケーションロジック: データ構造と処理のモジュール化
  - データと処理のセットを、データごとに階層化／分類





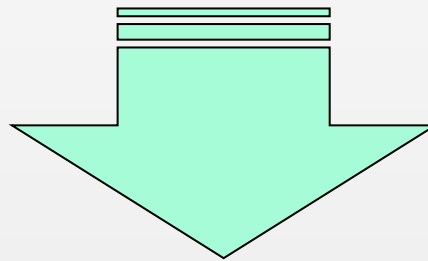
# 関心事と横断的関心事

- **関心事**＝実装に登場する様々な観点に基づく知識・技術
  - アプリケーションロジックが提供する機能
  - 実行環境における留意点
  - 対象ドメインに関する知識・技術
- **本質的関心事**＝開発上の支配的な関心
  - 対象世界の捉え方、視点
  - 機能・データの識別、モジュール化
- **横断的関心事**＝参加するモジュールが、階層・構造を横断して存在するような関心事
  - ロジック: ロギング、GUI描画処理など
  - 実行環境上の留意点: 高速化(キャッシュなど)、分散処理(ネットワーク関係)、ランザクション、セキュリティ、永続化など



# OOPにおける横断的関心事の悪影響

- OOPにおける横断的関心事の悪影響
  - 対応するコードが階層を横断して全参加要素に散らばってしまう



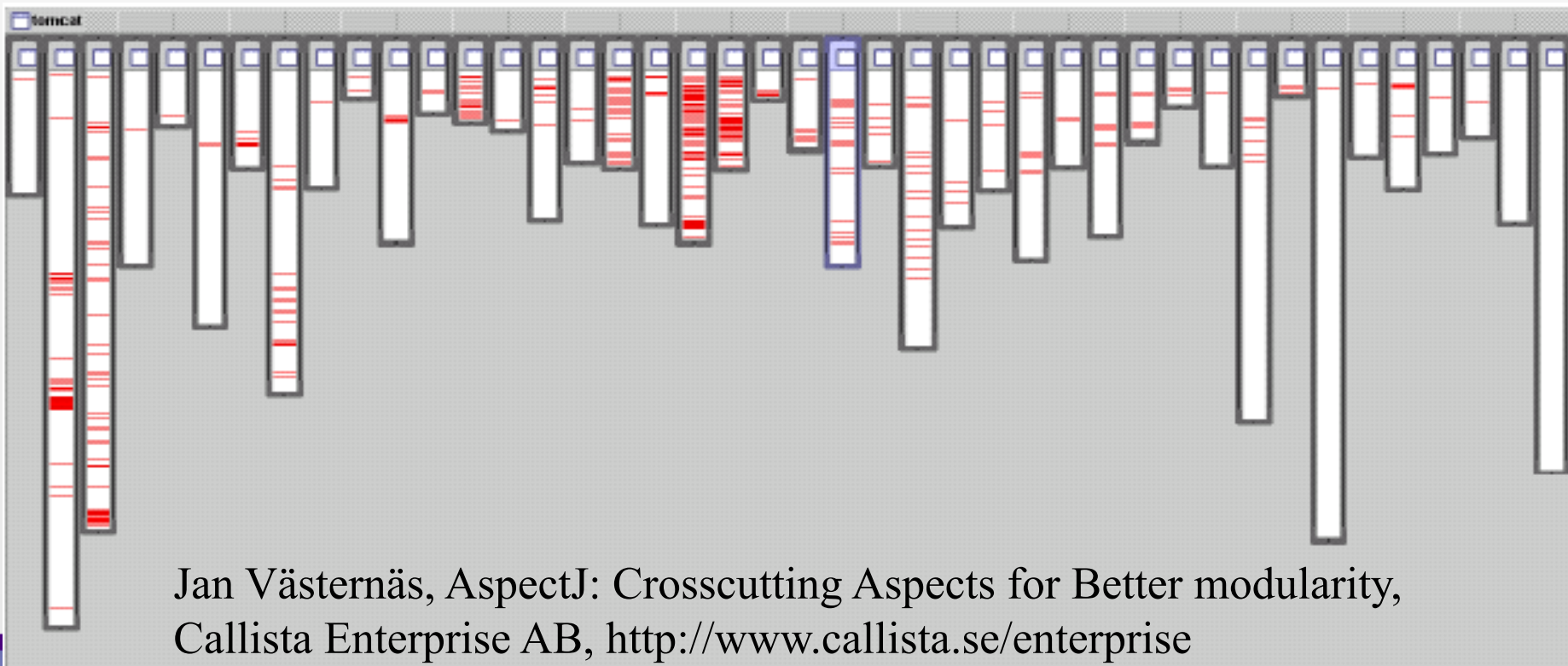
全体の構造が  
把握困難

同一のコードが  
複数の場所に出現、  
冗長な実装

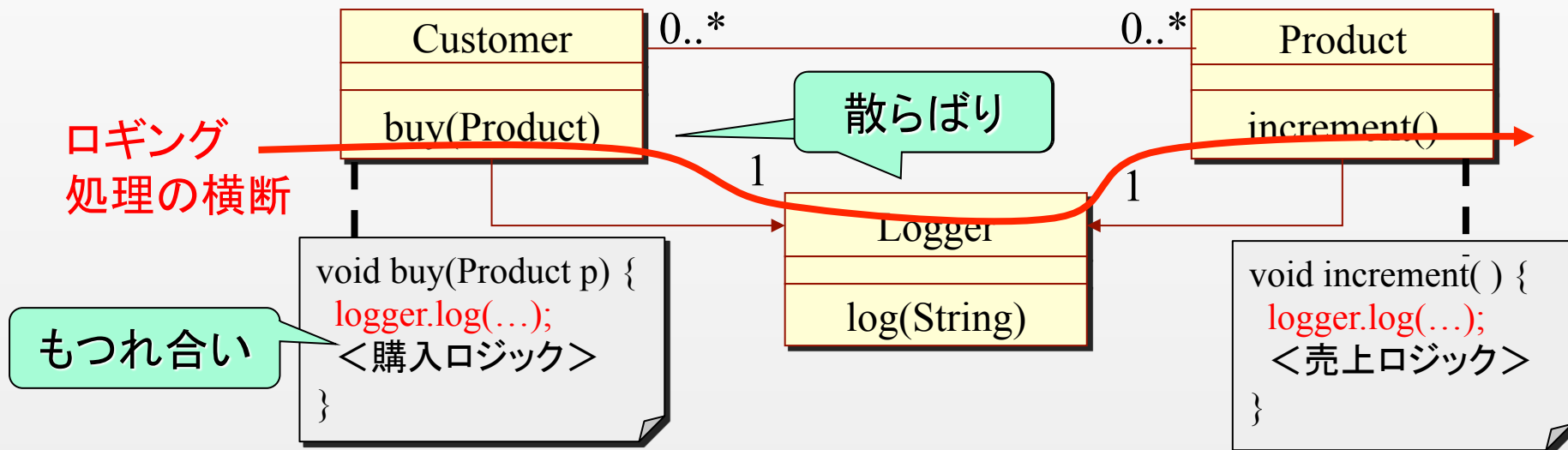
一貫性の維持  
保守管理が困難

# 散らばった重複コードの例

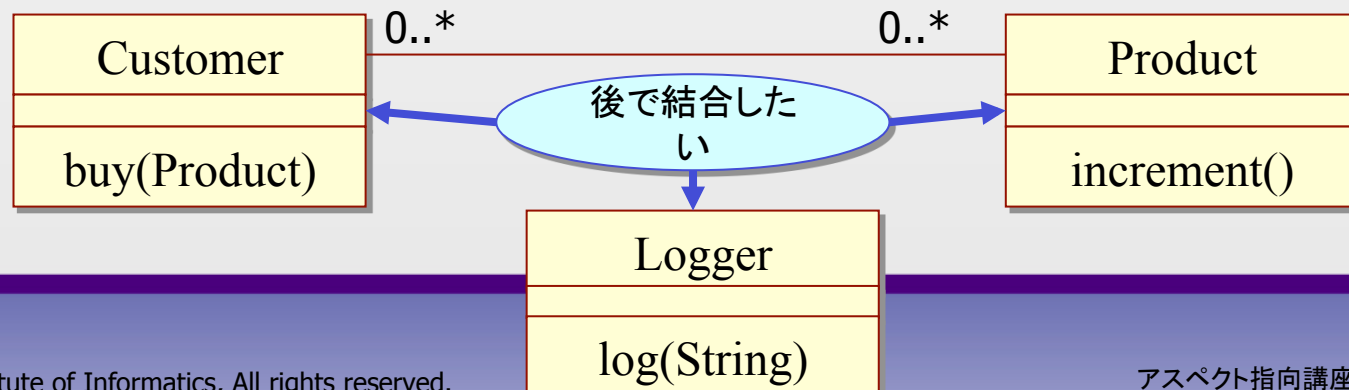
- Apache Tomcatにおけるロギングコードの散らばり
  - AJDT (AspectJ Eclipseプラグイン)による視覚化
  - ボックスがクラス、赤線が共通のロギングコード



- ビジネスロジックコードとロギングコードの混在
  - 何を、いつ：ログ出力を呼び出す側のモジュール群
  - どのように：ログ出力モジュール



- 異なる関心事としてモジュール化・分離したい
  - 何を、いつ：新しいモジュールにまとめたい



# OOPの限界を補う技術:AOP

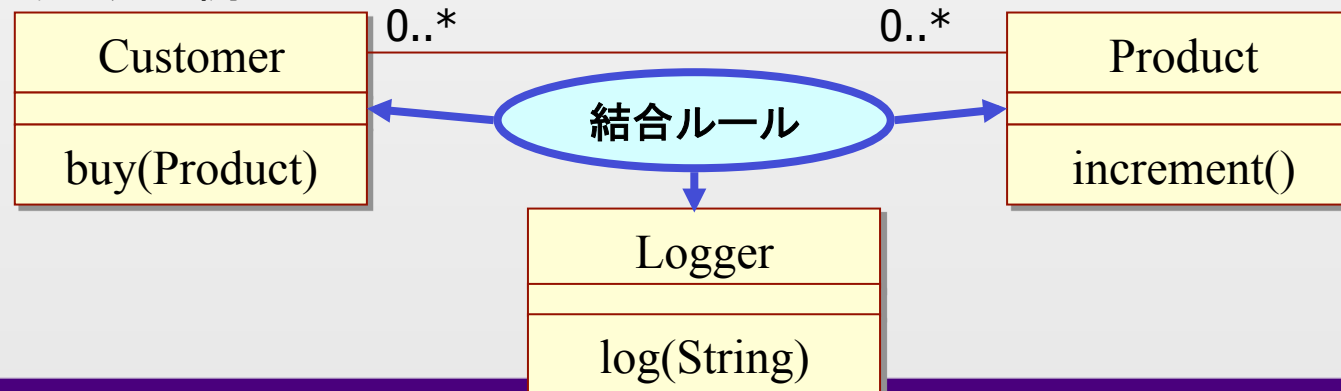
- OOPにおける横断的関心事の悪影響をAOPが補うと...

全体の構造が  
容易

同一のコードは  
一箇所に  
まとめて実装

保守管理が簡単

ロギングの例...





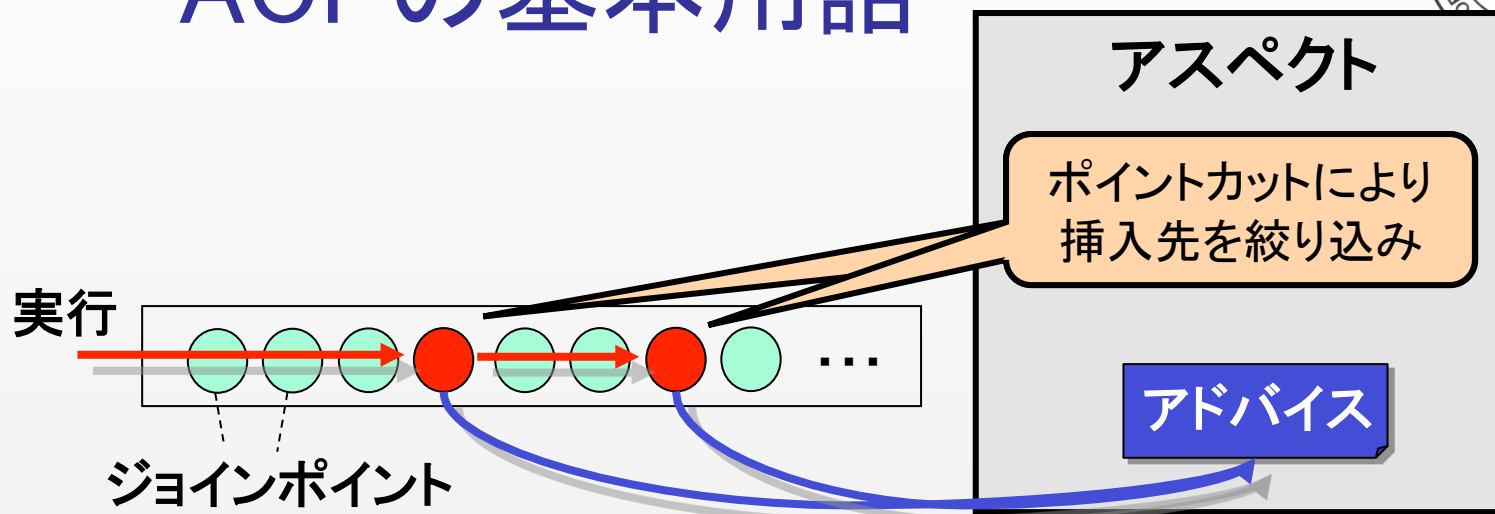
## AOPの概念



# AOP: Aspect Oriented Programming

- 対象を本質 (Core Concerns) と横断的関心事 (Crosscutting Concerns) の集合として分離して捉え、結合ルールによって1つのプログラムへ纏め上げる (Weaveする) プログラミング手法 [Kiczales97]
- ルーツ
  - 関心事の分離 (Separation of Concerns) [Dijkstra74]
  - 自己反映計算 [Kiczales93]
  - 開放型実装 [Kiczales97]
  - オブジェクト指向: Smalltalk (1972)  
※ただしオブジェクト指向を前提とするとは限らない
- モジュール単位「アスペクト」の追加を提案
  - 従来のモジュール単位を横断した処理をアスペクトに書く
  - 従来の単位もそのまま存続
  - クラスにアスペクトをウィーブする

# AOPの基本用語



用語	くだけて言う	説明
アスペクト	側面	ポイントカットとアドバイスの組み合わせを指定するモジュール
ジョインポイント	結合点	アドバイスの実行を割り込ませ可能なコード上の決まった位置
ポイントカット	点の絞り込み	プログラム中の全ジョインポイント集合から、部分集合を得るための絞り込み条件
アドバイス	挿入コード	スレッドの実行が、ポイントカットによって選択されたジョインポイントに到達したとき実行されるコード
ウィーブ	織込み	アドバイスを各モジュールに埋め込むこと

# 例: AspectJ [AspectJ]によるロギングの実現

クラス階層を横断



```
public class Test {
    public void work() {
        System.out.println("subwork.");
    }
    public static void main
    (String[] args) {
        new Test().work();
    } }

```

```
public class Logger {
    public void log(String msg)
    { ... }
}

```

work()メソッドの実行  
前にアドバイス実行

```
public aspect Logging {
    static Logger logger = new Logger();
    before() : execution(void Test.work()) {
        logger.log("Before work.");
    } アドバイス
}

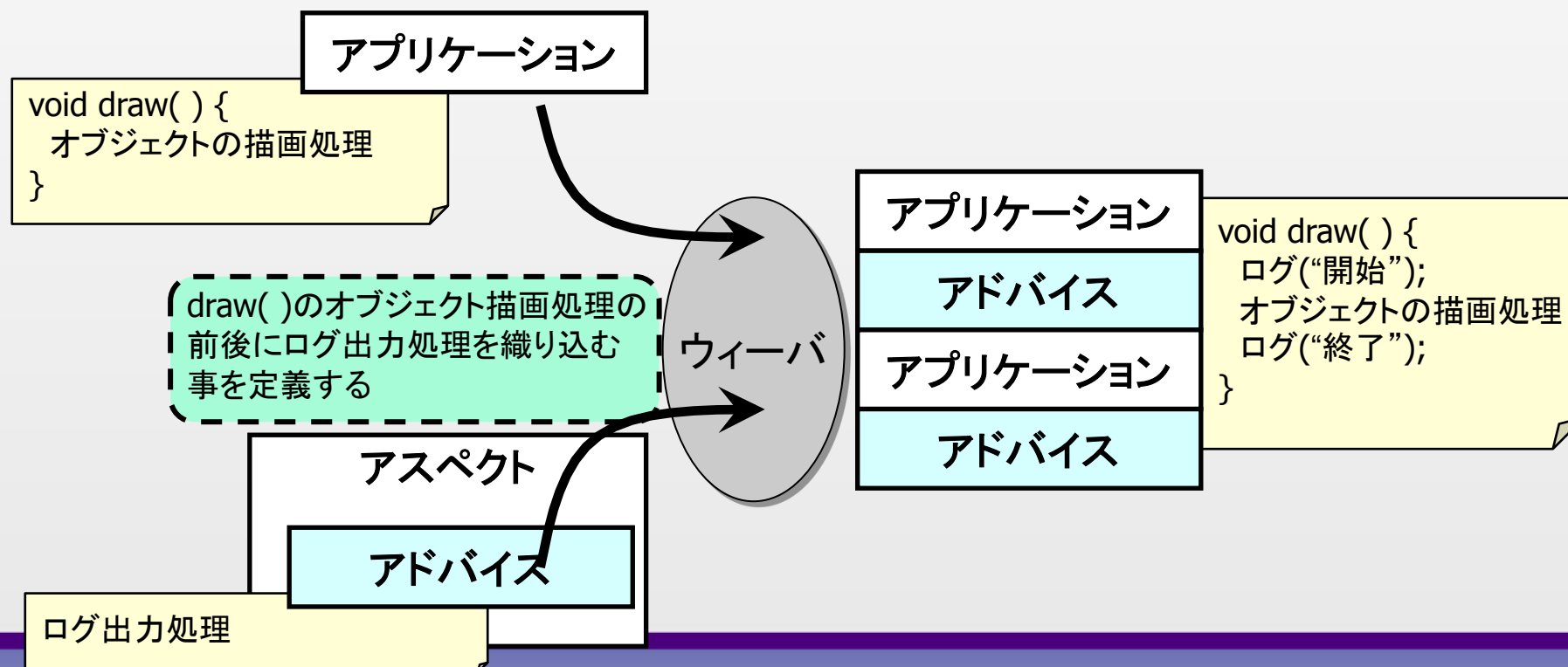
```

ポイントカット  
ジョインポイント



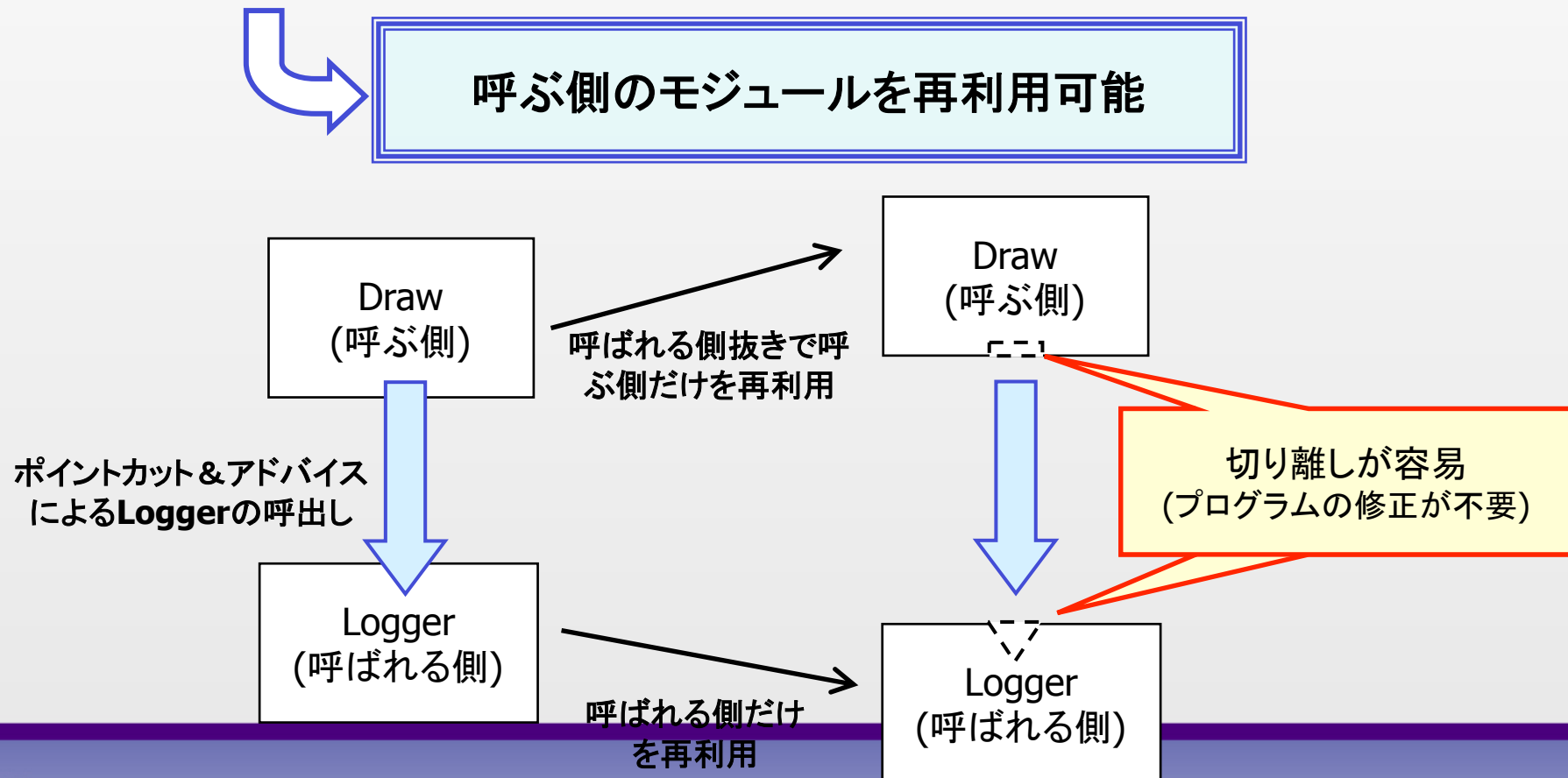
# アスペクト指向プログラムの構造

- モジュール単位「アスペクト」の追加
  - 従来のモジュール単位を横断する処理をアスペクトに書く
  - 従来の単位もそのまま存続
  - 既存のモジュール群へ修正なしに、アスペクト(アドバイス)を合成
- アスペクトのコンパイルの流れイメージ



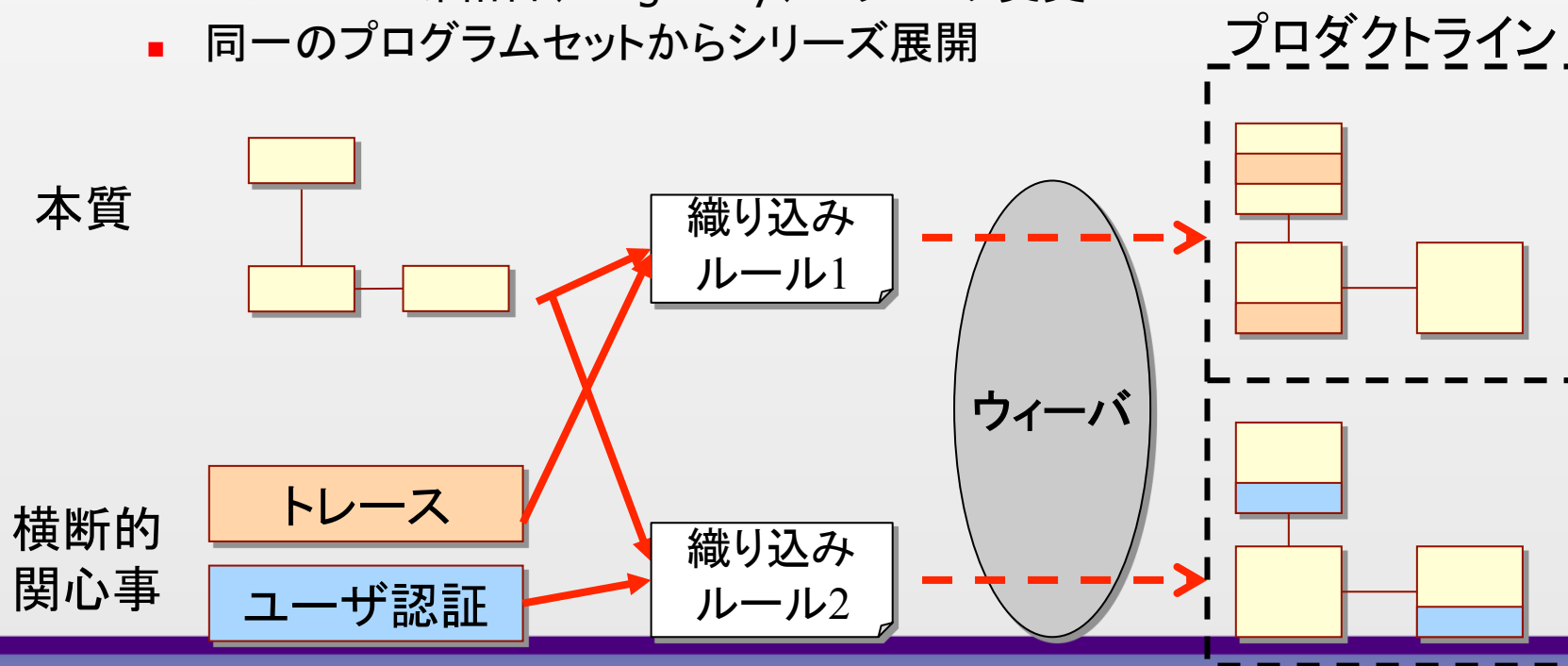
# AOPの利点

- 横断的関心事が「散らばらない」「もつれ合わない」 [千葉05]
  - アプリケーションの機能と非機能の分離が可能
  - モジュール間の結びつきを弱くする: 何を、いつ、どのように



# AOPの活用シーン

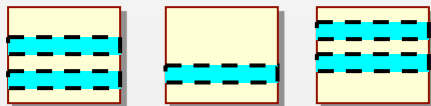
- 横断的関心事のモジュール化と集中管理
  - デバッグ支援: ログング、トレース、プロファイリング...
  - 高品質化: キャッシュ(効率)、分散(スケール)、認証、Mutex(並行性)
  - ミドルウェア/ライブラリとの疎な接続: トランザクション、永続化...
- プロダクトライン開発
  - モジュール疎結合、Plug&Play、パラメータ変更
  - 同一のプログラムセットからシリーズ展開



## 注意: AOPはオブジェクト指向を前提としない

- プログラム実行の幾つかの点(ジョインポイント)に割り込めて
- 体系的に、割り込み方法をリフレクションで指定できればよい
  - 例: AspectC [AspectC] によるメモリ割り当て時エラー処理

- Before: malloc使用箇所に処理コードが散らばる



```
int *x ;
x=(int *)malloc(sizeof(int)*4);
if (x == NULL) {
    /* malloc失敗時の処理コード */
}
/* 本来のロジックコード */
```

- After: 元のプログラムはロジックのみ

```
after(void * s) :
    (call($ malloc(...)) ||
     call($ calloc(...)) ||
     call($ realloc(...)))
    && result(s) {
    char * result = (char *) (s);
    if (result == NULL) {
        /* malloc失敗時の処理コード */
    }
}
```

+

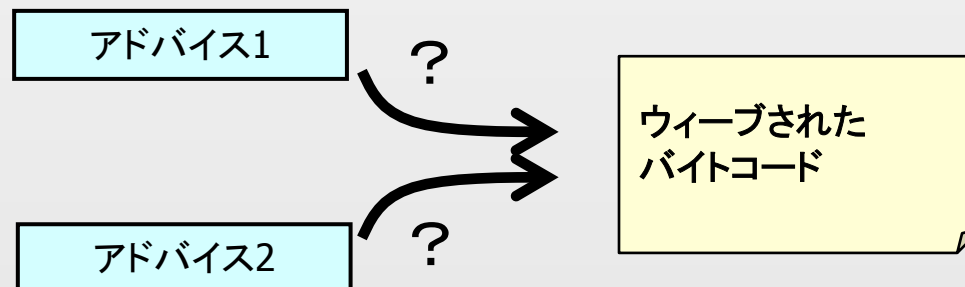
```
int *x ;
x=(int *)malloc(sizeof(int)*4);
/* 本来のロジックコード */
```

# AOPの問題点

- うまく使わないと複雑になる
  - 保守性低下: 全体の見通し、デバッグ困難
  - 効率性低下
  - 信頼性低下: アスペクト間の干渉
- 従来とは異なるパラダイム: ラーニングコストが高い



実行中のコードに、どのアドバイス(アスペクト)がどのように  
ウィーブされているか直感的に判断しづらい

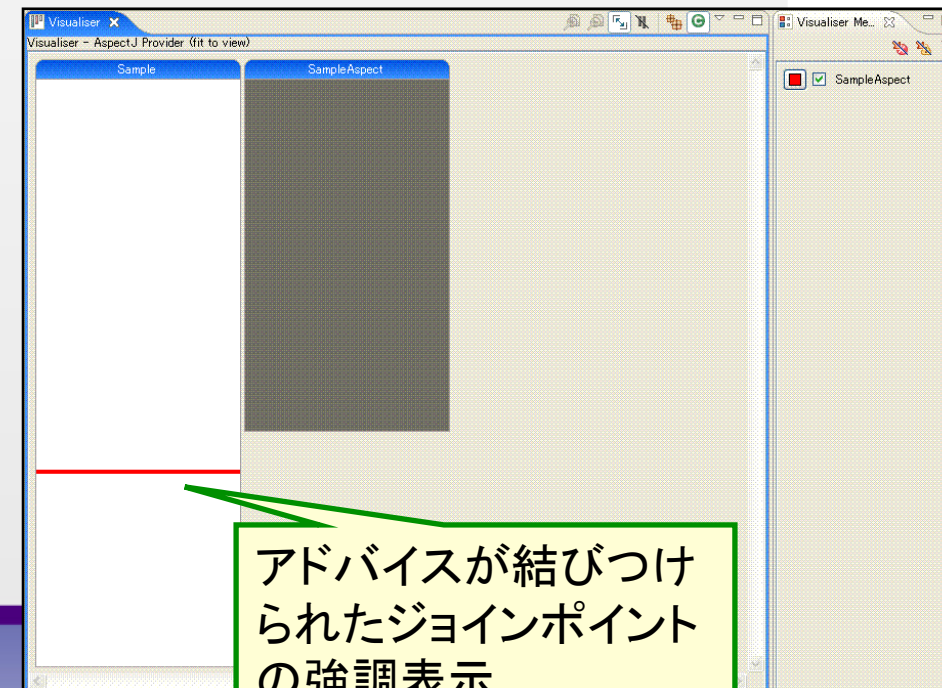


# 問題点への1つの解決策: 統合開発環境によるサポート

- AOP統合開発環境の活用
  - アドバイス挿入箇所の可視化
  - ジョインポイントに結び付けられたアドバイスへのジャンプ
- 例: AspectJのEclipseプラグインAJDT
  - AJDTのアドバイス・インジケータ、アスペクト可視化
  - アスペクトを含むデバッグ支援: ブレークポイント、ステップ実行

```
advised by SampleAspect.before(): hogecut..
{
    return "hoge";
}
```

ブレークポイントの切り替え(K)	
ブレークポイントを使用不可にする(D)	
注釈にジャンプ(G)	Ctrl+1
ブックマークの追加(K)...	
タスクの追加(T)...	
✓ クイック Diff の表示(Q)	Ctrl+Shift+Q
行番号の表示(N)	
折りたたみ(O)	
設定(E)...	
Advised By	SampleAspect.before(): hogecut..
AspectJ Tools	
ブレークポイント・プロパティ(R)...	



Visualiser - AspectJ Provider (fit to view)

Sample

SampleAspect

アドバイスが結びつけられたジョインポイントの強調表示



# アスペクト指向プログラミング のまとめ

- アスペクトによる横断的関心事のモジュール化
  - 振る舞いへの作用: ジョインポイントモデルによるクラス階層を横断する処理の宣言的記述
  - 構造への作用: プログラムの構造を変更可能
  - 既存のコードを書き換えずに, プログラムを修正可能  
⇒「散らばり」と「もつれ合い」の解消!
- アスペクトの応用
  - 開発アスペクト: トレース、テストコードなど
  - 製品アスペクト: GUI描画処理、ロギングなど
  - プロダクトライン開発への応用

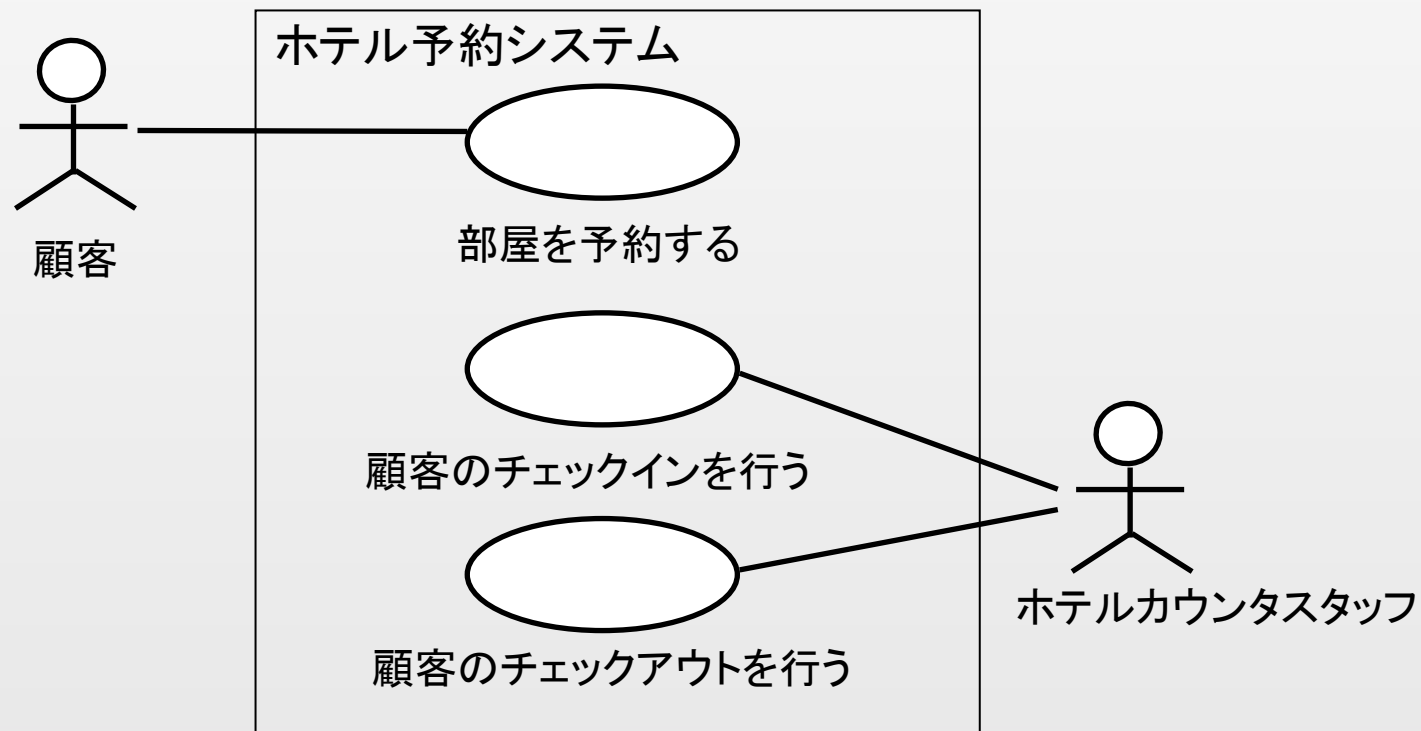


## AOSDの概念



# ユースケースを活用した オブジェクト指向分析/設計

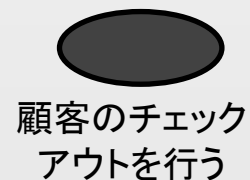
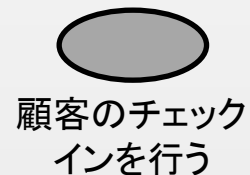
- 関心事(Concern): 利害関係者が興味を持つ事柄
- ユースケース: 外から見える振る舞い上の関心事を分離



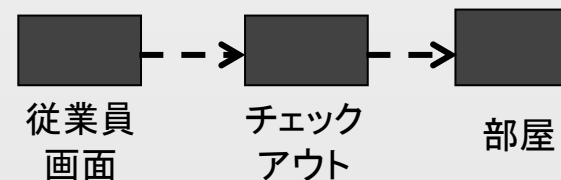
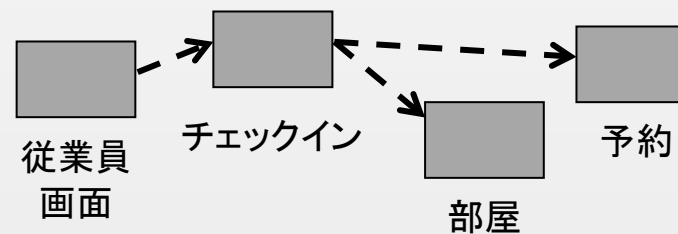
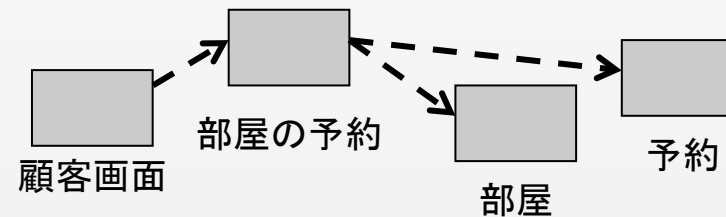
# 質問: ユースケースとOOA/Dの相性

以下のような分析/設計の進み方に問題はあるか？

ユースケース



ユースケース実現



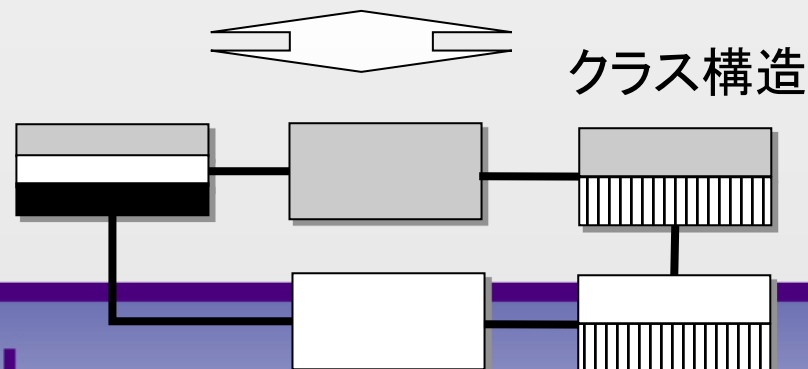
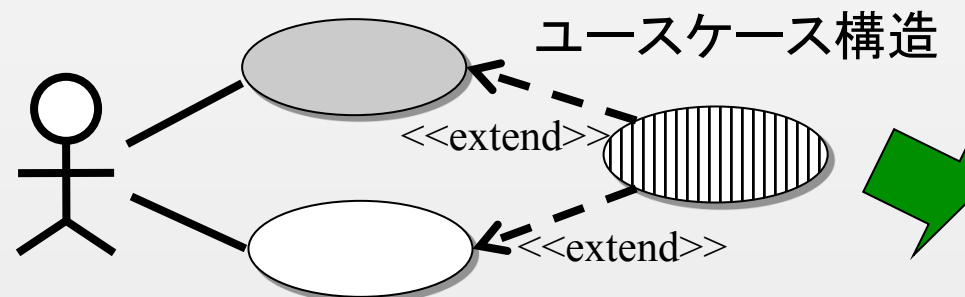


# AOSD(アスペクト指向開発)とは

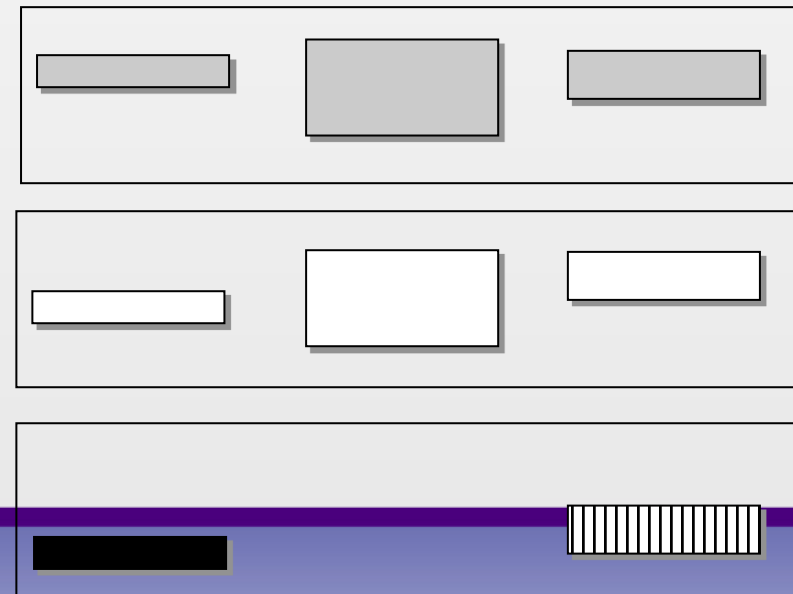
- AOSD (Aspect Oriented Software Development)
  - 要求定義から分析、設計、実装、テストに至るまで、アスペクトによってソフトウェアシステムを開発する全体的な手法のこと
  - 単なるAOPではなく、モジュール化をうまく行えるようにするあらゆる範囲の技法を網羅する
  - 既存の技法(オブジェクト指向、コンポーネントベース開発、デザインパターン、J2EE、.NET)とは競合しない (これらの技法の上に位置付けられる)
- AOSDの種類
  - サブジェクト指向を利用したもの[JW0601]
  - オブジェクト指向の親和性を重視したもの[JW0601]
  - ユースケースを用いたアスペクト指向開発 [Jacobson06]

# ユースケースを用いたAOSD

- 機能・非機能要求の両方をユースケースとして捉え、各実現をクラスとアスペクトで分離して実装、最後に纏め上げるモデリング手法
- 関心事の早期分離、拡張性・保守性・生産性の向上



クラスとアスペクトで分離





## 事例



- 日立製作所・日立コンサルティング
  - 既存のCOBOLプログラムについて、修正することなく、
  - コンプライアンスや内部統制に対応するための証跡・ログ出力機能をアスペクト指向で追加
  - <http://www.hitachi.co.jp/products/it/sox/service/aspect/index.html>
- 東芝
  - ソフトウェアの動作検証支援システムを開発
  - 検証対象の観測方法や分析方法をアスペクト指向スクリプト記述
  - [http://www.toshiba.co.jp/tech/review/abstract/2007\\_09.htm](http://www.toshiba.co.jp/tech/review/abstract/2007_09.htm)
- Siemens [Wikipedia]
  - 健康情報管理システムSoarian の開発におけるトレース、監査、パフォーマンスモニタリングにAspectJを利用



- 東工大・日立ソフト: Bugdel [Bugdel]
  - デバッグコード挿入用のAOP/Java Eclipseプラグイン
  - 日立ソフトによるRISCプロセッサJava開発環境SuperJ Engineへの組み込み
- 富士通研究所: JSPテストツールへのAOP適用 [小高07]
  - JSPから生成されるHTML/Webページでは、意味情報が消失しテスト困難
  - ⇒ JSP Webアプリに、意味情報をWebページに残すようにAspectJで自動拡張
- Motorola
  - UML 2.0/Telelogic Tauの拡張・アスペクト指向モデリング・シミュレーション環境WEAVRによる携帯電話インフラソフトウェアのデバッグ、テスト支援 [WEAVR]



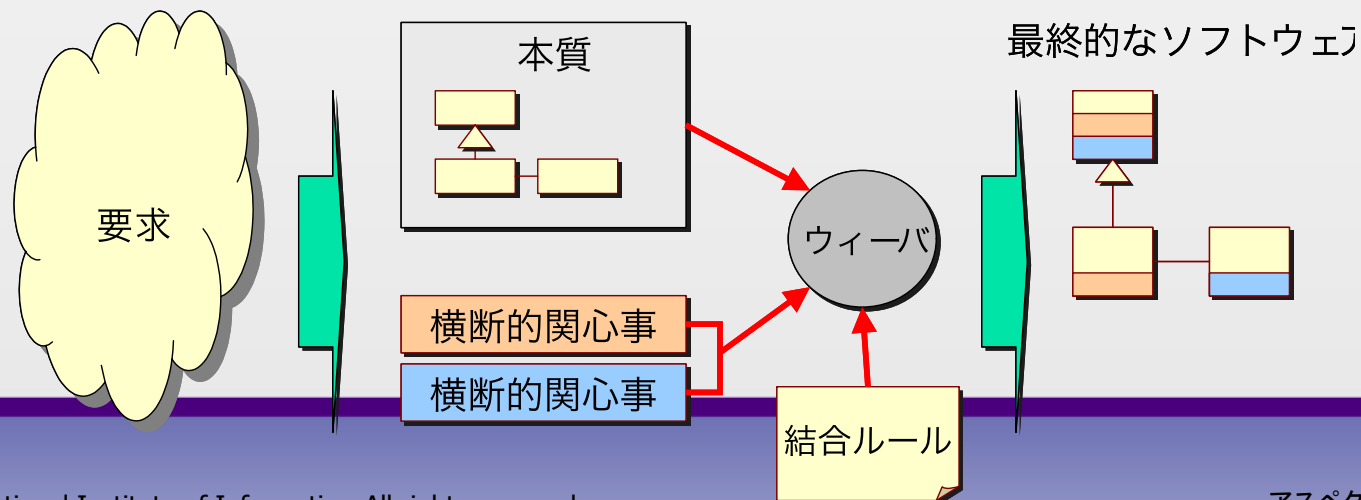
## 他の関心事の分離への活用

- Sun [Wikipedia]
  - Streamline mobile application の開発における多様性の扱いにAspectJを利用
- EXA [友野03]
  - Webアプリケーション開発における関心の分離
  - JavaServletインターセプトによる実装
  - アクセス制限、画面遷移、DBキャッシュなど



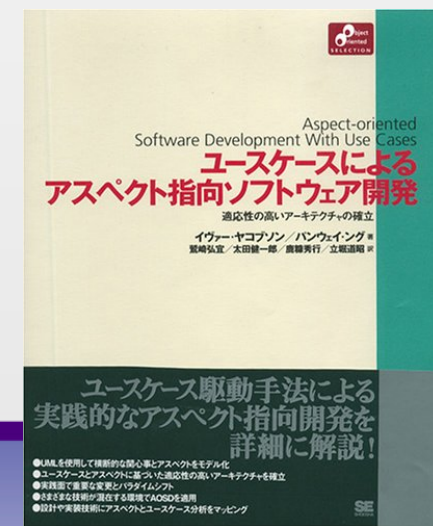
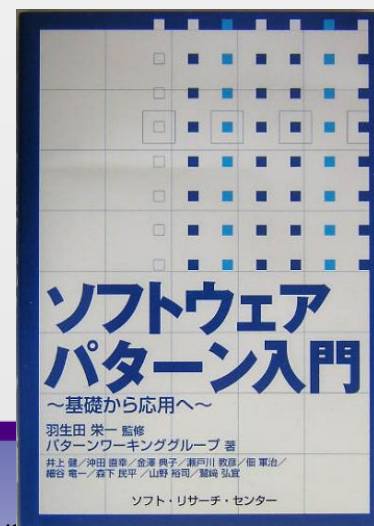
## まとめ

- アスペクト指向はオブジェクト指向がベースの“改良”パラダイム
  - しかし、その効果はオブジェクト指向に限らない。
  - 異なる関心事は分離して記述し、後で自動合成する
  - 活用シーンは幅広い: デバッグ、テスト、プロダクトライン開発...
- AspectJ, AspectC など実用的AOP処理系が存在
  - 標準化の動き: Aspect Alliance (Java/J2EE AOP 標準)
  - 実社会で使われ始めている: フレームワーク、デバッグ/テストツール
- AOPからAOSDへ
  - まずは「アスペクト思考」から



## 学習を深めるために

- アスペクト指向全般
  - Robert Filman, Tzila Elrad, Siobhan Clarke and Mehmet Aksit, “Aspect-Oriented Software Development,” Addison-Wesley, 2004.
  - 千葉滋, アスペクト指向入門, 技術評論社, 2005.
- アスペクト指向プログラミング
  - “AspectJによるアスペクト指向プログラミング入門”, ソフトバンクパブリッシング, 長瀬, 天野, 鷲崎, 立堀 著, 2004年
  - “ソフトウェアパターン入門”, 羽生田 監修, 金澤, 井上, 森下, 鷲崎, 佃, 細谷, 瀬戸川, 山野, 沖田 著, ソフトリサーチセンタ, 2005年7月
- アスペクト指向分析設計モデリング
  - “ユースケースによるアスペクト指向ソフトウェア開発”, I. Jacobson and P.W. Ng 著, 鷲崎, 太田, 鹿糠, 立堀 訳, 翔泳社, 2006年3月



# 参考文献

- [Brooks75] Frederick P. Brooks, The Mythical man-Month : Essays on Software Engineering, Addison-Wesley, 1975 (邦訳『人月の神話 – オオカミ人間を撃つ銀の弾丸はない』)
- [Aye04] SawSanda Aye and K. Ochimizu, "Defining Ontology for Complexity Issues in Software Engineering", Natnl Conf. of JSSST, 2004
- [Kiczales97] Gregor Kiczales, et al., Aspect-Oriented Programming, ECOOP'97 (1997)
- [Kiczales93] Gregor Kiczales, et al., Metaobject protocols, Object-Oriented Programming: The CLOS Perspective (1993)
- [Kiczales97] Gregor Kiczales, et al., Open Implementation Design Guidelines, ICSE (1997)
- [AspectJ] Eclipse Consortium, <http://eclipse.org/aspectj/>
- [AspectC] D. Gao et al., <http://www.aspectc.net/>
- [JW0601] "ジャバワールド 2006年1月号", IDGジャパン, 2006.
- [Jacobson06] I. Jacobson and P.W. Ng 著, 鷲崎, 太田, 鹿糠, 立堀 訳, "ユースケースによるアスペクト指向ソフトウェア開発", 翔泳社, 2006年3月
- [児玉04] 児玉公信, "UMLモデリングの本質", 日経BP社, 2004.
- [長瀬04] 長瀬嘉秀, 天野まさひろ, 鷲崎弘宜, 立堀道亜昭, "AspectJによるアスペクト指向 プログラミング入門", ソフトバンク パブリッシング, 2004.
- [千葉05] 千葉 滋, "アスペクト指向入門", 技術評論社, 2005.
- [長谷川05] 長谷川裕一, 伊藤清人, 岩永寿来, 大野渉, "Spring入門", 技術評論社, 2005.
- [Bugdel] <http://www.csq.is.titech.ac.jp/projects/bugdel/>
- [小高07] 小高, 上原: "アスペクト指向を利用したWebアプリケーションテストの自動化", 情報処理学会155回ソフトウェア工学研究会, 2007
- [友野03] 友野: "アスペクト指向開発適用記", exa review, 2003

# 参考文献(つづき)



- [Filman00] Filman, R. and D. Friedman. "Aspect-oriented programming is quantification and Obliviousness." Proceedings of the Workshop on Advanced Separation of Concerns, in conjunction with OOPSLA'00 (2000)
- [Yu04] Yijun Yu, Julio Cesar Sampaio do Prado Leite, John Mylopoulos: From Goals to Aspects: Discovering Aspects from Requirements Goal Models. RE 2004: 38-47
- [Baniassad04] Elisa Baniassad and Siobhán Clarke, "Theme: An Approach for Aspect-Oriented Analysis and Design," pp.158-167, 26th International Conference on Software Engineering (ICSE'04), 2004
- [Noda08] Noda and Kishi, Aspect-oriented Modeling for Variability Management, SPLC 2008
- [WEAVR]Cottenier, T. Motorola WEAVR: Aspect-Oriented Modeling for Simulation and Code Generation. Tutorial, ECOOP'07, 2006
- [Wikipedia] Wikipedia: Aspect-Oriented Software Development, [http://en.wikipedia.org/wiki/Aspect-oriented\\_software\\_development](http://en.wikipedia.org/wiki/Aspect-oriented_software_development) (2009/9/2)
- [久保08]久保淳人ほか, "AOJS: アスペクトを完全分離記述可能なJavaScriptアスペクト指向プログラミング・フレームワーク", 日本ソフトウェア科学会第15回ソフトウェア工学の基礎ワークショップ論文集「ソフトウェア工学の基礎 15」, 近代科学社, 2008.
- [Washizaki09] Hironori Washizaki, et al., "AOJS: Aspect-Oriented JavaScript Programming Framework for Web Development", Proc. 8th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS 2009), ACM Press, 2009.