IPSJ/SIGSE パターンワーキンググループ設立記念セミナー資料 (2003.5.23)

# ソフトウェアパターン研究の現在と未来　完全版）

鷲崎 弘宜　　深澤 良彰

早稲田大学理工学部

http://www.fuka.info.waseda.ac.jp/

# 本題に入る前に

- 本発表は、日本におけるソフトウェアパターンコミュニティの草分け的存在「JapanPLoP」における議論から、幾つかを、発表者の解釈に基づいて参考としている
  - JapanPLoPは、本日 2003年5月23日をもって解散しました
  - JapanPLoPの運営に尽力された皆様、参加されていた皆様に深く感謝いたします

- 本発表は、これまでのソフトウェアパターンに関する研究事例を、発表者の解釈に基づいて紹介する
  - 紹介する研究事例の実践者とは異なった解釈のもとに紹介してしまう可能性があることにご注意願いたい
  - あくまで、過去にこういう研究事例があったという「雰囲気」と、今後の「方向性」をお伝えできれば幸い
  - さらに、興味を持っていただければ、是非、本ワーキンググループ研究タスクにて、議論に加わっていただきたい

# ソフトウェアパターンとは

- 最大公約数的定義:
  - ソフトウェア開発の各局面で繰返し現れる問題に対する解法 •指針
- ソフトウェアパターンの歴史
  - GoFによるデザインパターン (1987-1994, 現在)
  - CoplienによるC++イディオム (1992)
  - CoadによるOODパターン集 (1992)
  - 以降、現在にいたるまで様々なソフトウェアパターンの提案 •拡張。2001年次の調査で「1000以上のソフトウェアパターンを確認」
- ソフトウェアパターンコミュニティの発展
  - The Hillside Group
  - Hillside Europe
  - JapanPLoP
  - IPSJ/SIGSE パターンワーキンググループ

# ソフトウェアパターン研究とは

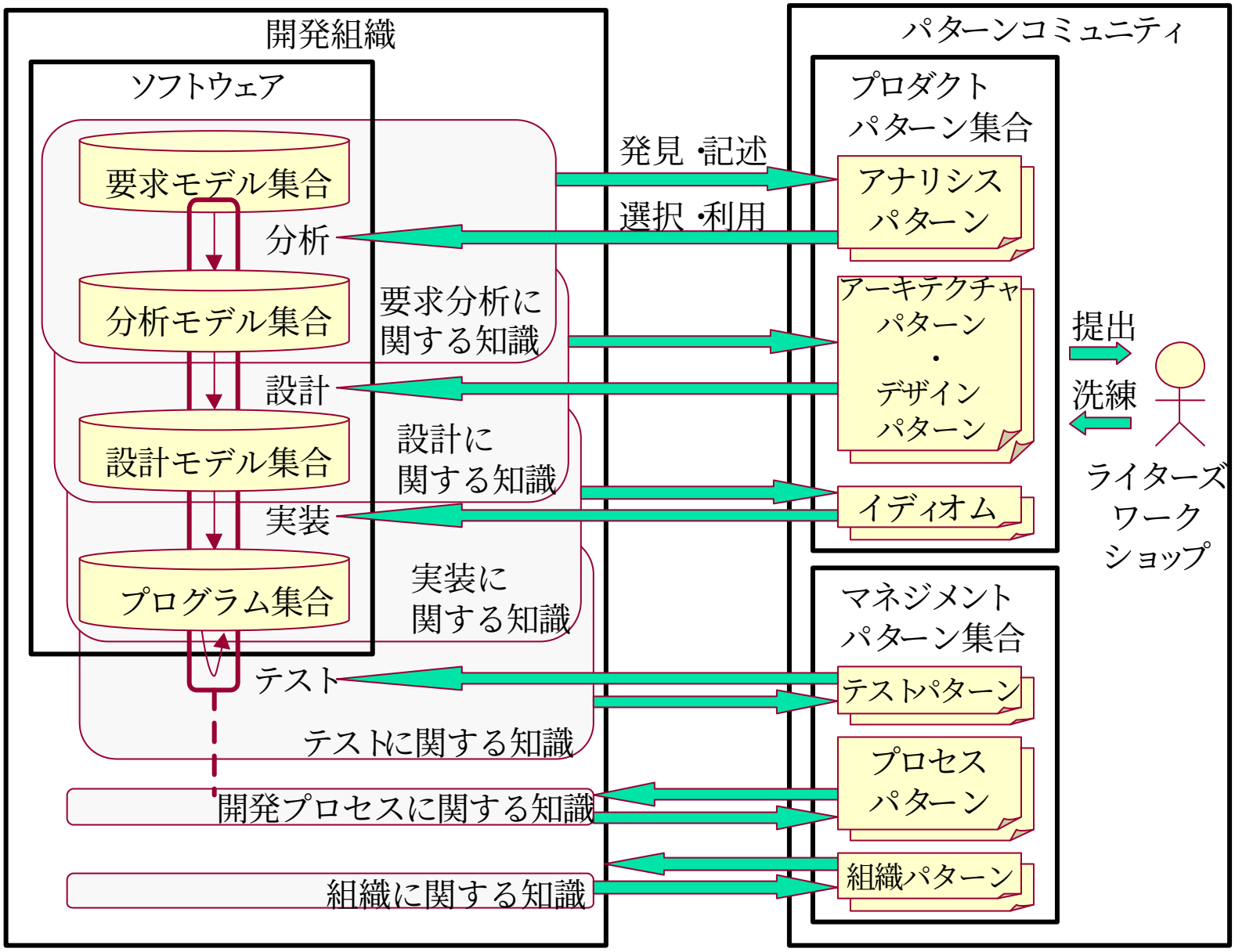
科学的な裏付けに基づいて、ソフトウェアパターンの定性的・定量的特性に関する何らかの主張を行っている研究活動

- なぜソフトウェアパターンを研究するのか？
  - 日々、ソフトウェアパターンは提案され、洗練されていく
  - ソフトウェア工学の観点からソフトウェアパターン、及び、ソフトウェアパターンを用いる活動を見直し、より良いものとしたい
  - 究極には、ソフトウェアパターンの仕組みを解明したい
    - なぜソフトウェアパターンは上手く機能するのか？
    - ソフトウェアパターンを「パターン」として認知しうる条件は何か？
    - 「無名の質（The Quality without a Name）」？
- 仮定:
  - 全てのパターン研究成果は、ソフトウェアパターンを用いた何らかの個人的・組織的活動を支援する

# パターンを用いたソフトウェア開発のあり方

- パターン指向開発:
  - ソフトウェアパターンの蓄積と再利用を重視したソフトウェア開発方法
- パターン指向開発プロセス:
  - パターン指向開発を実現するための一連の手順
    - 分析: アナリシスパターンの抽出、利用
    - 設計: アーキテクチャパターン、デザインパターンの抽出、利用
    - 実装: イディオムの抽出、利用
    - テスト: テストパターンの抽出、利用
    - (全体: プロセスパターン、組織パターンなど)
- パターンライフサイクルプロセス
  - パターン指向開発プロセスの各工程において、パターンの抽出から適用に至る一連の手順
    - 抽出: 発見 → 記述 → 提出 → 洗練
    - 利用: 選択 → 拡張 → 適用 → 評価

# パターン指向開発プロセス

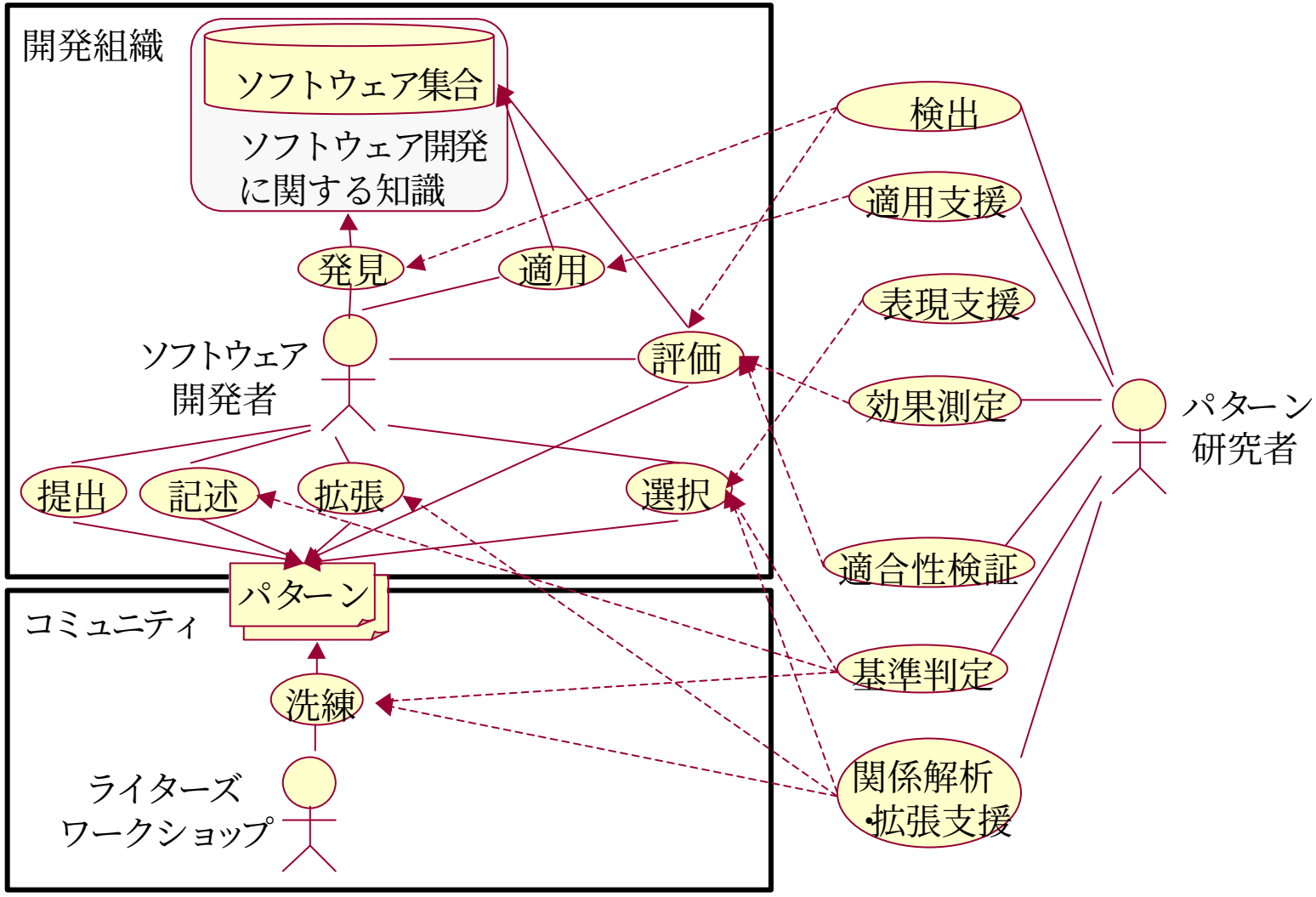


# パターン指向開発に関わる動作主体

- 開発組織
  - 活動内容: ソフトウェアパターンの発見と記述
  - 構成要素:
    - ソフトウェア開発者たち
    - ソフトウェア集合（モデル、プログラム）
    - 知識集合
    - （パターン集合）
- コミュニティ
  - 活動内容: ソフトウェアパターンの受理と洗練および管理
  - 構成要素:
    - （組織を横断した）ソフトウェア開発者たち
    - パターン集合

# パターンライフサイクルプロセス

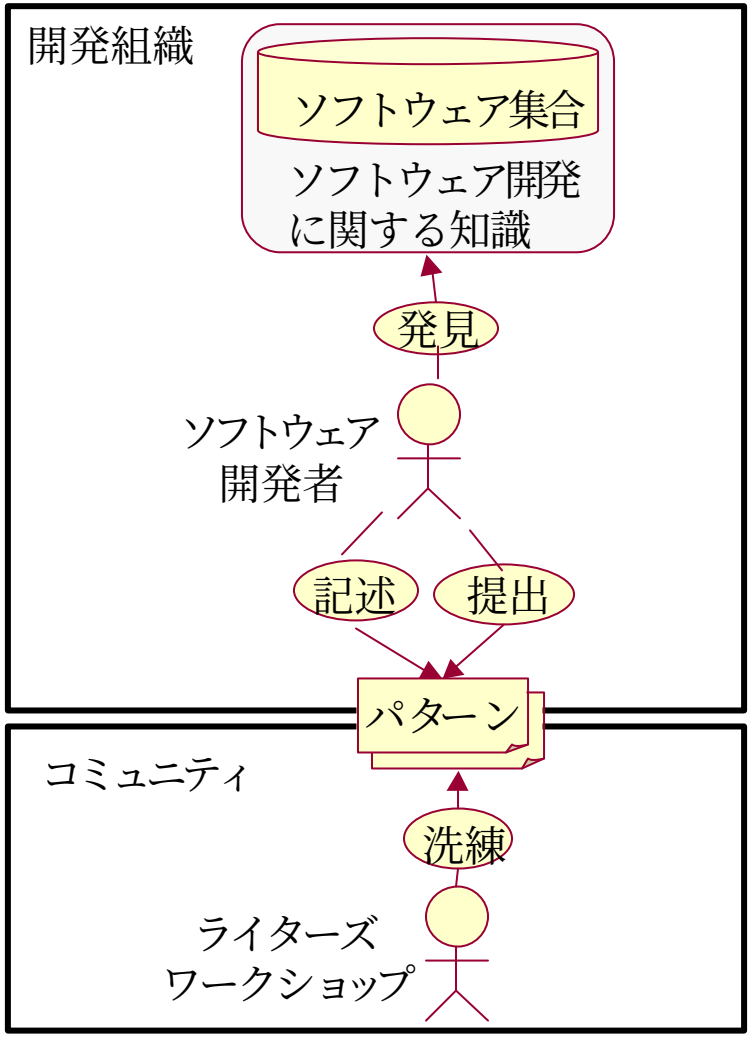
- パターンの抽出活動: 発見、記述、提出、洗練
- パターンの利用活動: 選択、拡張、適用、評価



# パターンの抽出活動

ソフトウェア開発に関する知識から、パターンとして
再利用可能な知識を発見し、記述し、提出し、洗練する

- 以下の4つのプロセス
  - パターンの発見
  - パターンの記述
  - パターンの提出
  - パターンの洗練



# 知識とソフトウェアパターン

- 知識とは？
  - 経験に基づいて理解し認識した事柄
  - 知識（保証無し）⟺知恵（保証有り）
- ソフトウェア開発に必要な知識
  - 対象とする問題領域に関する知識
  - ソフトウェアの開発方法に関する知識
  - 組織に関する知識
  - 開発プロセスに関する知識
  - 知識に関する知識　(メタ知識)
    - 例: Orders of Ignorance [Armour00]
      - { 無知、知識、自覚、手順、無知の概念 } の欠如
- ソフトウェアパターンとは？（知識の観点から）
  - ソフトウェア開発の各局面で繰返し現れる問題に対する解決としての暗黙知を、解決に至った理由が明らかな形式知にしたもの

この対応付けそのものがソフトウェア [藤野01]

[藤野01] 藤野晃延: ソフトウェア開発とパターンランゲージ, ソフトウェア・シンポジウム (2001)
[Armour00] P.G. Armour: The Five Orders of Ignorance, CACM, 43(10) (2000)

# パターンの発見

ソフトウェア開発知識から、繰り返し出現する問題と解決方法、および、解決に至った理由を特定する

- 考慮すべき事柄
  - **問題**: 繰り返し出現する共通性 ⟺ 個別性
  - **解決**: 再利用可能な程度の抽象度 ⟺ 適用範囲の大きさ
- 発見プロセスの支援
  - ソフトウェア開発・解析技術の利用
  - 対話や教育に基づく組織的パターン発見の試み [Rising99]
    - インタビュー、教育、講義、ワークショップ、会議、経験
  - **パターンの検出手法**の利用　後ほど詳しく
    - 既にどのようなパターンが適用されているか？

[Rising99] L. Rising: Patterns Mining, in Handbook of Object Technology, CRC Press (1999)

# パターンの記述

発見したパターンを、特定の形式に従って記述する

- 形式: GoF Form, Coplien Form, Alexandrian Formなど
  - 名前: 共通の語彙となるもの
  - 問題: 解決すべき課題
  - 状況: パターン適用前の周囲の環境
  - フォース: 解決に至る際に考慮した事柄
  - 解決: フォース間のバランスを考慮した解決策・手順

  （問題, 状況: 文脈）
- 記述プロセスの支援
  - **パターンの基準判定法**の利用
    - 記述した（している）パターンは、パターンとしての体をなしているか？
    - (例1) 表現上の基準: 3つ組 [Gabriel]
      - 文脈, フォースの体系, 解決を含む）ソフトウェア構成
    - (例2) 利用実績上の基準: Rule of Three [Tracz88]
      - 独立に3回以上は利用されているべきである

[Gabriel] : R. Gabriel: A Timeless Way of Hacking, in Core J2EE Patterns, Pearson Education

# パターンの提出

記述したパターンを、しかるべき場所に提出する

- しかるべき場所？
  - パターンのレビューを行う組織内活動
  - ライターズワークショップを開催するパターンコミュニティ
- 提出プロセスの支援
  - ソフトウェア構成管理ツール
    - 例: RCS, CVS, VSS
  - クリアリングハウス（clearing house）
    - 資料およびデータを収集し、保存し、報知し、利用できるようにする機関 [JIS]
    - 例: SourceForge.net http://sourceforge.net/
      - CVSリポジトリ, FTP, バックアップ, バグ追跡システム
      - Webサイト, ML, BBS

[JIS] 日本工業規格: JIS X 0701 ドキュメンテーション用語（基本概念）(1989)

# パターンの洗練

提出されたパターンの品質を、レビューもしくはライターズワークショップによって高める

- ライターズワークショップとは？
  - コミュニティの同意を得る
  - 「集団による詩作」
  - 集団による改善（NOT あら探し）

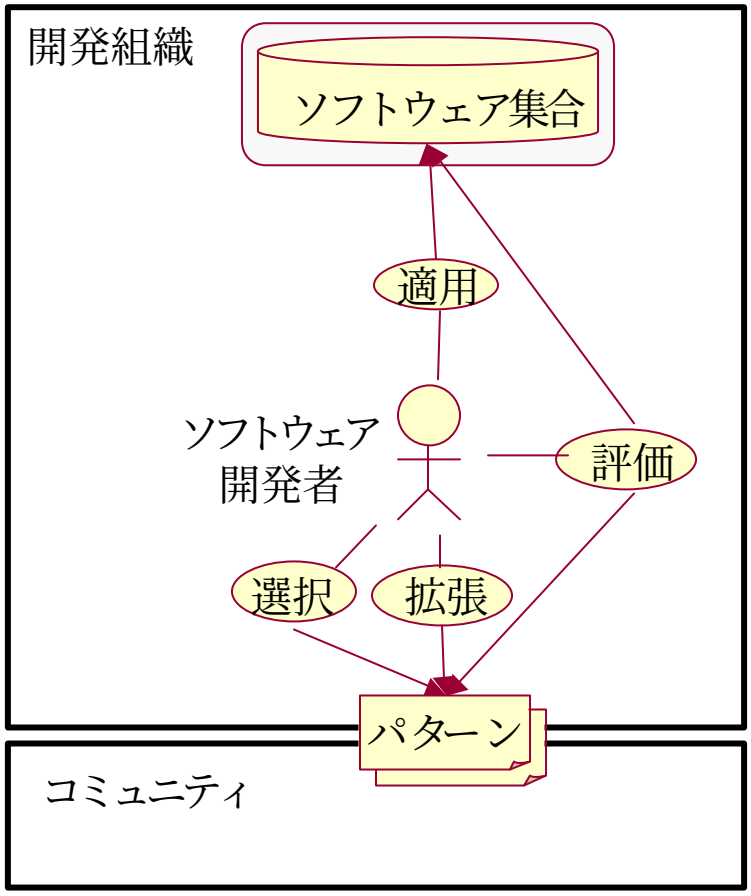
MensorePLoP2001の様子
（撮影: 瀬戸川さん）

- 洗練プロセスの支援
  - **パターンの拡張・関係解析手法**の利用　後ほど詳しく
    - 表記された関連パターンは妥当か？抜けはないか？
  - **パターンの基準判定法**の利用
    - 記述した（している）パターンは、パターンとしての体をなしているか？

# パターンの利用活動

既存のパターン集合から、直面する文脈に適合するパターンを選択し、修正・拡張し、適用し、適用結果を評価する

- 以下の4つのプロセス
  - パターンの選択
  - パターンの拡張
  - パターンの適用
  - パターンの評価



# パターンの選択

既存のパターン集合から、利用者が直面する問題と状況に合致するパターンを検索して選択する

- 選択手順 [Buschmann00]
  - 問題の定義 → パターンカテゴリの選択 → 問題カテゴリの選択 → 問題の検討 → 利点と欠点の検討 → バリエーションの検討
- 選択プロセスの支援
  - **パターンの表現支援手法**の利用　後ほど詳しく
    - これは一体どのようなパターンなのか？
  - **パターンの拡張・関係解析手法**の利用　後ほど詳しく
    - 他にはどのようなパターンがあるのか？その関係は？
  - **パターンの基準判定法**の利用
    - これはパターンとして妥当か？

[Buschmann00] : F. Buschmann et al.著 金澤 他訳: ソフトウェアアーキテクチャ, 近代科学社 (2000)

# パターンの拡張

選択したパターンを、利用者が直面する状況にあわせて
カスタマイズ（修正・拡張）する

- カスタマイズの方法
  - パターンの新バリエーションを作成する
  - 複数のパターンを組み合わせる
- 拡張プロセスの支援
  - **パターンの拡張・関係解析手法**の利用　後ほど詳しく
    - 複数のパターンをどのように組み合わせればよいのか？

# パターンの適用

得られたパターンを、ソフトウェア、もしくは、開発プロセス・組織に適用する

- 適用の方法
  - パターンをよく読んで解決に至る理由と解決手順を理解し、実行する
  - 「パターンはものであり、プロセスである」
    - パターンは、問題から解決に至る道筋を表す
- 適用プロセスの支援
  - **パターンの適用支援手法**の利用　後ほど詳しく
    - パターンをどの部分にどのように適用すればよいのか？

# パターンの評価

パターンを適用した結果を調査し、適用したパターンと適用方法の妥当性を評価し、パターン活動を改善する

- 評価の方法
  - 適用して得られた状況と、適用したパターンが主張している適用結果の比較
  - パターンの効果の定量的評価
  - 無名の質」？
- 評価プロセスの支援
  - **パターンの効果測定法**の利用　後ほど詳しく
    - 適用することで、適用対象にどのような効果が現れたか？
  - **パターンの検出手法**の利用　後ほど詳しく
    - 得られたソフトウェアは、本当にパターンを検出した結果なのか？
  - **パターンの適合性検証手法**の利用　後ほど詳しく
    - 適用して得られた結果は、パターンが主張する適用結果に合致するか？

# パターン指向開発支援手法

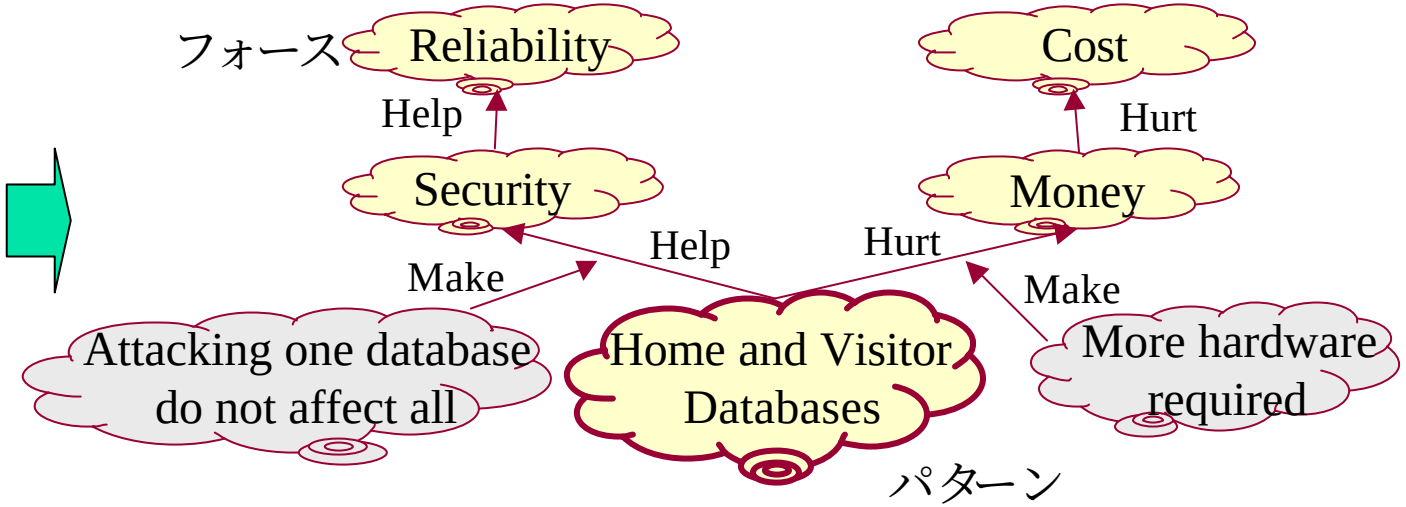
- パターンの検出
- パターンの表現支援
- パターンの拡張、関係解析
- パターンの適合性検証
- パターンの適用支援
- パターンの基準判定
- パターンの効果測定

# 表現支援手法

パターンそのものとは異なる表現方法を与えることで、パターンの理解を促進する

- UMLを拡張したロール・タイプ・クラスモデル [Lauder98]
  - GoFデザインパターンの動的・静的特徴の記述
- フォース階層によるパターン(ランゲージ)理解支援 [Ong03]
  - NFRフレームワークの利用 (Non-Functional Requirements)
    - 要求工学におけるゴール指向分析
  - パターンやフォースの間の正負の関係を図示

名前: Home and Visitor Databases
問題: ・・・
状況: ・・・
フォース: ・・・
解決: ・・・
適用後の状況: ・・・
例: ・・・



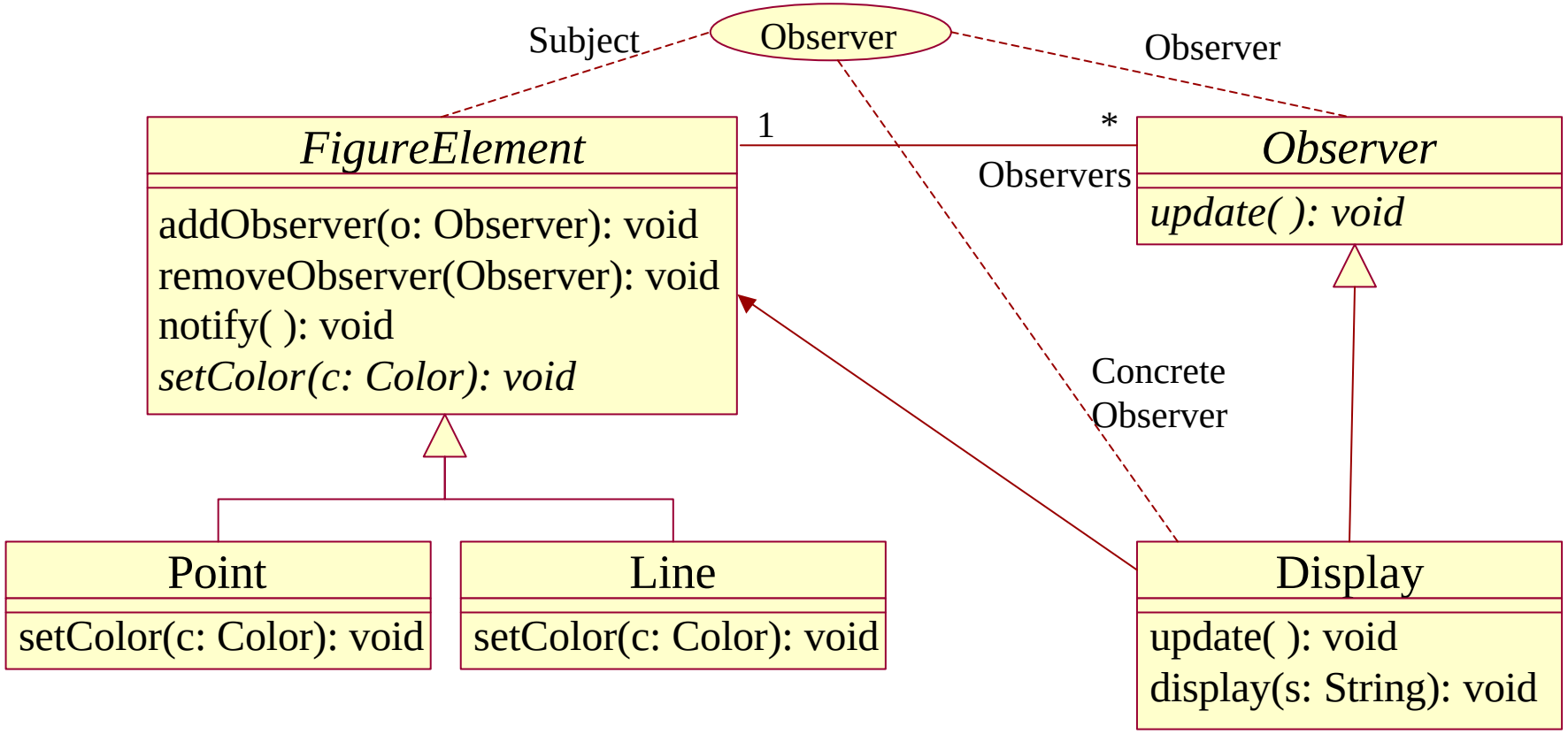
# 適用支援手法

幾らかの情報（パラメータ）入力に基づいて、既存のモデルもしくはプログラムへのパターンの適用を半自動化する

- ツールを用いた既知のデザインパターンの適用支援
  - デザインパターンのHTML, SGML出力とプログラム生成
  - 既存モデル・プログラムへの登録済みパターンの適用
  - ※商用・非商用ツールのサポートによる実用化の流れ
    - 例: Pattern Support for Eclipse （Eclipseプラグイン）
- 特別な言語を用いたGoFデザインパターンの適用支援
  - デザインパターンの関心 ⟺ 適用前のプログラムの関心
  - デザインパターンを独立して記述し、コンパイル時に適用
    - アスペクト指向言語処理系: AspectJ [Hannemann02]
    - マクロ機構: OpenJava [Tatsubori98]
    - 差分ベースモジュール言語処理系: MixJuice [田中03]

# 適用支援手法の例

- オブジェクト指向言語を用いたGoFデザインパターン適用
  - 例: Observerパターン
    - Subject役の状態変化をObserver役に通知する
    - Pushモデル？Pullモデル？



# 適用支援手法の例（つづき）

- AspectJ を用いたGoFデザインパターンの記述と適用
  - 例: Observerパターン [Hannemann02]

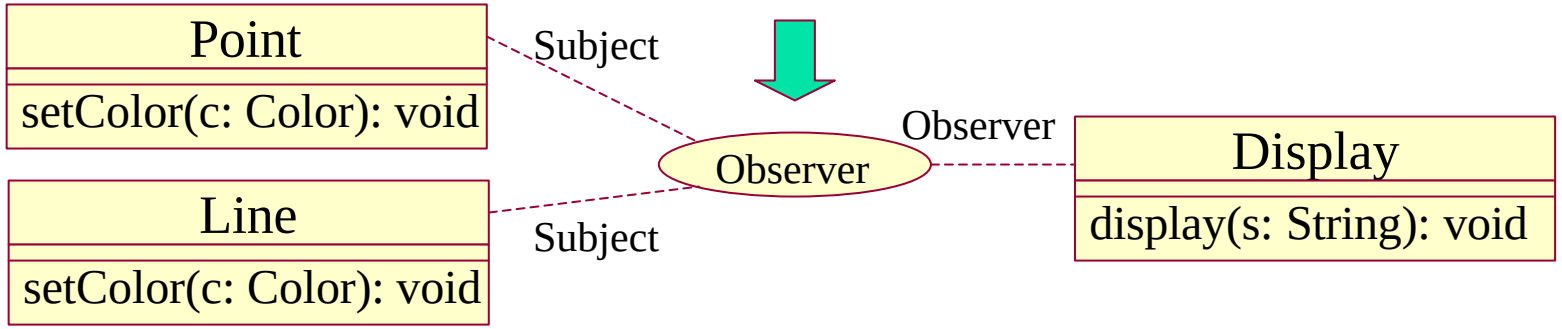
```
public abstract aspect ObserverProtocol {
  protected interface Subject { }
  protected interface Observer { }

  abstract protected pointcut subjectChange(
    Subject s);
  abstract protected void updateObserver(
    Subject s, Observer o);

  after(Subject s): subjectChange(s) {
    Itretor iter = …;
    while(iter.hasNext()) updateObserver(…);
  } … }
```

```
public aspect ColorObserver extends ObserverProtocol{
  declare parents: Point implements Subject;
  declare parents: Line implements Subject;
  declare parents: Display implements Observer;
  protected pointcut subjectChange(
    Subject s):
    (call (void Point.setColor(Color))
   || call (void Line.setColor(Color)) && target(s);

  protected void updateObserver(Subject s,
    Observer o) {
    ((Display) o).display("Color change.");
  }  }
```



# 適合性検証手法

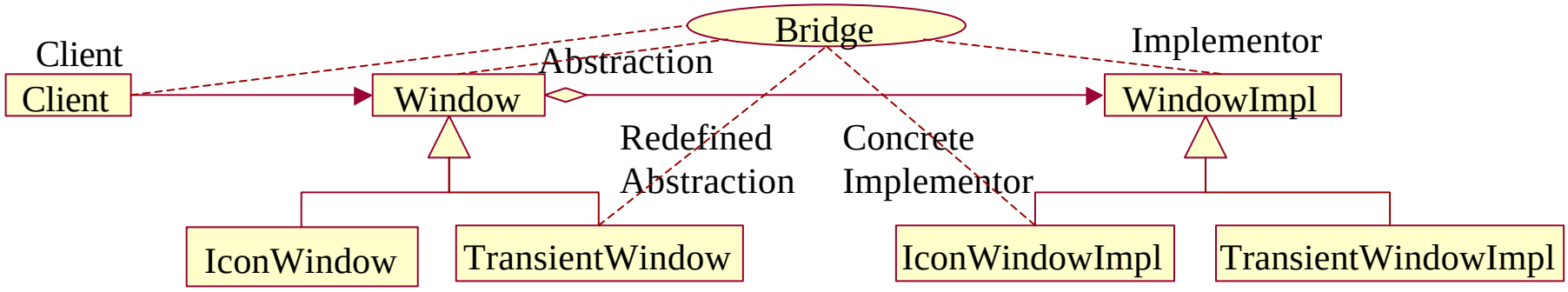
パターンを形式的に記述し、パターンを適用して得られた結果が、適用したパターンの制約を満たすかを検証する

- GoFデザインパターン適用結果の実行履歴に基づく協調関係検証 [畑口00]
  - メッセージ送受信の順序の検証
  - 「このパターンを適用しているのなら、こうなっているはず」

- 参考）「パターンの形式化は役に立たない」[Buschmann00]
  - 問題の形式化はパターンの理解を困難とし適用範囲を限定する
  - 本当に役に立たない？

# 適合性検証手法の例

- GoFデザインパターン適用結果の形式的記述の検証 [Cechich00]
  - オブジェクト間のメソッド呼出し順序は想定通りに適切か
  - デザインパターン参加要素の特定ロールが出現するか、など
- 形式化の利点
  - 当たり前の事を明らかにして、きちんと書いて確かめる

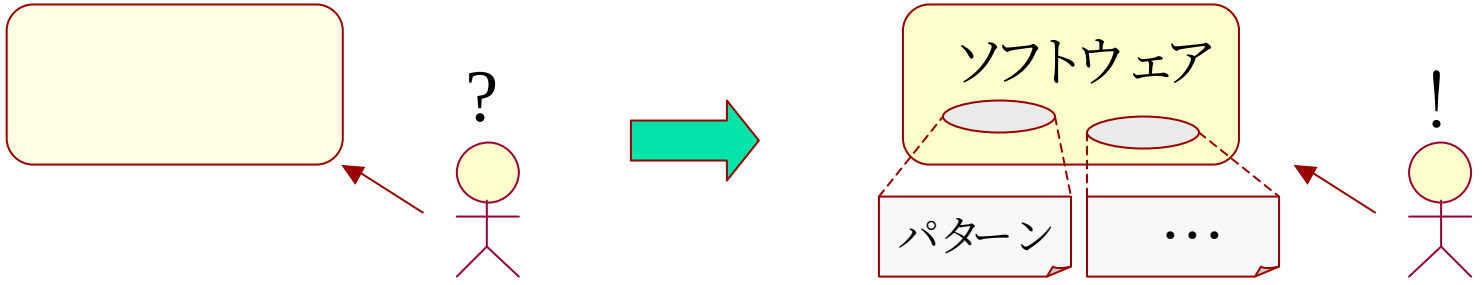



```
// Bridgeパターンの参加要素
type
  Role_Type == abstraction | implementor | refined_abstraction | concrete_implementor | client
value
  role_type: G.Role → Role_Type,
  exist_one_abstraction : PS.WF_Pattern_Structure → Bool
  exist_one_abstraction(psc, psr)≡(∃! c: C.Wf_Class •c∈psc∧(role_type(C.p_role(c)) = abstraction))
```

# 検出手法

既存のソフトウェアについて、既知のパターンが適用されているかどうかを検出し、理解を支援する（理由の推測手掛り）

- GoFデザインパターンの事前記述に基づく検出
  - 静的特徴に基づくGoFデザインパターン検出 [Kramer96] [金子98] [Wuyts98] [Keller99] [Amiot01]
    - 検出対象プログラム：Java, C++, Smalltalk
    - デザインパターン記述形式: 既存プログラム言語の拡張形式、UMLクラスモデル、論理型言語
  - 静的＋動的抽出に基づくGoFデザインパターン検出 [Heuzeroth03]
    - 検出対象プログラム：Java
- ソフトウェア場の定常状態に基づく検出 [Kambayashi03]
- GoFデザインパターン間の関連検出 [高橋03]

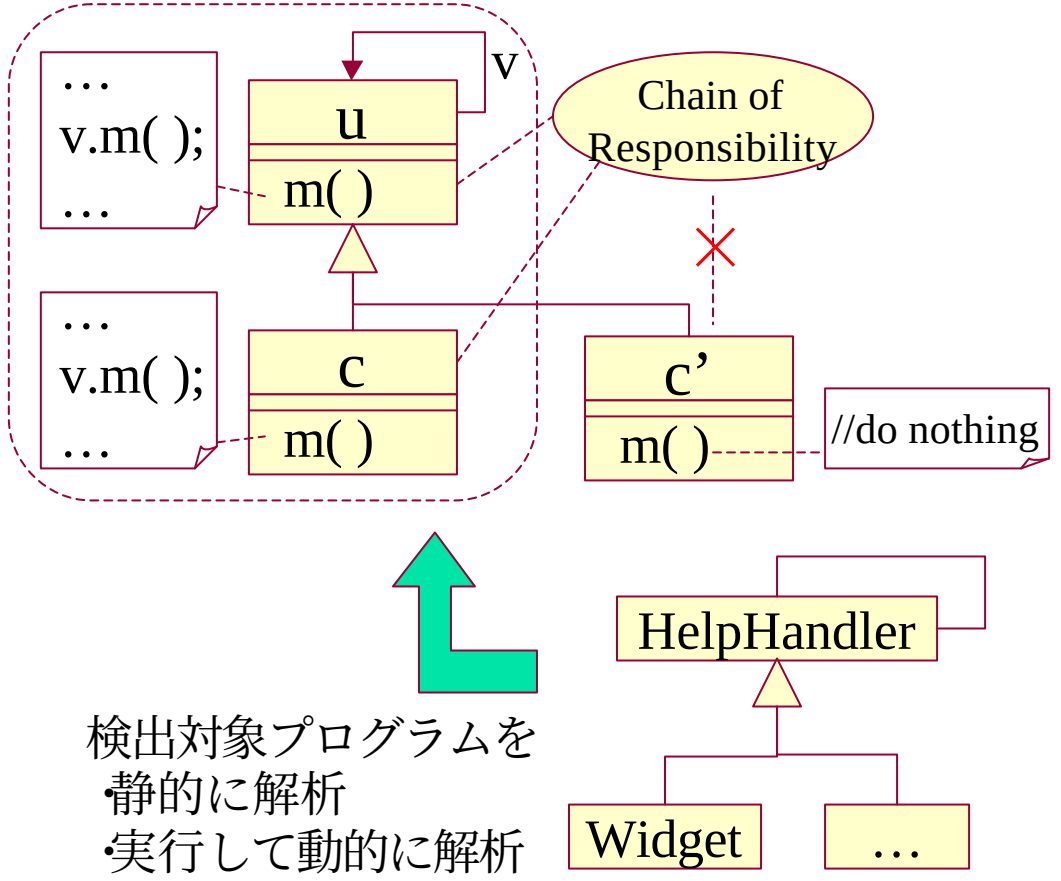


# 検出手法の例

- 静的＋動的特徴に基づくデザインパターン検出[Heuzeroth03]
  - 例: Chain of Responsibilityパターンの検出
    - 静的特徴: 以下の最小構造例に関する静的解析アルゴリズム
    - 動的特徴: 用いる実行時の型が、m中でv.mするクラス

```
// 静的解析アルゴリズム
C ← φ
for each class c do
  Y ← φ
  for each variable v in c do
    for each class u ← class(v) do
      if c = u ∨ c <: u
        Y ← Y ∪ { (c,v,u) }

for each (c,v,u) ∈ Y do
  for each method m in u do
    if "… v.m …" in body(m) in c
      C ← C ∪ { (c,v,u,m) }
```



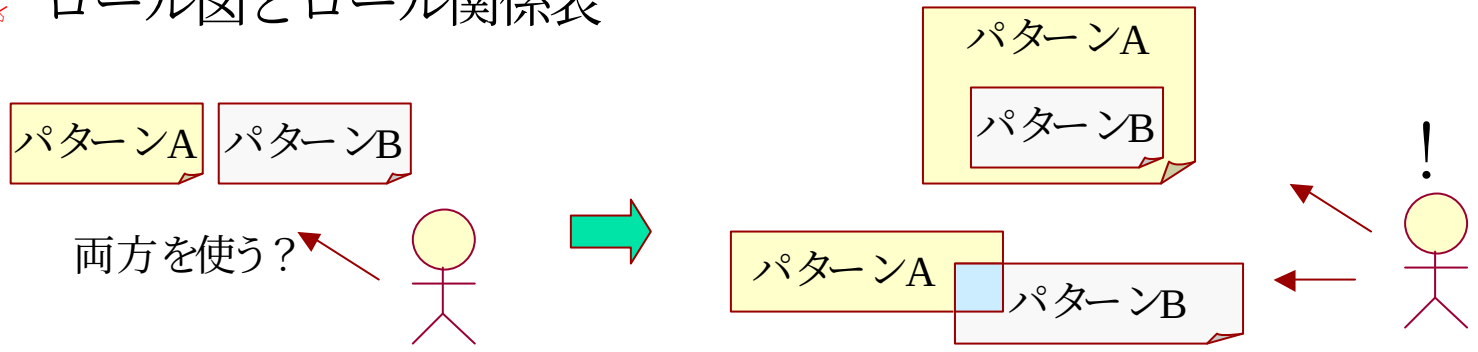
# (余談) ソフトウェアパターンと形式化

- 参考）「パターンの形式化は役に立たない」[Buschmann00]
  - 問題の形式化はパターンの理解を困難とし適用範囲を限定する
  - 本当に役に立たない？
- 形式化の利点
  - 当たり前の事を明らかにして、きちんと書いて確かめる

# 拡張手法 •関係解析手法

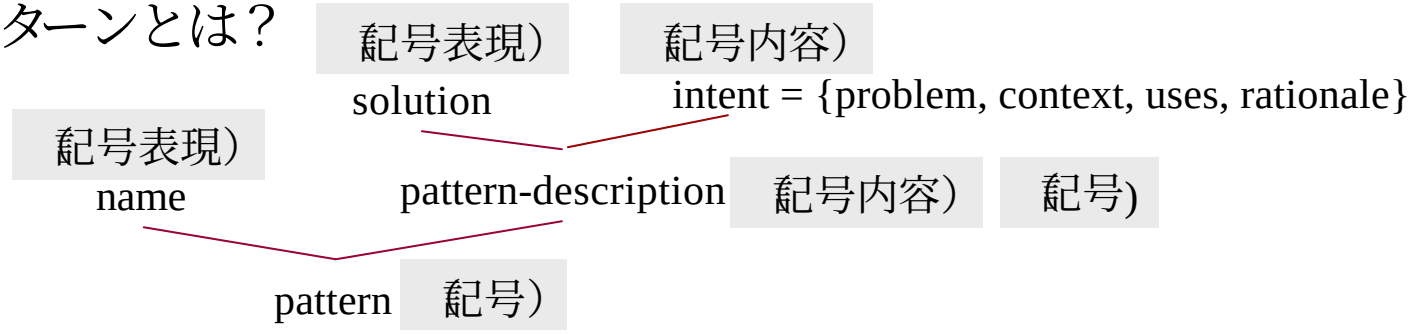
複数のパターン間の関係を導出して、パターンの組み合わせの妥当性を確認した後に、組み合わせて適用する

- パターンの進化パターンに基づく組み合わせ [青山99]
  - マイクロ進化パターンとマクロ進化パターン
- GoFデザインパターンの形式化と関係解析 [Eden02]
  - パターン構造例のクラス階層、クラス、メソッドなどの関係記述
    - Assignment, Creation, Invocation, Inheritance 等の利用
  - Multicast – Observer ≠ Typed Message
- ロールモデルに基づくGoFデザインパターン表現 [Riehle97]
  - オブジェクト毎に持つロールの違い
  - ロール図とロール関係表

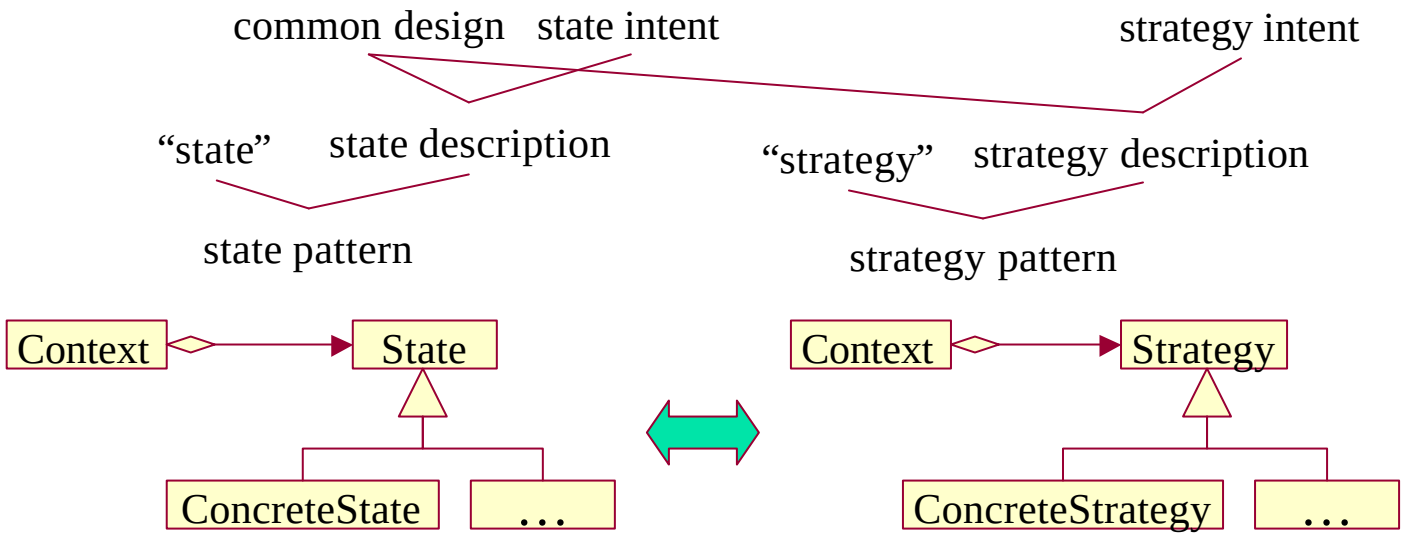


# 拡張 •関係解析手法の例

- 記号としてのパターンとパターン間の関係解析 [Noble02]
  - 対話=記号の交換、記号=表現+内容　例: 赤=“赤”+赤の概念）
  - パターンとは？




  - 例: StateパターンとStrategyパターンの関係
    - name と intent が異なり、solution は同一



# 基準判定法

パターンが、パターンとしての体をなしているかどうかの判定

- 表現上の基準: 3つ組 [Gabriel]
  - 文脈
  - フォースの体系
  - 解決を含む）ソフトウェア構成
- 利用実績上の基準: Rule of Three [Tracz88]
  - 独立に3回以上は利用されているべきである

[Gabriel] : R. Gabriel: A Timeless Way of Hacking, in Core J2EE Patterns, Pearson Education

# 効果測定法

適用結果から、パターンの有効性を定量的に評価する

- メトリクスを用いたデザインパターンの効果測定 **[増田00]**
- 修正作業でのデザインパターンの効果測定実験 **[Prechelt01]**
  - パターンが適用されたC++プログラム版と、素朴な設計がなされた版について、同じ修正を行うのにかかる作業時間比較
    - 4種類のプログラム。各種類ごとに両版は同一機能を実現。
  - 結果:
    - パターンが複雑な設計をもたらしているものについて、パターンの知識が不足している場合は（素朴な版よりも）作業時間が多くかかった
    - パターンによって設計中で機能が分離されているものについて、（素朴な版よりも）作業時間が短かった。
  - 考察:
    - 素朴な設計で解決できるならば、パターンを用いる必要は無い
    - 時としてデザインパターンは有功であったので、素朴な設計を用いることに確信を持てないときは、デザインパターンを用いるとよい
    - デザインパターンを**十分に理解**することで、パターンが適用されたプログラムの理解が深まる　**逆も然り: 理解が不十分だと。。。）**

# これからのソフトウェアパターン研究

- GoFデザインパターン以外への展開
  - これまでの研究事例の大部分はGoFデザインパターンに偏っていると感じる
  - より粒度の粗いパターンへの展開
  - プロダクトパターンへの展開 → マネジメントパターンへの展開
- ソフトウェアパターンの定量的評価
  - 形式化・メトリクスと合わせた研究のあり方
- ソフトウェアパターン、人、パターンコミュニティの捉え方
  - 知識とソフトウェアパターン
  - 生き生きとした活動に共通するもの:
    - アジャイルな開発, オープンソース
    - 利用者の参加？主役？）
- 新しいパターン
  - パターンを書く
  - パターンそのものは研究となるのか？

# 研究タスクの活動内容案

- ソフトウェアパターン関連研究の発表と議論
  - ソフトウェアパターン研究内容の洗練、公開、実用化へ
  - 実践タスクへの積極的な関与、現状調査
- パターン関連ツールの調査と公正な評価
  - デザインパターン適用ツール等
- 学術的研究の継続的な動向調査 など
  - Web上で継続的な更新、「ここを見れば、いつでも最新」