ソフトウェアパターン研究の現在と未来

鷲崎弘宜[†]　深澤良彰[†]

本稿では，ソフトウェアパターンに関するこれまでの研究動向について述べ，研究成果とパターン指向開発プロセスの関連付けを行う．さらに，ソフトウェアパターン研究の今後について述べる．

Present and Future Prospects of Research on Software Patterns

Hironori Washizaki[†] and Yoshiaki Fukazawa[†]

In this paper, we outline the research trend in terms of software patterns, and associate the research results with the pattern-oriented software development process. Moreover, we present the future prospects of the research on software patterns.

1. はじめに

ソフトウェアパターンとは，ソフトウェア開発の各局面で繰り返し現れる問題に対する解法・指針である．ソフトウェアパターンの歴史は，Beck と Cunningham によるオブジェクト指向言語 Smalltalk を用いた GUI（Graphical User Interface）設計のためのパターン言語の提案[1] にまで遡ることができる．その後，Gamma による Smalltalk を用いた GUI フレームワーク設計に基づくデザインパターンの発見と提案[2] および Helm, Johnson, Vlissides（GoF: Gang of Four）らとまとめたデザインパターンカタログの提案[3),4] や，Coplien による C++ 言語を用いたプログラミングイディオムの発見と提案[5]，Coad によるオブジェクト指向設計のためのパターンの発見と解説[6] などを経て，ソフトウェアプロダクトおよびソフトウェア開発プロセス・組織に関する様々な粒度におけるソフトウェアパターンの発見と提案・拡張がなされてきた．同時に，ソフトウェアパターンの蓄積と普及・発展を目的として，The Hillside Group[7] や Hillside Europe, JapanPLoP[8),9] などのパターンコミュニティが形成されてきた．ソフトウェアパターンとパターンコミュニティの発展の経緯については文献[10),13] が詳しい．

ソフトウェアパターンが数多く提案されるに従い，提案済みのソフトウェアパターンを題材としたソフトウェア工学の見地からの研究が数多く行われてきている．具体的には，ソフトウェアパターン間の関連と分類，ソフトウェアパターンの利用支援環境，ソフトウェアパターンの表現方法，および，ソフトウェアパターンの定性的・定量的効果などについて研究が行われてきている．複数の研究事例をまとめて報告する文献としては，文献[11),12),14] などがある．

本稿では，これまでにソフトウェアパターンを対象として行われてきた研究の幾つかを示し，分類し，従来の研究成果とパターン活動の関係について述べる．さらに，ソフトウェア開発の現状に関する考察に基づき，ソフトウェアパターン研究について将来期待される事柄について述べる．

本稿で対象とするソフトウェアパターン研究とは，科学的・技術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を指すものとする．

2. パターン指向開発プロセス

ソフトウェアパターンの蓄積と再利用を重視するソフトウェア開発方法をパターン指向開発と呼び[22)]，同開発を実現する一連のプロセスをパターン指向開発プロセスと呼ぶ．パターン指向開発プロセスは，パターンの抽出から適用へ至る一連のプロセスであるパターンライフサイクルプロセス[12] を内包する．パターン指向開発プロセスのモデルを**図 1** に示す．

2.1 動作主体

パターン指向開発プロセスにおける動作主体は，ソフトウェア開発を行う開発組織と，パターンの洗練と管理を行うパターンコミュニティである．

† 早稲田大学理工学部
School of Science and Engineering, Waseda University

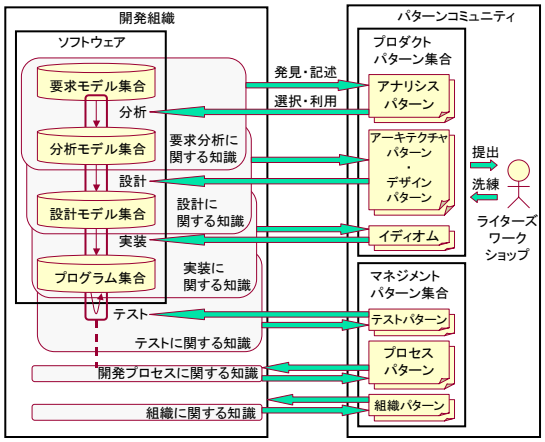

図 1　パターン指向開発プロセスのモデル

- 開発組織

開発組織は，ある開発要求について要求モデルを作成し，要求モデルを分析して分析モデルを作成し，分析モデルから設計モデルを設計し，設計モデルからプログラムを実装し，プログラムについてテストを行う．これらの分析・設計・実装・テストの各プロセスは全て，開発組織が持つ知識を用いて実施される．得られる成果物としてのソフトウェア（モデルやプログラム）は，そのようなソフトウェア開発のための知識の一部を表したものである．さらに組織は，一連の開発プロセス全体に関する知識や，組織そのものの形成に関する知識も持つ．

ソフトウェアパターンとは，成果物としてのソフトウェアそのものに限らず，これらのソフトウェア開発に関する知識から，再利用可能な文脈・フォース体系・解決の 3 つ組を発見して，組織内において同意を得て，記述したものである．フォースとは，パターンの解決に至る際に考慮した制約である．

得られるパターンの種類は，発見対象とする知識の種類に依存する．例えば，設計に関する知識を対象とする場合は，主にアーキテクチャパターンおよびデザインパターンを発見できる可能性がある．ここで，設計に関する知識とは，設計モデル集合そのもの，および，分析モデルを設計モデルへ変換するための知識を指す．

- コミュニティ

パターンコミュニティは，複数の開発組織を横断して，あるいは，開発組織から独立して，パターン指向開発プロセスに関わる人々によって形成される共同体である．パターンコミュニティは，ある組織において記述されたパターンを，同組織に限らず広く共有するための環境を提供する．具体的には，記述されたパターンを受理し，ライターズワークショップ[37] もしくはパターンについてレビューを行う場を設けて，受理したパターンを洗練し，パターンを管理し公開する．

2.2　パターン活動

パターン指向開発プロセスにおけるパターン活動は，パターンの抽出活動とパターンの利用活動の 2 つの活動から構成される．両活動を構成するプロセスを**図 2** に示す．

パターンに関する研究成果は全て，パターン指向開発プロセス中の何らかの活動を支援することができると考え，どのような活動を支援するかという観点から，種々のパターン研究成果を整理することができる．以下において，抽出活動・利用活動を構成するプロセスの詳細と，各プロセスを支援可能と考えられるソフトウェアパターン研究成果として提案される手法を合わせて示す．

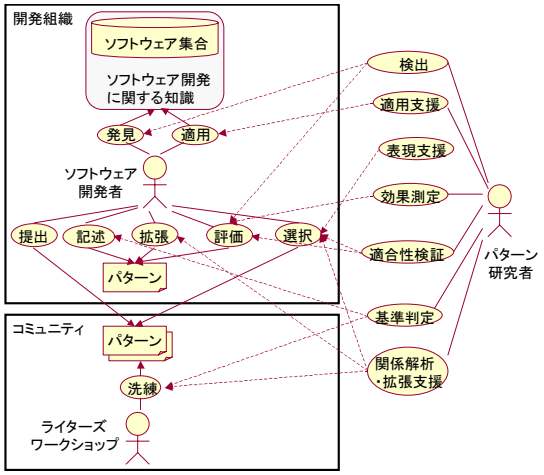

図 2　パターン活動の概要

(1)　パターンの抽出活動

抽出活動とは，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，特定の形式にしたがって記述し，組織内または組織外のコミュニティにパターンを提出し，コミュニティにおけるワークショップを経た洗練を受ける活動である．パターンの抽出活動は一般に，次にあげる 4 つのプロセスを順に実行することで達成される: 発見・記述・提出・洗練.

- パターンの発見

パターンの発見プロセスは，パターンの抽出対象とするソフトウェアと同ソフトウェアの開発に関する知識から，繰り返し出現する問題の共通性と問題ごとに変わる個別性を，問題に対する解決の再利用可能な程度の抽象度と，同解決の適用範囲の適度な大きさのバランスを保ちながら捉えることで達成される．

パターンの発見対象は成果物としてのソフトウェア集合である．しかし，パターンは，成果物としてのソフト

ウェア集合から単純に類似する問題と解決を探し出せば良いというものではない．パターンの発見プロセスでは，抽出対象とするソフトウェアの開発経緯や開発に必要な知識を含む総合的な知識から，解決に至った理由などを探ることが大切である．

このような発見プロセスは経験に依存するところが大きく，直接的に支援する手法は研究成果としては得られていない．ソフトウェアパターン技術の周辺技術として，モデリング技術・プログラム解析技術などが考えられる．これらの周辺技術を対象に合わせて選択して用いることで，パターンの発見を幾らか支援できると考えられる．

また，発見プロセスでは，抽出対象とするソフトウェアに，既にどのようなパターンが適用されているのかを知ることも大切である．これまでに，デザインパターンを中心として，既存のソフトウェアから既知のパターンを検出する手法が提案されている．

- パターンの記述

パターンの記述プロセスは，発見したパターンを開発に関わる人々が解釈可能な形式に従って，パターンを用いる文脈と扱う問題・解決を記述することで達成される．このとき，なぜそのような解決を取るのかという理由と，解決の具体的な手順を明らかにする必要がある．

パターンの記述型としては， GoF Form[4] や Coplien Form, Alexandrian Form[21] といった様々なものが提案され用いられている．

記述プロセスの支援手法として，パターンの基準判定法を用いることができる．

- パターンの提出

記述したパターンをコミュニティに提出する．

- パターンの洗練

パターンの洗練プロセスは，コミュニティにおいて提出されたパターンを，その品質を高めるために洗練する作業によって達成される．洗練作業として，ライターズワークショップを行うことが一般的である．

パターンは単独で成り立つものではなく，他の依存・関連するパターンと共にパターンシステム[20] を形成する．関連するパターンが協調して働くことで，個々のパターンがもたらす解決集合よりも大きな解決をもたらすと指摘される[40]．

従って，洗練プロセスでは，提出されたパターンについて，表記された関連するパターンが妥当か・抜けがないかといった検証も行われる．このような検証作業を，複数のパターン間の関係解析手法を用いて支援できる．

また，記述プロセスと同様に，提案されたパターンの妥当性の評価にパターンの基準判定法を利用できる．

(2) パターンの利用活動

利用活動とは，コミュニティにおける同意を得て形成されたパターン集合またはパターンシステムから，利用者が直面する問題と文脈に適合するパターンを選択し，利用者の重視する制約に合わせてパターンの修正と拡張を行い，得られたパターンを適用し，適用した結果について評価を行う活動である．パターンを適用した結果として得られる状況（Resulting Context）が，新たにパターンを適用する問題を伴う場合には，再びパターンの選択と適用を行う．パターンの利用活動は一般に，次にあげる 4 つのプロセスを順に実行することで達成される: 選択・拡張・適用・評価.

- パターンの選択

パターンの選択プロセスは，既存のパターン集合から，利用者が直面する問題と文脈に適合するパターンを検索して選択することで達成される．

選択プロセスでは，パターンの表現支援手法やパターンの適合性検証手法，関係解析手法によってパターンの検索と選択作業を支援することができる．

- パターンの拡張

パターンの拡張プロセスは，利用者が選択したパターンを，自身が重視する制約に基づいて適宜修正・拡張することで達成される．さらに，拡張プロセスにおいて，複数の同一の種類・抽象度におけるパターンや，あるいは異なる種類・抽象度のパターンを組み合わせることがある．

これまでに，幾つかのパターン拡張手法が提案されている．

また，拡張プロセスにおいて，選択したパターンに対する利用者の理解が不十分であるとき，不用意にパターンを拡張してしまい，パターンが本来意図する仕組みを逸脱してしまう可能性がある．パターンの関係解析手法を用いることで，このような誤ったパターンの拡張を防ぐことができると考えられる．

- パターンの適用

パターンの適用プロセスは，選択して拡張することで得られたパターンを，目的とする適用対象ソフトウェアの開発に用いることで達成される．

これまでに，主にデザインパターンの適用を自動化する幾つかの手法が提案されている．

- パターンの評価

パターンの評価プロセスは，パターンを適用した結果として得られた状況（Resulting Context）についての定性的特性に関する評価と，適用結果として得られたパターンの効果についての定量的評価を行うことで達成される．

これまでに，小規模な実験結果に基づいて，デザインパターンの効果の定量的評価に関する報告がある．また，パターンの適合性検証手法・検出手法を用いて，パターンが実際に正しく対象へ適用されたかどうかを確認することができる．

3. パターン指向開発支援手法

前節で述べたパターン指向開発プロセスにおける個々のパターン活動を支援する手法を以下に示す．

3.1 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のかどうかを検出することにより，ソフトウェアの理解を助けて保守性を向上させる試みがある．

提案されている検出手法は，主にオブジェクト指向プログラムからのデザインパターンの検出を目的としたものがほとんどであり，検出対象プログラムの言語として Java を扱うもの[23),24),26)]，C++ を扱うもの[27),28)]，Smalltalk を扱うもの[25)] などがある．

それらの検出手法の相違点として，既知のデザインパターンを事前に記述しておく記述形式と，デザインパターンのどのような特徴に基づいて検出を行うかという仕組みが挙げられる．

デザインパターンの記述形式として，Java 言語を拡張した形式[23)]，UML に基づく抽象化されたクラスモデル[24),28)]，論理型言語[25),27)]，抽象構文木と動的なメソッド呼び出し集合[26)] などが用いられている．

検出時に扱うデザインパターンの特徴は，ほとんどが静的特徴のみ（構造）を扱うが，Heuzeroth らの手法では静的特徴と動的特徴（振舞い）の両面からパターンの検出を行っている．

Kambayashi らは，オブジェクト指向分析モデルや設計モデルが変化する様子を，識別子軸や時間軸などによって形成される座標系における状態関数として表し，状態関数上で安定状態にあるモデルをパターンとして捉えることによって，モデルが変化する様子から既知のデザインパターンを検出する手法を提案している[29)]．

また，高橋らは，プログラムからデザインパターンを検出する際に，事前にデザインパターンを組み合わせて適用した場合の構造的な特徴情報を規則として準備し，その規則と照らし合わせることで検出元プログラムにおける複数のデザインパターン間の関連を抽出することを試みている[38)]．

3.2 表現支援手法

ソフトウェアパターンそのものの記述とは異なるパターンの表現方法をソフトウェア開発者が解釈可能な形式で与えることで，既存のソフトウェアパターンの理解を支援する試みがある．

Ong らは，ソフトウェアパターン中のフォースを非機能的特性に関する要求として捉え，フォース間の対立や協調・導出といった関係を網羅したフォース階層図を作成することで，既存のソフトウェアパターンの理解を支援し，さらに，パターンランゲージの洗練を支援する手法を提案している[30)]．

Lauder らは，デザインパターンの構造と振舞いを UML を拡張した表記法によって表すことで，デザインパターンの自然言語表現における曖昧さを排除し，理解を促進する手法を提案している[31)]．

3.3 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，パターンの組み合せの妥当性を確認した後に，パターンを組み合わせて利用することを支援する試みがある．

青山は，デザインパターンなどのソフトウェアパターンを組み合わせて発展させていく手法を提案している[32)]．

Eden は，高階論理 LePUS を用いてデザインパターンの静的な構造（クラス・フィールド・メソッド間の関係）を記述することで，デザインパターン間の洗練や利用と言った関係を導出することを試みている[33)]．関係を導く例として，Vlissides がパターンの持つ意味に基づいて Multicast パターンから Observer パターンと共通する部分を取り除くと Typed Message パターンが得られると主張した[34)] のに対して，Eden らは LePUS を用いて，Multicast パターンは Typed Message パターンから成り立つが，Multicast パターンと Observer パターンには類似の関係があり，Vlissides が主張するような 3 つのパターン間の関係を確認できなかったと報告している[35)]．

Riehle は，GoF のデザインパターンについて，デザインパターンの個々の参加要素が持つ役割（ロール）同士の間の関係を捉えることで，デザインパターンの分類を行い，，さらに，複数のデザインパターンを適切に組み合わせて用いる手法を提案している[40)]．

Noble らは，パターンを，パターン名という記号表現と，パターン記述という記号内容を持った記号として捉えている．このとき，パターン記述もまた，解決という記号表現と，意図という記号内容を持った記号として捉えている．この考えに基づき，GoF のデザインパターンを記号として表し，デザインパターン間の関係を導出することを試みている[41)]．

3.4 適合性検証手法

ソフトウェアパターンの形式的な仕様記述を行い，あるパターンを適用した結果として得られたソフトウェア

が，適用したパターンの特性を全て正しく満しているかどうかを検証する試みがある．

Cechich らは，デザインパターンの構造を仕様記述言語を用いて形式的に記述し，デザインパターンを適用して得られたオブジェクト指向設計モデルが，適用したパターンの特性を全て満すような正しいパターンインスタンスであるかどうかを検証する手法を提案している[36]．

同様に，畑口らは，オブジェクト指向設計が，デザインパターンが意図する意味に沿ったものかどうかを，実行情報を用いて確認する手法を提案している[39]．

3.5 適用支援手法

これまでに数多くのパターンの適用支援に関する手法が提案されている．適用支援手法は，分析・設計モデルやプログラムに対して特別なツールを用いてパターンを適用するツール系の手法と，拡張されたプログラム言語またはプログラムに近い形でパターンの記述を行うプログラム言語系の手法の 2 種に大別できる．

- ツール系

デザインパターンを HTML 文書または SGML 文書として利用者に提示し，実装に依存する情報の入力を受けつけて，選択されたパターンからプログラムを自動生成する試みがある[42),43)]．

既にあるオブジェクト指向設計モデルに対して，専用の設計ツールを用いて登録済のデザインパターンを適用する試みがある[44),45)]．これらの手法では，ツールの利用者が新たなデザインパターンを用意することもできる．

既にあるオブジェクト指向プログラムに対して，専用のツールを用いて登録済みのデザインパターンを適用する試みがある[46)~48)]．これらの手法では一般に，パターンを適用する際に，パターンを構成する役割に対応してクラス名等を指定することができる．

Shizuki らは，ソフトウェアパターンを視覚的に定義し，視覚的な穴埋め操作によってパターンを具体化していくプログラム開発環境 KLIEG を提案している[49]．

また，デザインパターンの適用を支援する機能は，幾つかの商用・非商用の統合開発環境・モデリング環境に組み込まれている．例えば，モデリングツールである Pattern Weaver[50] や Rational Rose[51]，および，統合開発環境である Borland Together ControlCenter[52] などは，事前に登録済みの GoF のデザインパターンや新規に定義するパターンの適用機能を提供する．

オープンソースの統合開発環境である Eclipse では，Sametinger らによる Pattern Support for Eclipse[53] をプラグインとして用いることで，デザインパターンの適用機能を利用できる．

- プログラム言語系

GoF のデザインパターンは本来，オブジェクト指向設計モデルと同モデルのオブジェクト指向プログラム言語による実装を想定したものであるが，プログラム中において，デザインパターンに関する箇所は 1 つの箇所にまとまらず複数の箇所に散在することが多い．また，デザインパターンを適用する以前の元の処理内容と，デザインパターンの仕組みを実現する処理が混在する箇所も出現することとなる．

そこで，デザインパターンを，デザインパターンを適用する以前の元の処理内容とは直交する関心（アスペクト）と捉えて，既存のオブジェクト指向言語を拡張したアスペクト指向言語やマクロ機構，および，差分ベースモジュール言語によって，デザインパターンを元の処理内容とは独立して記述しておき，プログラムのコンパイル時にデザインパターンを適用する試みがある[54)~56)]．これらの手法は，上述のようなデザインパターンに関するプログラムが散在することを防ぎ，全体の拡張性と保守性を向上させることができる．

3.6 基準判定法

パターンを作成して洗練する際には，対象とするパターンがパターンとしての体をなしているかどうかを判断可能とする基準が求められる．

パターンが満すべき基準として良く知られるものに，パターンが文脈・フォースの体系・ソフトウェア構成の 3 つ組を成すという特性上の基準（A Timeless Way of Hacking[17]）と，パターンが 3 回以上利用されているという利用実績上の基準（Rule of Three[15]）がある．

Brown らは上述の基準に加えて，パターンがレビューを受けていることなどを含めてパターン基準として提案している[16]．

また，Coplien らは，ソフトウェアパターンはパターンを用いる対象のある特性について対象性を破壊する（Symmetry Breaking）という特徴をもつものであると捉えて，ある実装上の仕組みが，特定の文脈においてパターンであるかどうかを判定することを試みている[37]．

3.7 効果測定法

パターンを用いた結果について，何らかの特徴を定量的に捉えることで，パターンの有効性を評価する試みがある．

Prechelt らは，小規模な C++ プログラムの修正作業におけるデザインパターンの効果を，実験に基づいて定量的に評価している[57]．複数のソフトウェア開発者を被験者として，デザインパターンを使用する場合と，より素朴な解決策を用いる場合とで修正作業にかかる作業時間と品質を比較した結果，多くの作業内容について，

デザインパターンを使用する場合に作業時間が同等もしくは短く，品質も高かったと報告している．また，現実には想定外の新しい要求が出されることがしばしばあるため，素朴な解決策を取るよりも，柔軟性をもたらすデザインパターンを使用した方が良いと結論づけている．

増田らは，プログラムの拡張作業におけるデザインパターンの効果を，ソフトウェアメトリクスを用いて評価している[58]．同一のプログラムに対して，デザインパターンを用いて拡張作業を行った版と，用いずに拡張作業を行った版を比較した結果，クラス数やメソッド数には差違はなかったものの，デザインパターンを用いた版はコード行数が少なかったと報告している．また，CK メトリクス[59] を用いて，両版におけるクラスの複雑度を測定したところ，幾つかのメトリクスの測定結果について有意な差を確認している．

4. これからのパターン研究

前節でみたように，ソフトウェアパターンが誕生して以来，これまでに様々な研究がなされている．研究成果の内で，デザインパターンの適用手法については，既に商用・非商用なデザインパターン適用ツールとして実用化されるに至っている．

一方，それ以外のパターン・手法については，研究事例が少ないのが現状である．このようなソフトウェアパターン研究の現状を踏まえて，今後，ソフトウェアパターン研究に求められる事柄などを以下に示す．

4.1 デザインパターン以外への展開

これまでのソフトウェアパターンに関する研究のほとんどは，デザインパターンを題材としている．これは，ソフトウェアパターンの中で GoF のデザインパターンが最も一般に普及していることや，分析モデルなどと比較してデザインパターンが関係する設計モデルとプログラムはその特徴を捉えやすいことなどが原因として考えられる．

上記のような原因から，今後もデザインパターンが題材として取り上げられる傾向は続くものと予想される．

しかし，粒度の粗いソフトウェアパターンは，比較的粒度の細かいデザインパターンよりも大きな解決をもたらすため，適切に再利用できた場合の効果はデザインパターンのそれと比較して大きいと考えられる．

従って今後は，アーキテクチャパターンやアナリシスパターンといったデザインパターンよりも粒度の粗いソフトウェアパターンを題材とした研究が活発に行われることを期待したい．また，開発プロセスや組織に関するマネジメントパターンについても，伝統的な開発プロセス研究と関連して，活発に研究されることを期待したい．

4.2 コミュニティの捉え方

パターン指向開発においては，パターンの洗練と管理を行うパターンコミュニティが重要な役割を担う．しかしながら，これまでのソフトウェアパターン研究は，コミュニティの存在と役割を大きくは考慮していないように考えられる．

ソフトウェアパターンそのものの仕組みと解決にのみ目を奪われることなく，ソフトウェアパターンを取り巻く人々の有り様も視野にいれた研究が今後行われていくことを期待したい．

4.3 ソフトウェアパターンと研究成果

国内において，独自のソフトウェアパターンを研究成果として提案する事例が出てきている[60),61)]．今後，情報処理学会ソフトウェア工学研究会パターンワーキンググループ[62)] が提供・支援する場を通じて，このような独自のソフトウェアパターンの研究成果としての提案が活発に行われることを期待したい．

我々は以前に，パターンコミュニティにおいてパターンとして提案したものを[63)]，学会において報告したことがある[64)]．新しいソフトウェアパターンの提案が研究成果である場合に，コミュニティと学会等における発表のあり方の違いや，記述形式の違いなどは，議論されるべき事柄である．

5. おわりに

ソフトウェアパターンに関する研究動向について述べ，研究成果とパターン指向開発プロセスの関連付けを行った．さらに，ソフトウェアパターン研究の今後について述べた．今後は，パターンワーキンググループにおける活動などを通じて，ソフトウェアパターン研究が活性化されることを期待したい．

参考文献

1) Kent Beck and Ward Cunningham: Using a Pattern Language for Programming, In Addendum to the Proceedings of OOPSLA'87, ACM SIGPLAN Notices, 23(5)(1987).
2) Eric Gamma: Object-Oriented Software Development Based on ET++: Design Patterns, Class Library, Tools, PhD Thesis, University of Zurich (1991).
3) Eric Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns: Abstraction and Reuse of Object-Oriented Design, In Proc. of ECOOP'93, LNCS 707 (1993).
4) Eric Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns, El-

ements of Reusable Object-Oriented Software, Addision-Wesley (1994).
5) Jim Coplien: Advanced C++ Programming Styles and Idioms, Addison-Wesley (1992).
6) Peter Coad: Object Oriented Patterns, Communications of the ACM, 35(9)(1992).
7) The Hillside Group, http://hillside.net/
8) Masao Tomono, Tatsuo Kato and FUJINO Terunobu: On the way to establishing a jPLoP community, In Proc. of APSEC'99 (1999).
9) 児玉公信, 矢崎博英: JapanPLoP 活動報告, オブジェクト指向 2000 シンポジウム (2000).
10) History Of Patterns, http://c2.com/cgi-bin/wiki?HistoryOfPatterns
11) Dirk Rihele and Heinz Zullighoven: Understanding and Using Patterns in Software Development, Theory and Practice of Object Systems, 2(1)(1996).
12) 中谷多哉子, 青山幹雄, 佐藤啓太: bit 別冊 ソフトウェアパターン, 共立出版 (1999).
13) 鈴木純一, 田中祐, 長瀬嘉秀, 松田亮一: ソフトウェアパターン再考: パターン発祥から今後の展望まで, 日科技連 (2000).
14) 情報技術コンソーシアム, 次世代コンポーネントウェア技術に関する研究開発・コンポーネントとパターンに関する動向調査報告書 (2000)
15) Will Tracz: RMISE Workshop on Software Reuse Meeting Summary, In Software Reuse: Emerging Technology, IEEE CS (1988)
16) William Brown, Raphael Malveau, Hays McCormick and Thomas Mowbray: The Software Patterns Criteria: Proposed Definitions for Evaluating Software Pattern Quality (1998), http://www.antipatterns.com/whatisapattern/
17) Deepak Alur, John Crupi and Dan Malks: Core J2EE Patterns: Best Practices and Design Strategies, Pearson Education (2001).
18) 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 (2001).
19) James Coplien and Bobby Woolf: A Pattern Language for Writers' Workshop, In Pattern Language of Programming Design 4, Addison-Wesley (2000).
20) Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerland and Michael Stal: Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons (1996).
21) James Coplien: Software Design Patterns: Common Questions and Answers, In The Patterns Handbook: Techniques, Strategies, and Applications, Cambridge University Press (1998).
22) 吉田裕之, 森崎雅稔: パターン指向アプリケーション開発, FUJITSU, 50(3)(1999).
23) 金子崇之, 深澤 良彰: プログラムからのデザインパターン抽出技法とその評価, 電子情報通信学会, ソフトウェアサイエンス研究会報告, SS98-40 (1998).
24) Herve Albin-Amiot and Yann-Gael Gueheneuc: Meta-modeling Design Patterns: aplication to pattern detection and code synthesis, In Proc. of Workshop on Adaptative Object-Models and Metamodeling Techniques at ECOOP'01 (2001).
25) Roel Wuyts: Declarative Reasoning about the Structure of Object-Oriented Systems, In Proc. of TOOLS USA'98 (1998).
26) Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom and Welf Lowe: Automatic Design Pattern Detection, In Proc. of Workshop on Program Comprehension at ICSE'03 (2003).
27) Christian Kramer and Lutz Prechelt: Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software, In Proc. of WCRE'96 (1996).
28) Rudolf Keller, Reinhard Schauer, Sebastien Robitaille and Patrick Page: Pattern-Based Reverse-Engineering of Design Components, In Proc. of ICSE'99 (1999).
29) Yasushi Kambayashi and Mikio Ohki: Extracting the Software Elements and Design Patterns from the Software Field, In Proc. of ICEIS'03 (2003).
30) Han-Yuen Ong, Michael Weiss and Ivan Araujo: Rewriting a Pattern Language to Make it More Expressive, In Proc. of ChiliPLoP2003 (2003).
31) Anthony Lauder and Stuart Kent: Precise Visual Specification of Design Patterns, In Proc. of ECOOP'98, LNCS 1445 (1998).
32) 青山幹雄: ソフトウェアパターンの組合わせと進化のパターン, 情報処理学会ソフトウェア工学研究会報告, SE124 (1999).
33) Amnon Eden: A Visual Formalism For Object-Oriented Architecture, In Proc. of IDPT-2002 (2002).
34) John Vlissides: Multicast - Observer = Typed Message, C++ Report, Nov./Dec. (1997).
35) Amnon Eden, Yoram Hirshfeld and Amiram Yehudai: Multicast - Observer $\neq$ Typed Message, C++ Report, Oct. (1998).
36) Alejandra Cechich and Richard Moore: A Formal Basis for Object-Oriented Patterns, In Proc. of APSEC'99 (1999).
37) James Coplien and Liping Zhao: Symmetry and Symmetry Breaking in Software Patterns, In Proc. of GCSE'00, LNCS 2177 (2000).
38) 高橋克暢, 丸山勝久: 振る舞いのルール化によるデザインパターン間の関連抽出, 情報処理学会第 65 回

全国大会 (2003).
39) 畑口剛之, 池田健次郎, 岸知二: デザインパターンへの適合性確認手法について, 情報処理学会, ソフトウェア工学研究会報告, SE121 (1998).
40) Dirk Riehle: Composite Design Patterns, In Proc. of OOPSLA'97 (1997).
41) James Noblea and Robert Biddle: Patterns as Signs, In Proc. of ECOOP'02, LNCS 2374 (2002).
42) Frank Budinsky, Marilyn Finnie, Marilyn Finnie, John Vlissides and Patsy Yu: Automatic code generation from design patterns, IBM Systems Journal, 35(2)(1996).
43) Mika Ohtsuki and Norihiko Yoshida: A Source Code Generation Support System Using Design Pattern Documents Based on SGML, In Proc. of APSEC'99 (1999).
44) 永山英嗣, 原田実: デザインパターン適用における設計図融合と最適パターン探索の支援系 OOPAS, オブジェクト指向 2000 シンポジウム (2000).
45) Takashi Kobayashi, Masahiko Kamo, Takayuki Sanui and Motoshi Saeki: Object-Oriented Modeling of Software Patterns, In Proc. of ISPSE2000 (2000).
46) Johannes Sametinger: Design Composition, Journal of Computer Science and Technology, 3(1)(2003).
47) 上原忠弘, 山本里枝子, 吉田裕之, 森崎雅稔: パターン指向開発とパターン自動適用ツール, オブジェクト指向'99 シンポジウム (1999).
48) 山下純司, 谷川健, 高木俊幸, 林雄二: Java プログラミングに対するデザインパターン適用支援ツール, 第 9 回ソフトウェア工学の基礎ワークショップ (2002).
49) Buntarou Shizuki, Masashi Toyoda, Etsuya Shibayama and Shin Takahashi: Smart Browwing among Multiple Aspects of Data-Flow Visual Program Execution, Using Visual Patterns and Multi-Focus Fisheye Views, Journal of Visual Languages and Computing, 11(5)(2000).
50) Pattern Weaver, http://www.foundatao.com/
51) Rational Rose, http://www.rational.co.jp/products/rose/
52) Borland Together ControlCenter, http://www.borland.co.jp/together/
53) Pattern Support for Eclipse, http://www.swe.uni-linz.ac.at/people/sametinger/research/pse.html
54) Michiaki Tatsubori and Shigeru Chiba: In Proc. of Workshop on Reflective Programming in C++ and Java at OOPSLA'98 (1998).
55) Jan Hannemann and Gregor Kiczales: Design Pattern Implementation in Java and AspectJ, In Proc. of OOPSLA'02 (2002).
56) 田中哲, 一杉裕志: MixJuice 言語によるデザインパターンの改善, 情報処理学会論文誌, 44(SIG4)(2003).
57) Lutz Prechelt, Barbara Unger, Walter Tichy, Peter Brossler and Lawrence Votta: A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions, IEEE Trans. on SE, 27(12)(2001).
58) 増田剛, 坂本憲広, 牛島和夫: 発展型ソフトウェア構築のためのデザインパターンの活用とその評価, コンピュータソフトウェア, 18(0)(2000).
59) Shyam Chidamber and Chris Kemerer: A Metrics Suite for Object Oriented Design, IEEE Trans. on SE, 20(6)(1994).
60) 鹿糠秀行, 三部良太, 矢島敬士: ソフトウェアパターンの適用における逸脱パターンの提案, 情報処理学会ソフトウェア工学研究会報告, SE137 (2002).
61) 入沢賢一, 青山幹雄: 分散オブジェクト環境のパターン指向性能設計方法論, オブジェクト指向 2001 シンポジウム (2001).
62) 情報処理学会ソフトウェア工学研究会パターンワーキンググループ, http://patterns-wg.fuka.info.waseda.ac.jp/
63) 鷲崎弘宜, 深澤良彰: パターン: 引用からの品質, JapanPLoP 第 7 回パターン研究会 (2001).
64) 鷲崎弘宜, 深澤良彰: 片方向引用情報に基づく論文の品質評価, 電子情報通信学会 2001 年総合大会 (2001).