特集●サーベイ論文

デザインパターンへのソフトウェア工学的取り組み

鷲崎 弘宜　坂本 一憲　大杉 直樹　権藤 克彦　服部 哲
久保 淳人　小林 隆志　大月 美佳　丸山 勝久　榊原 彰

自由度の高いソフトウェアの設計において，偶然ではなく繰り返される問題や解決をデザインパターンとして識別し，再利用することが効率性と一貫性の向上に欠かせない．本論文では，理解や再利用，対話を目的としたオブジェクト指向開発におけるデザインパターンの有効活用を目的として，デザインパターンへのソフトウェア工学上の取り組みの発展経緯を解説する．続いて，デザインパターンに関する適用や検出，検証といった主要な研究成果を取り上げる．

Due to high flexibility in software design, it is important to extract and reuse repeated problems and solutions as design patterns in order to improve design efficiency and consistency. In this paper, we report the result of the survey on engineering researches on design patterns in object-oriented software development. Targeted researches include the design pattern application, detection and verification.

1　はじめに

デザインパターンとは，ソフトウェアの設計における特定の文脈上で頻出する問題，その解法や考慮すべき事柄をまとめた記述である†0．扱う要求や制約の類似性に伴って繰り返し共通に出現する設計をデザインパターンとして再利用することで，新たな設計を効率よく一貫性のある形で進めることができる．Gamma らによりオブジェクト指向設計における典型的な 23 のデザインパターン [27]（Gang-of-Four: GoF パターンと呼ばれる）が提案されて以来，オブジェクト指向設計に限らず非オブジェクト指向設計や並行性設計など，各種設計における様々なデザインパターンが提案され今日に至っている．その中で本稿では，GoF パターンを中心としてオブジェクト指向設計のためのデザインパターンに関するソフトウェア工学的取り組みを扱う．

デザインパターンへのソフトウェア工学的取り組みは，パターン集の電子化と閲覧（パターンブラウザ）に端を発する．続いて，デザインパターンの普及に伴い，プログラミング技術の応用による単一デザインパターンのプログラムソースコードや設計モデルへの適用手法の提案が盛んとなった．しかしながら，デザインパターンを扱うことの難しさは，デザインパターンをコードやモデルに展開することよりも，対象モデルやコードの状況ならびに問題を把握しデザインパターンを適切に選択することにあることが認識されるようになった [12]．そのような認識に伴い，検出や検証といった適用以外の活用支援が研究されるようになった．プログラム解析技術の発展の影響もあ

Software Engineering Approaches for Design Patterns.

Hironori Washizaki, 早稲田大学, 国立情報学研究所 GRACE センター, Waseda University, National Institute of Informatics. Kazunori Sakamoto, 早稲田大学, Waseda University. Naoki Ohsugi, NTT データ, NTT DATA CORPORATION. Katsuhiko Gondow, 東京工業大学, Tokyo Institute of Technology. Satoshi Hattori, 東京工業大学, Tokyo Institute of Technology. Kubo Atsuto, 青山メディア研究所, Aoyama Media Laboratory. Takashi Kobayashi, 名古屋大学, Nagoya University. Mika Ohtsuki, 佐賀大学, Saga University. Katsuhisa Maruyama, 立命館大学, Ritsumeikan University. Akira Sakakibara, 日本アイ・ビー・エム, IBM Japan.

コンピュータソフトウェア, Vol.27, No.YY (2011), pp.ZZ–ZZ.

[解説論文] 2011 年 4 月 15 日受付.

†0　本論文の一部は，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†1の有志によって実施されたパターン研究調査活動 [72] [73] に基づく．

り，特に既知のデザインパターンの検出は世界的に活発な研究題材である．また実務上の観点から，デザインパターンの有効性を測定等により実証することも取り組まれつつある．

本稿では最初に，ソフトウェア工学におけるデザインパターンへのそれらの取り組みの発展の経緯を解説する．続いて，デザインパターンに関する主要な研究成果を取り上げる．

2 デザインパターンへの取り組みの経緯

2.1 ソフトウェアパターンの発展経緯とコミュニティ

ソフトウェア開発におけるパターンの歴史は，Christopher Alexander が提唱する建築におけるパターンの原理，ならびにそれをまとめあげたパターンランゲージの考え方[2] に触発されて，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ランゲージの提案[5] 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[25]，同じく Gamma および Helm, Johnson, Vlissides（GoF: Gang of Four）らによる GoF パターンのまとめ[26][27]，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[15]，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[14] などを経て，ソフトウェアプロダクトに関するデザインパターンを中心とした拡充と適用が進められてきた．さらにパターンの原理に限らず，Alexander の提唱する様々な原理（有機的秩序，参加，漸進的成長，診断，調整）が，プロセスや組織構成のパターン，ならびに，その後の eXtreme Programming（XP）に代表されるアジャイル開発の基礎を与えている[23]．アジャイル開発とは，軽量かつ反復漸進や顧客参加により変化に対応しやすく高い顧客満足度を得るソフトウェア開発プロセスや手法の総称である．

これらのソフトウェアパターンの普及と発展は，The Hillside Group[†2]や JapanPLoP[64][38] などのパターンコミュニティの形成と発展によるところが大きい．ソフトウェアパターンやパターンコミュニティの発展の経緯については文献[61][75][76] が詳しい[†3]．

2.2 デザインパターン再考

デザインパターン（以降，“ パターン ”と略記）とは，ソフトウェアの設計における典型的な問題と，問題を生じる文脈，および解法としての設計上の構造や振る舞いをまとめあげた記述である．代表的なパターン集（カタログ）として，オブジェクト指向に基づくという制約以外には実装方法や処理内容に依存しない汎用型として以下のものがある．

- GoF パターン[27]
- 主にアーキテクチャパターンを扱う文献[10][3][24] において関連する形で収録されたデザインパターン

実装方法や処理内容に特化したものとして例えば以下のものがある．

- データベース処理のパターン[24]
- Doug Lee らの並行処理のパターン[43]
- セキュリティ処理のパターン[80]

例として，GoF パターンの一種である Observer パターン[27] の記述を図 1 に示す．図 1 において Observer パターンは，名前（Name），文脈（Context，文献[27] では適用可能性: Applicability），問題（Problem，目的・意図/動機: Intent/Motivation），解決（Solution, 構造/協調関係: Structure/Collaborations），関連パターン（Related Patterns）の 5 つの項目として定義されている．解決においては設計上の構造と振る舞いが，モデル，コード，文章により表現される．これらの項目の種類や意味は，パターンの種類や記述する作者によってまちまちであるため，もともとのパターン文書は半構造文書ということができる．

GoF パターンは，具体的なオブジェクト指向クラス集合を再利用するのではなく，特定の役割（ロー

†2 http://hillside.net/

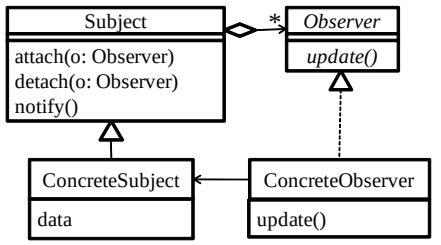
†3 他には下記など.
http://c2.com/cgi-bin/wiki?HistoryOfPatterns

ル）を持った複数のクラスの組み合わせの方法の再利用を与える．例えば図 1 においては，設計上の解決として Subject, ConcreteSubject, Observer, ConcreteObserver の 4 つのロールからなる構造が示されている．

このようなパターンの主要な利用目的 [27] を以下に示す．

- 理解: 過去のソフトウェア設計の解読を促進できる．適用されているパターンが分かれば，元の設計意図を容易に推測し，その意図に沿って活用できる．
- 再利用: 熟練者の思考過程と設計を一貫して再利用できる．同じ設計問題を同じように解決するため，一貫性のあるアーキテクチャを高効率に得ることに繋がる．
- 対話: 設計上の意思疎通を促進できる．つまり，パターン名を共通の語彙として集団で共有しておけば，いちいち内容を伝えるよりも，パターン名を一言伝えた方が効率が良い．

Name: Observer
Applicability (Context): ... However, objects cannot simply work in isolation. ... When objects collaborate, the objects may have to notify each other when an object changes state. …
Intent / Motivation (Problem): How can an object notify other objects of state changes without being dependent on their classes?
Structure / Collaborations (Solution): Maintain a list of interested dependents (observers) in a separate object (the subject). …



Related Patterns: …

図 1 Observer パターンの記述（括弧内は他の一般的なパターン記述形式 Canonical Form[†5] に照らした場合の対応項目）

2.3 デザインパターンのモデル化と研究経緯

パターンが認知されオブジェクト指向設計において再利用されるに従い，ソフトウェア開発においてパターンを上述の目的（理解，再利用，対話）に照らして有効な活用を促進するための技術の開発が数多く行われてきた．ここで，各技術によるパターン活用の内容は，パターンの表現形式やモデル化の対象側面に依存するところが大きい．モデル化・表現の発展経緯と，その発展に付随する主な研究成果を表 1 に示す．それぞれの詳細を以下に述べる．

(1) 半構造文書としてのパターン

元来，パターンの形式は，Alexander による建築学におけるパターンランゲージの記述形式にならって，目的や状況，問題，および解決といった項目と，その項目に対応する事柄の自然言語記述の集合として定義されてきた．パターンを単なる半構造文書として扱うならば，計算機による活用支援は，キーワード検索や単純な関連付けに基づく集積といった，一般的な文書処理技術に基づいた技術に限定される．

以上の理由から，パターンへのソフトウェア工学的研究成果は最初に，大小様々なパターンカタログの電子化として現れた．1995 年に Web 上でのソフトウェアパターンリポジトリ Portland Pattern Repository（PPR）[†6] が登場し，PPR は世界最大のパターンリポジトリとして継続的に発展してきた．PPR は閲覧と同時に多人数による迅速な編集を可能とし，その仕組みは Wiki として今日ではパターンに限らず Wikipedia といった様々な応用がなされている [23]．

(2) 解決としての構造への着目

カタログの電子化に続いて，プログラミング技術を応用することにより，主にパターンの直接的な適用作業の自動化について行われてきた．具体的には，1990 年代後半から 2000 年代初頭にかけて，GoF パターンが解決として与える構造にのみ着目して，同パターンの特定のオブジェクト指向プログラミング言

†6 http://c2.com/ppr/

表 1　パターンのモデル化・表現と研究発展の経緯

時期	目的	表現形式	モデル化の側面	主要な研究技術
1995–	対話	自然言語，半構造文書	全体	閲覧や編集
–1999	再利用	プログラム，図	解決（構造のみ）	適用
2000–	理解，再利用，対話	形式仕様，グラフ構造など	解決（構造および振る舞い），問題など	適用，検出，検証など

語環境に基づくコード生成[9]や，設計モデルへのパターンの自動的適用[37]について，活発に取り組まれてきた．具体的には，個々のパターンが解決として与える構造を，関連付けられたロールの集合（図 1 における構造）として定義しておき，各ロールが持つ属性やメソッド，さらにはロール間の関係を，適用先のモデルやコード中の構成要素（クラスやインタフェース）に対応付ける形で展開する．

この成果は，今日において幾つかのモデリングツールやプログラミング環境において，パターン適用機能として実用に至っている．

(3) 仕様が記述されたデザインパターン

固定されたロール集合としてのパターンのモデル化は，パターンが与える最終的な固定された解法の再利用のみを実現し，パターン本来の目的である“ 設計に至った理由も含めた意思疎通 ”[6]を実現しなかった．設計活動は，対象とする元の設計状況の理解や，変更時の構成要素間の調整といった，開発者による判断を必要とする高度で複雑・知的なものである．その思考プロセスを計算機によって支援するには，パターンが与える設計上の構造や振る舞いの意味をより厳密に定義することや，パターンを適用した後にその事実を後から振り返って確認可能とすることなどが求められる．

そこで 1990 年代後半から 2000 年代に入ると，形式仕様記述やメタプログラミング技術を活用することで，個々のパターンの仕様を厳密に定義する技術[60][45][22]や，振る舞いやセマンティクスを含む詳細な定義によりパターンを既存のモデルやコードから検出する技術[11][1][35][78][32][8]，および，もとの設計モデルやコードとパターンに関わるモデルやコードを別個に管理・統合する技術[63][29][62][18]などが活発に提案されてきた．

さらに 2005 年頃以降においては，プログラム解析技術やリポジトリマイニング技術の研究発展と，プログラムソースコードを含む種々の開発データの蓄積利用に伴い，大量の開発資産データを活用あるいは対象としたパターンの検出[20][53]や検出手法の検証[52]の研究について取り組まれつつある．リポジトリからの検出に関する最近の成果は文献[20]が詳しい．

3　パターン指向開発プロセス

パターンの抽出から適用へ至る一連のプロセスをパターンライフサイクルプロセス[47]と呼ぶ．パターンのライフサイクルプロセスは，パターンの抽出活動とパターンの利用活動の 2 つから構成される．両活動を図 2 に示す．

パターンに関する支援技術や研究成果は概ね，パターンや関連ソフトウェアのライフサイクルにおける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，抽出や利用の活動と，各活動を支援可能な技術・研究成果を合わせて示す．

3.1　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，ワークショップ等を経た洗練を受ける活動である．

- パターンの発見
  パターンの発見作業は，パターンの抽出対象とするソフトウェアや開発知識から，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．

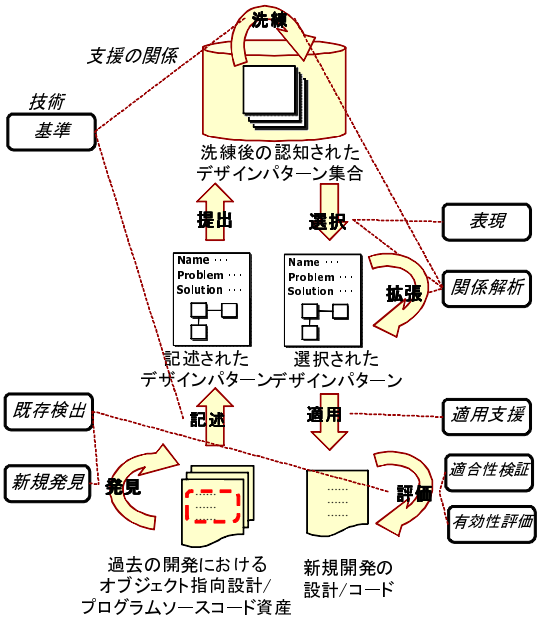


図 2 パターンライフサイクルプロセスと支援技術の関係

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．

このような発見作業は経験に依存するところが大きいが，直接的に支援する新規発見技術（[65] [28] など）が提案されつつある．また，パターンの周辺技術として，モデリング技術や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．

さらに発見作業においては，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，既存のソフトウェアから既知のパターンを検出する技術（[1] など）が提案されている．

- パターンの記述

  パターンの記述作業は，開発に関わる人々が解釈可能な形式に従って，発見したパターンを用いる文脈や扱う問題・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．

  パターンの記述形式としては，GoF Form [27] や Coplien Form, Alexandrian Form [16] といった様々なものが提案され，用いられている．

  記述作業の支援技術として，様々なパターンの基準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やレビュー組織に提出し，蓄積する．

- パターンの洗練

  パターンの洗練作業とは，コミュニティにおいて提出されたパターンを，ライターズワークショップ [61] 等を通じてその品質を高めるために洗練する作業である．

  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（複数のパターンが結びついて成り立つ全体・体系）[10] 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ことが期待される [56]．

  従って，洗練作業では，提出されたパターンについて，記述された関連パターンの妥当性や抜けなどの検証も行われる．このような検証作業は，複数のパターン間の関係解析技術を用いて支援できる．

  また，記述活動と同様に，提案されたパターンの妥当性評価にパターンの基準を利用できる．

3.2 パターンの利用活動

利用活動とは，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，利用者が重視する制約に合わせてパターンを修正あるいは拡張し，適用し，適用結果について評価する活動である．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．

- パターンの選択
  パターンの選択作業は，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．
  選択作業では，パターンの表現・モデル化技術やパターンの適合性検証技術，関係解析技術によってパターンの検索と選択作業を支援できる．
- パターンの拡張
  パターンの拡張作業は，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，拡張作業において，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．
  ここで，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あるいは他と組み合わせてしまい，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．そこで，本来意図する仕組みを逸脱することなく妥当にパターンを組み合わせ・拡張することを支援するために，幾つかのパターン組み合わせ・拡張手法（[56] [50]）が提案されている．既存パターン間の未知の関係を自動解析・検出する技術 [41] も，パターンの適切な組み合わせや拡張の支援に用いることができる．
- パターンの適用
  パターンの適用作業は，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，パターンの適用を自動化あるいは支援する様々な手法が提案されている（例えば [37] [58] [29]）．
- パターンの評価
  パターンの評価作業は，パターンを適用した結果として得られた状況（Resulting Context）を観察・分析し，パターンの有効性や影響について定性的あるいは定量的に評価することで達成される．
  これまでに，小規模な実験結果に基づいて，パターンの有効性の定量的評価に関する報告（例えば [54] [44]）がある．また，パターンの適合性検証技術 [11] [39] や検出技術（例えば [1] [8]）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．

4 デザインパターンの活用支援技術

前節で述べたライフサイクルプロセスと技術整理に基づき，主要な研究成果を以下に示す．

4.1 新規デザインパターンの発見

ソフトウェア工学におけるリバースエンジニアリングとは，プログラムの理解・保守や特定部分の再利用を目的として，既存のプログラムを解析する技術である．パターン研究の分野では，リバースエンジニアリングを活用して，既存のプログラムから共通に出現するプログラムコードを新規のパターンとして抽出することが試みられてきている．

Tonella らは，一般的な関係（継承や関連，メソッド呼び出しなど）を共有するクラス群を識別するために，コンセプト分析（Concept Analysis）手法を用いて既存のソースコードからオブジェクト指向設計パターンを発見する手法を提案している [65]．プログラムコードに対して，その内部に存在するクラスとクラス間の関係を，提案するアルゴリズムに基づき複数のコンセプトという単位に分類する．分類したコンセプトにおいて同値性が見られるものをまとめることで，最終的なパターンを発見している．論文では，21000 行程度の C++アプリケーションに対して，提案手法を用いてパターン検出の評価実験を行っている．実験においては，例えばクラス数 3 個のパターンとして，クラスの派生（継承）を用いた Adapter パターン [27] と同様の構造が 34 箇所に存在することを自動的に特定し，そのうち 27 箇所は Adapter パターンにおけるインタフェースを適合させる目的を実現したものであることを人手で確認している．発見されたパターン（コード中の共通の特徴と呼ぶべきもの）に対する意味付けは利用者に委ねられるものの，多様なプログラムコードに自動的に適用できる点が興味深い．

さらに Arevalo らは，Tonella らのクラス間関係に基づくコンセプト分析手法[65] について，ラティス（完全半順序）を考えることで，クラス発見手法の効率化とパターン間の協調関係の定義に成功している[4]．コンセプトの構築には，サブクラスかどうか，抽象クラスかどうか，利用関係にあるかどうかの 3 つの特性を用いる．各コンセプトにおけるクラスをノード，クラス間の関係をアークとする Intent Relation Graph に対して，すべてのノードが結合されているコンセプトだけを残す．さらに，等価なコンセプトを 1 つにまとめることにより，従来手法[65] よりも探索空間を削減している．さらに，コンセプトに現れるクラス群およびそれらの特性の包含関係で定義した半順序関係に基づいた幾つかの協調関係を導入することで，類似パターンの発見を実現している．パターンの発見のみならず，パターン間の関連までを含めた抽出を行っている点が興味深い．

Niere らは，既存のプログラムコードからパターンを発見する際の問題を，実装におけるバリエーションと捉えて，ASG（Abstract Syntax Graph）の導入によりその問題を解消し，パターン発見の精度を向上させる手法を提案している[49]．ASG とは，ソースコードの構成要素を頂点，それらの間に成立する関係（属性としての所有やメソッドとしての所有など）を辺で表現した有向グラフである．ASG に対して特定のグラフとのマッチングを行うことで，パターン（サブグラフ）を見つける．パターンが見つかった場合に，関連するノードに対して，パターン名を記した注釈を付加する．パターンの発見作業者は，この注釈を見ながらさらなるグラフマッチングの適用を考え，最終的にアプリケーション内部のパターンを識別していく．パターンの抽出に ASG を用いる点と，利用者とのインタラクションを前提としている点が興味深い．

メトリクス技術を，新たなデザインパターンの発見に活用する試みもある．デザインパターンを特定する作業は，複雑なグラフにおいてパターンを見つけることに等しく，可能なクラスの組み合わせとプログラムのサイズという点から一般的に困難である．そこで[28] では，ソフトウェア設計の基調（モチーフ）において，特定の役割を担うクラスの外部的特性（メソッドやフィールドの数，親子関係，凝縮度，結合度）に基づく数値（フィンガープリント）を導入している．いくつかのアプリケーションに対して，この数値を機械的に学習させることで，デザイン基調において特定の役割を担うクラスおよびクラス群が持つべき数値を割り出している．得られた数値を用いて，特定のデザイン基調において明らかに役割を担っていないクラスを探索空間から取り除くことで，パターンの発見工程において，より重要な役割を担うクラスを特定している．さらに，この数値はデザイン基調を用いて実現されたプログラムの品質特性を定量的に評価する際の指標となることが期待できる．メトリクスに基づく数値を，探索空間の削減としてパターン発見の初期段階に適用することで実用性を狙っている点が興味深い．

4.2 既知デザインパターンの検出

既存のソフトウェアについて，既知のパターンを検出することで，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[1][8][32][19][31]，C++を扱うもの[40][35]，Smalltalk を扱うもの[78] などがある．

それらの検出手法の相違点として，既知のパターンを事前に記述しておく記述形式と，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．

パターンの記述形式として，UML に基づく抽象化されたクラスモデル[1][35]，論理型言語[78][40][31]，抽象構文木と動的なメソッド呼び出し集合[32] などが用いられている．

検出時に扱うパターンの特徴は，ほとんどが静的特徴のみ（構造）を扱うが，Heuzeroth らの手法や Hayashi 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[32][31]．また，Blewitt らの手法や Dong らの手法では，静的特徴と動的特徴に加えて，プログラムのセマンティクスについても考慮して検出を行なっている[8][19]．これらの手法は，動的特徴やセマンティクスの情報を利用

することで，パターン検出における精度の向上を図っている．例えば Hayashi らは，Pree らによって提案されたメタパターンと呼ばれるパターン間に存在する共通の構造 [55] と実行時のトレース情報の両方，つまり，静的特徴と動的特徴の両方を用いて，パターンの検出条件を階層的に表現している．この検出条件を Prolog における述語で記述して検出処理を行うことで，検出手法の計算量の低減を実現している [31].

Dong らは，検出する際に着目する特徴，検出に用いるマッチング手法，検出対象のプログラムの表現方法，パターンの記述形式，検出結果の表示方法，人手を介するかどうか，検出可能なパターン，検出実験の対象ソフトウェアなど，様々な観点から検出手法を整理して，パターンの検出手法が抱える未解決の問題点についてよくまとめている [20].

検出上の問題として例えば，構造が類似するパターン群を判別することの難しさがあり，静的特徴に加えて上述のように動的特徴を併用することが検出精度向上の一つの方策として知られている．また鷲崎らは，プログラムソースコードの様々な静的特徴をメトリクスにより定量的に捉えたうえで機械学習を通じて検出する手法 [68] や，パターン適用前のソースコードの静的特徴を解析して検出する手法 [74] を提案し，精度向上を確認している．

水野は，複数のパターンの検出手法について整理した上で，それらの検出手法を和集合，多数決，適正値付き和集合の三種類の方法で組み合わせることで，検出能力の改善を試みている [46].

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パターンを検出する手法を提案している [34].

Posnett らは，メタパターンについて，構造抽出と記号実行による静的解析により，検出対象のプログラムサイズに比例するスケーラブルな検出手法を提案している [53].

また，Wegrzynowicz らは，あらかじめ用意したパターンの標準的なコードに対して，パターンの性質を維持した汎用的な変換，パターンの性質を維持したパターンに特化した変換，さらに，パターンの性質を壊すパターンに特化した変換の三種類のコード変換を適用することで，検出手法同士を比較可能なパターンの検出実験の実現のために，一貫性のあるテストスイートを生成する手法を提案している [52].

4.3 デザインパターンの基準

パターンを発見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．

デザインパターンに限らずパターン一般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（関連するフォースの集合からなる全体）・ソフトウェア構成の 3 つ組を成すという特性上の基準 [3] 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 [67]）がある．ここで“ フォース ”とは，当該パターンが問題の解決にあたり重視している事柄や品質その他への良い・悪い影響を指し，パターンによっては独立した項目として明示されている [76].

しかしながら，上述の条件が真に妥当なものであるとの保証はない．実際，条件の幾つかを満たさないパターンは多数存在する，そこで Winn らは，パターンの多様さに起因してパターンの必要条件を見出すことは難しいことを指摘して，必要条件の代わりに，パターンが備えるべき以下の 9 つの本質的特性を提案している [77]．ただし，これらの特性を備えていることをもって，対象とする知識文書が絶対的にパターンであるとは主張していない．論文では，特性一覧を用いて，既存のパターンの幾つかをパターンあるいはパターンではないものと区別することを試みている．“ パターンらしさとは何か ”という根源的な課題への取り組みとして興味深い．

- ソフトウェア／ドメインの設計を暗示する
- 複数の抽象レベルにおける表現を結びつける
- 機能的特性と非機能的特性の両者を扱う
- 得られるソフトウェア設計中に痕跡が残る
- ホットスポット（設計上の変動箇所）を捉える
- パターンランゲージの一部を形成する

- 繰り返し利用されることで有効性が確認済みである
- 特定のドメインに根ざしている
- 設計上の巨視的な方針を捉える

また，Coplien らは，Alexander の建築におけるパターンと対称性の概念[2] を参考として，デザインパターンに限らずパターン一般は対称性を破壊する（Symmetry Breaking）という特徴をもつと捉えて，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[17]．例えば Bridge パターン[27] の適用により，あるクラスとその派生クラスの間にリスコフの置換原則[57] が成立しなくなる場合，派生（継承）という“ 変換 ”に関してクラスの性質が維持されず対称性が破壊されたとみなす考え方を示している．

4.4 デザインパターンの表現と理解

パターンそのものの記述とは異なる表現やモデルを，開発者が解釈可能な形式で与えることで，既存のパターンの理解や選択を支援する試みがある．

Lauder らは，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，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[42]．

Soundarajan らは，非形式的に記述されていたパターンに対して，設計者がそのパターンを適用する際に満たさなければならない要求を正確かつ曖昧のないように理解するための仕様の形式化手法を提案している[60]．提案する仕様は，契約による設計（DbC: Design by Contract）に類似した記法により形式化されており，責任を担う部分と報酬を担う部分で構成されている．責任とは，クラスの構造，そのクラスの特定のメソッドの振る舞い，クラス間のやり取りに関して設計者が守らなければならない条件を指す．報酬とは，パターンを適用した際に受け取る恩恵（具体的には，責任に含まれるすべての要求が満たされる場合に外部に公開されることが保証されている特別な振る舞い）を指す．論文では，Observer パターンを例題とし，パターン全体の仕様，パターンに参加している個々の役割に関する形式仕様（図 3）を示している．図 3 では，Observer パターンを構成するロールとして 1 つの Subject 役と任意数の Observer 役をそれぞれ定義し，それらのロール間に成立する関係や制約を述語や論理式を用いて宣言している．例えば，`auxiliaryConcepts` 句において以下の関係や制約を宣言している．

- Subject 役の状態 $\alpha\sigma$ と，Subject 役に依存する Observer 役の状態 $\alpha\sigma$ が一貫している
- Subject 役の状態 $\alpha\sigma_2$ は，状態 $\alpha\sigma_1$ から変化したものである
- Subject 役の状態 $su\alpha\sigma_1$ と $su\alpha\sigma_2$ が同一で，かつ，$su\alpha\sigma_1$ と $ob\alpha\sigma$ が一貫している場合は，$ob\alpha\sigma$ は $su\alpha\sigma_2$ についても一貫している

パラメータ化された上述の付属概念を用いることで，パターンの仕様は，それぞれのインスタンスに対して拡張や変更の余地を残し，柔軟性を損なわずに正確性を追求できる．

```
pattern Observer {
 role: Subject, Observer*;
 state:
  Subject: set[Observer] _observers;
  Observer: Subject _subject;
  pattern: null;
 auxiliaryConcepts:
  relation:Consistent(Subject.ασ,Obsever.ασ);
  relation:Modified(Subject.ασ1,Subject.ασ2)
  constraint:
   [¬Modified(su ασ1,suασ2 )
    ∧Consistent(su ασ1,obασ)]
   ⇒Consistent(su ασ2,obασ)
```

図 3 Soundarajan らによる Observer パターンの形式仕様記述の一部[60]

Mehlitz らは，コンポーネントのプログラムコードについて，その利用のための制約条件や保証される特性に関する形式仕様記述を付加したモジュールを対象ドメインに特化したパターン（D4V パターンと呼ぶ）として記述し，再利用する手法を提案している[45]．D4V パターンは，保証される特性を用いて索引付けされる．解決されるべき各システム要求について，各 D4V パターンが保証する特性との合致を調査することで，適切な D4V パターンを選択する．論文では例として，イベント多重化コンポーネントに対して，優

先順位付きイベント処理・非ブロック・非同期が要求され，特性を満たす D4V パターンが選択される様子が示されている．

また Hsueh らは，パターンについて品質とその影響の観点から分析したモデルを用意し，ゴール指向分析法の適用によって得られたサブゴールのうちで，非機能要求の達成やトレードオフの関係にあるサブゴールの解消のための設計に役立つパターンを選択する手法を提案している [33].

4.5 デザインパターンの関係と拡張

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．

Eden は，高階論理 LePUS を用いて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，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 [22]．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 [70] ことを取り上げている．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 [21].

Riehle は，GoF パターンについて，パターンの個々の参加要素が持つロール間の関係を捉えることで，パターンを分類し，さらに，複数の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 [56].

Noble らは，パターンを，パターンの名前という記号表現と，パターンの記述という記号内容を持った記号として捉えている．さらに，パターンの記述もまた，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（つまり記号の中に記号が含まれる）．この考えに基づき，GoF パターンを記号として表し，パターン間の関係を導出することを試みている [50].

Kubo らは，既知のパターンを問題・解決の対としてモデル化し，異なるパターンの問題間や解決間，問題・解決間に対して文書間類似度を求めることで，大量のパターン集合における未知の関係を自動的に解析および検出する手法を提案し，組込みソフトウェア開発におけるパターン間の関係検出に成功している [41].

4.6 デザインパターンの適用

これまでに適用支援に関する数多くの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ミング言語またはプログラムに近い形でパターンを記述するプログラミング言語系，さらには一度のコード/モデル変換を通じた適用ではなく複数回の（外部への挙動を変えない）リファクタリングを通じた適用系の 3 種に大別できる．

(1) ツール系

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 [9] [51].

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パターンを適用する試みがある [48] [37]．これらの手法では，ツールの利用者が新たな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パターンを適用する試みがある [58] [69] [79]．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ロールに対応してクラス名等を指定することができる．

Shizuki らは，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 [59].

また，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†7や Rational Rose†8，および，統合開

†7 http://www.foundatao.com/

†8 http://www.ibm.com/developerworks/rational/products/rose/

発環境である Borland Together ControlCenter[†9]などは，事前に登録済みの GoF 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[†10]をプラグインとして用いることで，パターンの適用機能を利用できる．

(2) プログラミング言語系

GoF 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ミング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，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，パターンを適用する以前の元の処理内容と，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．

そこでパターンを，パターンを適用する以前の元の処理内容やモジュール群に対して直交あるいは横断的に関わる関心事と捉えて，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，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パターンを適用する試みがある [63] [29] [62]．これらの手法は，上述のような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．

例えば [18] では，オブジェクトの内容や構造を実行時に変更可能な動的オブジェクトシステム CLOS を用いて，GoF パターンを明示的に実装する手法を提案している．従来の実装手法では，クラス間の関連はコード自身で表現されており，パターンの表現が明示的でないという欠点を持つ．この論文では，CLOS のメタプログラミングの仕組みを活用して，各パターンについて実装コードの生成規則をメタクラスとして事前に定義し，そのメタクラスの再利用によって，パターンの実装を明示的に用意して再利用することに成功している．

†9 http://www.borland.com/us/products/together/

†10 http://www.swe.uni-linz.ac.at/people/sametinger/research/pse.html

Hannemann らは，GoF パターンの多くについて，構成するロールやロールを担当する実装クラスが対象プログラム内で横断的に出現する性質に着目して，アスペクト指向プログラミング言語 AspectJ を用いてパターンを実装する手法を提案し，AspectJ を用いた実装によってモジュール性（局所性，再利用性，合成透過性，およびプラグイン可能性）の改善が得られるかどうかを評価している [29]．この論文では，23 個の GoF パターンのうち 17 個についてモジュール性の改善が得られることを示している．AspectJ を用いた Observer パターンの実装例を図 4 に示す．図 4 では，Observer パターンの核となる仕組みを `ObserverProtocol` アスペクト，そのアスペクトを具体的に対象プログラムに適用する仕組みを `FigureObservation` アスペクトとしてそれぞれ別個に定義している．これらのアスペクトを実際に適用した場合に得られる設計の例を図 5 に示す [30]．パターンの核となる部分の実装の独立再利用，および，依存性の逆転（適用対象プログラムがパターンに依存するのではなく，パターンの適用方法を定義したアスペクトが適用対象に依存すること）を実現している．

```
public abstract aspect ObserverProtocol {
 protected interface Subject  { }
 protected interface Observer { }
  protected abstract pointcut
    subjectChange(Subject s);
 after(Subject s): subjectChange(s) {
  Iterator iter = getObservers(s).iterator();
  while(iter.hasNext())
   updateObserver(s,(Observer)iter.next()); }
 protected abstract void
  updateObserver(Subject s, Observer o);

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

図 4 Hannemann らによる Observer パターンの AspectJ による定義の一部 [29]

(3) リファクタリング

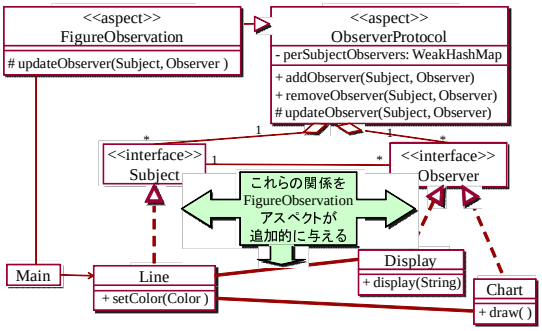


図 5　Observer パターンのアスペクトの適用例 [30]

パターンの適用は，非機能的特性の向上を目的としたプログラム設計を促すため，類似の目的を持つリファクタリングと密接な関係にある．そこで，両者の関係を詳細に明らかにすることが試みられてきている．

Tokuda らは，ソフトウェアが発展する過程においてパターンが設計の目標を与えていると仮定し，基本的なリファクタリングを繰り返し適用することで，既存プログラムの設計を改良する手法を提案している [66]．提案しているリファクタリングは，パラメータ化された Inherit（親子関係を構築），Factory Method（インスタンスを生成するメソッドを追加），Substitute（あるクラスへの参照から別のクラスへの参照への置換），CreateClass（新規クラスの作成），および，CreateInstanceVariable（新規インスタンス変数のクラスへの追加）である．論文では，これらの基本変換を用いて，プログラムコードがパターンを有する形に推移する様子が具体的に示されている．

[36] では，既存の設計（あるいはコード）を改良するために，パターンを標的とした 27 個のリファクタリングをカタログ形式で紹介している．

4.7　デザインパターンの適合性検証

ソフトウェアの形式仕様とは，意味が一意に定まるように数学的に厳密に記述された仕様である．形式仕様記述は，仕様中に数学的な誤りが存在しないことの検証や，プログラムの自動生成などに広く応用されてきている．パターン研究の分野では，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，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，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 [11]．

Konrad らは，組込みシステム開発の分析工程において用いる幾つかのオブジェクト分析パターンが与える分析モデルの雛形について，その状態遷移の仕様を時相論理式によって記述しておくことで，それらのパターンの雛形に基づいて得られたモデルをモデル検査器 SPIN によって自動的に検査する方法を提案している [39]．高品質なソフトウェアを得るための有望な技術が自動検証であり，その成果をパターン指向開発に応用した試みとして興味深い．

4.8　デザインパターンの有効性評価

メトリクスとは，対象の特徴を定性的あるいは定量的に測定する方法と尺度を合わせた概念である測定法（メトリック）の集合である．これまでパターンのコミュニティでは，パターンがもたらす“ 良さ ”は，言葉では表せないものであるとして QWAN（Quality without a Name）と呼ばれていた．これは建築のパターンランゲージにおける概念を受けついだものであるが，測定と評価がその実施と発展に不可欠なソフトウェア“ 工学（エンジニアリング）”の枠組みにおいては，扱いにくいとの考え方がある．そこで，パターンの特性や与える影響を定量的に調査し評価するために，メトリクス技術が用いられてきている．

Prechelt らは，小規模な C++プログラムの修正作業におけるパターンの効果を，実験に基づいて定量的に評価している [54]．複数のソフトウェア開発者を被験者として，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，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パターンの効果を，メトリクスを用いて評価している [44]．同一のプログラムに対して，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，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 [13] 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．

文献 [71] では，適切なパターンに基づく設計の方が，場当たり的な設計に比べて欠陥が少ないことを仮定して，商用アプリケーションにおける欠陥率の測定によりその仮定が成り立つかどうかを分析した結果が述べられている．測定においては，スケーラビリティと精度を十分に考慮したシステムを構築している．実験では，5 つのパターン (Singleton，Template Method，Decorator，Observer，Factory) とそれらの組み合わせに対して，ロジスティック回帰分析を適用して以下を報告している．以下の報告は，パターンを適用すると設計が良くなるという主張に対して，それが事実あるいは“ 神話 ”なのかを示す一つの指標になる．

- 一般的なコードと比較して，Factory パターンを適用したコードは欠陥率が低いが，Observer パターンを適用したコードは欠陥率が高い．
- Singleton パターンと Observer パターンにおいては，それらのパターンのサイズの増加と欠陥頻度に相関が見られる．
- Template Method パターンに関しては，さまざまな状況において何度も適用されているため，欠陥頻度に関する明確な傾向は見られない．

Bieman らは，パターンが与える可変性を調査した結果を報告している [7]．論文では，パターンが与える可変性は，パターンのロールを担うクラス（パターンクラス）そのものの変更ではなく，サブクラスの追加・修正によって実現されるものと仮定して，幾つかの実用プログラムの改変履歴を調査したところ，パターンクラスの方がより変更される傾向にあり，先の仮定は当てはまらなかったことを報告している．

5 おわりに

本論文では，デザインパターンに対する主要なソフトウェア工学的取り組みとして得られている活用支援技術群を解説した．また，デザインパターンを扱う活動を構成するプロセスをモデル化したうえで，プロセスを構成する活動や作業と，それらの活用支援技術を対応付けて整理した．

ここまで取り上げたように，デザインパターンの活動支援に応用された技術（およびパターンの概念を導入した分野）は広範にわたって存在する．今後もデザインパターンへのソフトウェア工学的取り組みは，ソフトウェア工学やプログラミングといった研究分野における様々な成果を取り込むと同時に，デザインパターン研究の成果が他の分野に影響を与えるといった相互に作用した発展が期待される．その成果は様々な会議や論文誌において“ 横断的に ”発表されるため，デザインパターン研究の分野の進展を概観するためには，今後も本論文と同様に様々な情報源についての継続的な調査が必要である．

謝辞 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における研究調査にご参画いただいた下滝亜里氏, 藤枝和宏氏に感謝します．本研究の一部は日揮・実吉奨学会および国際科学技術財団の助成を受けました．貴重なご指摘をいただいた匿名の査読者に感謝します．

参 考 文 献

[1] Herve Albin-Amiot and Yann-Gael Gueheneuc: Meta-modeling Design Patterns: aplication to pattern detection and code synthesis, Proceedings of the Workshop on Adaptative Object-Models and Metamodeling Techniques at ECOOP'01, 2001.

[2] Christopher Alexander, Sara Ishikawa, Murray Silverstein: A Pattern Language: Towns, Buildings, Construction, Oxford University Press, 1977. (平田翰那 訳: パタン・ランゲージ, 鹿島出版, 1977.)

[3] Deepak Alur, John Crupi, Dan Malks: Core J2EE Patterns: Best Practices and Design Strategies, Pearson Education, 2001. (中野明彦, 佐野祐一郎, 宮田泰宏, 土屋聡一郎 訳: J2EE パターン: 明暗をわける設計の戦略, ピアソンエデュケーション, 2002.)

[4] Gabriela Arevalo, Frank Buchli and Oscar Nierstrasz: Detecting Implicit Collaboration Patterns, Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04), pp.122-131, 2004.
[5] Kent Beck and Ward Cunningham: Using a Pattern Language for Programming, ACM SIGPLAN Notices, 23(5), pp.27-31, 1987.
[6] Kent Beck and Ralph Johnson: Patterns Generate Architectures, Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP'94), pp.139-149, 1994.
[7] James M. Bieman, Greg Straw, Huxia Wang, P. Willard Munger and Roger T. Alexander: Design Patterns and Change Proneness: An Examination of Five Evolving Systems, Proceedings of the 9th International Symposium on Software Metrics (Metrics 2003), pp.40-49, 2003.
[8] Alex Blewitt, Alan Bundy and Ian Stark: Automatic verification of Java design patterns, Proceedings of the International Conference on Automated Software Engineering, pp.324-327, 2001.
[9] Frank Budinsky, Marilyn Finnie, Marilyn Finnie, John Vlissides and Patsy Yu: Automatic code generation from design patterns, IBM Systems Journal, 35(2), pp.151-171, 1996.
[10] Frank Buschmann, Hans Rohnert, Michael Stal, Regine Meunier, Peter Sommerlad: Pattern-Oriented Software Architecture - A System of Patterns, Wiley, 1996. (金澤典子, 桜井麻里, 千葉寛之, 水野貴之, 関富登志 訳: ソフトウェアアーキテクチャ: ソフトウェア開発のためのパターン体系, 近代科学社, 2000.)
[11] Alejandra Cechich and Richard Moore: A Formal Basis for Object-Oriented Patterns, Proceedings of the Sixth Asia Pacific Software Engineering Conference (APSEC'99), pp.284-291, 1999.
[12] C. Chambers, Bill Harrison, John Vlissideset: A Debate on Language and Tool Support for Design Patterns, Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'00), pp.277-289, 2000.
[13] Shyam Chidamber and Chris Kemerer: A Metrics Suite for Object Oriented Design, IEEE Trans. on SE, 20(6), pp.476-493, 1994.
[14] Peter Coad: Object Oriented Patterns, Communications of the ACM, 35(9), pp.152-159, 1992.
[15] Jim Coplien: Advanced C++ Programming Styles and Idioms, Addison-Wesley, 1992.
[16] James Coplien: Software Design Patterns: Common Questions and Answers, In The Patterns Handbook: Techniques, Strategies, and Applications, Cambridge University Press, 1998.
[17] James Coplien and Liping Zhao: Symmetry and Symmetry Breaking in Software Patterns, Proceedings of the Second International Symposium on Generative and Component-Based Software Engineering (GCSE'00), LNCS 2177, pp.37-56, 2000.
[18] Daniel von Dincklage: Making Patterns Explicit with Metaprogramming, Proceedings of the 2nd international conference on Generative programming and component engineering (GPCE'03), pp.287-306, 2003.
[19] Jing Dong, Dushyant S. Lad and Yajing Zhao: DP-Miner: Design Pattern Discovery Using Matrix, Proceedings of the Fourteenth Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS), pp.371-380, 2007.
[20] Jing Dong, Yajing Zhao, Tu Peng: A Review of Design Pattern Mining Techniques, International Journal of Software Engineering and Knowledge Engineering, Vol.19, No.6, pp.823-855, 2009.
[21] Amnon Eden, Yoram Hirshfeld and Amiram Yehudai: Multicast - Observer ≠ Typed Message, C++ Report, Vol.10, No.9, pp.33-38, 1998.
[22] Amnon Eden: LePUS: A Visual Formalism For Object-Oriented Architecture, Proceedings of the 6th World Conf. Integrated Design and Process Technology (IDPT 2002), pp.22-28, 2002.
[23] 江渡浩一郎: パターン，Wiki，XP - 時を超えた創造の原則, 技術評論社, 2009.
[24] Martin Fowler: Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002. (株式会社テクノロジックアート訳, 長瀬嘉秀 監訳: 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, 翔泳社, 2005.)
[25] Eric Gamma: Object-Oriented Software Development Based on ET++: Design Patterns, Class Library, Tools, PhD Thesis, University of Zurich, 1991.
[26] Eric Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns: Abstraction and Reuse of Object-Oriented Design, Proceedings of the 7th European Conference on Object-Oriented Programming (ECOOP'93), LNCS 707, pp.406-431, 1993.
[27] Eric Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addision-Wesley, 1994. (本位田真一, 吉田和樹 監訳: 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, ソフトバンククリエイティブ, 1999.)
[28] Yann-Gael Gueheneuc, Houari Sahraoui and Farouk Zaidi: Fingerprinting Design Patterns, Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04), pp.172-181, 2004.
[29] Jan Hannemann and Gregor Kiczales: Design Pattern Implementation in Java and AspectJ, Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming (OOPSLA'02), pp.161-173, 2002.
[30] 羽生田栄一 監修, ソフトウェアパターン入門 - 基礎から応用へ, ソフトリサーチセンタ, 2005.
[31] Shinpei Hayashi, Junya Katada, Ryota Sakamoto, Takashi Kobayashi, Motoshi Saeki: Design Pattern Detection by Using Meta Patterns, IEICE Transactions on Information and Systems, vol.E91-D, no.4, pp.933-944, 2008.

[32] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom and Welf Lowe: Automatic Design Pattern Detection, Proceedings of the Workshop on Program Comprehension at ICSE'03, pp.94-103, 2003.
[33] Nien-Lin Hsueh and Wen-Hsiang Shen: Handling Nonfunctional and Conflicting Requirements with Design Patterns, Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004), pp.608-615, 2004.
[34] Yasushi Kambayashi and Mikio Ohki: Extracting the Software Elements and Design Patterns from the Software Field, Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS'03), pp.603-608, 2003.
[35] Rudolf Keller, Reinhard Schauer, Sebastien Robitaille and Patrick Page: Pattern-Based Reverse-Engineering of Design Components, Proceedings of the 21st international conference on Software engineering (ICSE'99), pp.226-235, 1999.
[36] Joshua Kerievsky: Refactoring to Patterns, Addison-Wesley Professional, 2004.
[37] 小林隆志, 佐伯元司: デザインパターンのオブジェクト指向モデル化と支援ツールへの応用, 日本ソフトウェア科学会 コンピュータソフトウェア, Vol.21, No.1, pp.60-75, 2004.
[38] 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.
[39] Sascha Konrad, Betty H. C. Cheng and Laura A. Campbell: Object Analysis Patterns for Embedded Systems, IEEE Trans. Soft. Eng., 30(12), pp.970-992, 2004
[40] Christian Kramer and Lutz Prechelt: Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software, Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE'96), pp.208-215, 1996.
[41] Atsuto Kubo, Hironori Washizaki, Atsuhiro Takasu, and Yoshiaki Fukazawa: Extracting Relations among Embedded Software Design Patterns, Integrated Design and Process Science, Vol.9, No.3, pp.39-52, 2006.
[42] Anthony Lauder and Stuart Kent: Precise Visual Specification of Design Patterns, Proceedings of the 12th European Conference on Object-Oriented Programming (ECOOP'98), LNCS 1445, pp.114-134, 1998.
[43] Doug Lea: Concurrent Programming in Java: Design Principles and Patterns, Second edition, Addison-Wesley, 1999.
[44] 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), pp.4-18, 2000.
[45] Peter C. Mehlitz and John Penix: Design for Verification Using Design Patterns to Build Reliable Systems, Proceedings of the Workshop on Component-Based Software Engineering (CBSE) at International Conference on Software Engineering, 2003.
[46] 水野恵祐: デザインパターン検出能力の向上を目的とした複数検出手法の併用, 奈良先端科学技術大学院大学情報科学研究科 修士論文, 2010.
[47] 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.
[48] 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, pp.157-164, 2000.
[49] Jorg Niere, Wilhelm Schafer, Jorg P. Wadsack, Lothar Wendehals and Jim Welsh: Towards Pattern-Based Design Recovery, Proceedings of the 24th International Conference on Software Engineering (ICSE'02), pp.338-348, 2002.
[50] James Noblea and Robert Biddle: Patterns as Signs, Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP'02), LNCS 2374, pp.368-391, 2002.
[51] Mika Ohtsuki and Norihiko Yoshida: A Source Code Generation Support System Using Design Pattern Documents Based on SGML, Proceedings of the Sixth Asia Pacific Software Engineering Conference, APSEC'99, pp.292-299, 1999.
[52] Wegrzynowicz Patrycja and Stencel Krzysztof: Towards a Comprehensive Test Suite for Detectors of Design Patterns, Proceedings of the 24th ACM/IEEE International Conference on Automated Software Engineering (ASE), pp.103-110, 2009.
[53] Daryl Posnett, Christian Bird, and Premkumar Devanbu: THEX: Mining Metapatterns from Java, Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp.122-125, 2010.
[54] 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), pp.1134-1144, 2001.
[55] Wolfgang Pree: Design Patterns for Object-Oriented Software Development, Addison-Wesley, 1996.
[56] Dirk Riehle: Composite Design Patterns, Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97), pp.218-228, 1997.
[57] Robert C. Martin, 瀬谷啓介 訳: アジャイルソフトウェア開発の奥義, ソフトバンククリエイティブ, 2004.
[58] Johannes Sametinger: Design Composition, Journal of Computer Science and Technology, 3(1), pp.27-33, 2003.
[59] Buntarou Shizuki, Masashi Toyoda, Etsuya Shibayama and Shin Takahashi: Smart Browsing 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), pp.529-548, 2000.
[60] Neelam Soundarajan and Jason O. Hallstrom: Responsibilities and Rewards: Specifying Design

Patterns, Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), pp.666-675, 2004.
[61] 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, 日科技連出版社, 2000.
[62] 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, Vol.44 No.SIG 4(PRO 17), pp.25-46, 2003.
[63] Michiaki Tatsubori and Shigeru Chiba: Programming Support of Design Patterns with Compile-time Reflection, Proceedings of the Workshop on Reflective Programming in C++ and Java at OOPSLA'98, pp.56-60, 1998.
[64] Masao Tomono, Tatsuo Kato and FUJINO Terunobu: On the way to establishing a jPLoP community, Proceedings of the Sixth Asia Pacific Software Engineering Conference (APSEC'99), pp.546-549, 1999.
[65] Paolo Tonella and Giulio Antoniol: Object Oriented Design Pattern Inference, Proceedings of the IEEE International Conference on Software Maintenance (ICSM'99), pp.230, 1999.
[66] Lance Tokuda and Don Batory: Evolving Object-Oriented Designs with Refactorings, Automated Software Engineering, 8(1), pp.89-120, 2001.
[67] Will Tracz: RMISE Workshop on Software Reuse Meeting Summary, In Software Reuse: Emerging Technology, IEEE CS, 1988.
[68] Satoru Uchiyama, Atsuto Kubo, Hironori Washizaki, Yoshiaki Fukazawa: Design Pattern Detection using Software Metrics and Machine Learning, Proceedings of the Fifth International Workshop on Software Quality and Maintainability (SQM2011), 2011.
[69] 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.
[70] John Vlissides: Multicast - Observer = Typed Message, C++ Report, Vol.10, No.9, 1997.
[71] Marek Vokac: Defect Frequency and Design Patterns: An Empirical Study of Industrial Code, IEEE Trans. Soft. Eng., 30(12), pp.904-917, 2004.
[72] 鷲崎弘宜, 深澤良彰: ソフトウェアパターン研究の現在と未来, 情報処理学会研究会報告, SE-141, pp.31-38, 2003.
[73] 鷲崎弘宜, 大杉直樹, 権藤克彦, 服部哲, 久保淳人, 下滝亜里, 小林隆志, 藤枝和宏, 大月美佳, 丸山勝久, 榊原彰: ソフトウェアパターン研究の発展経緯と最近の動向, 情報処理学会研究会報告, SE-147, pp.127-134, 2005.
[74] 鷲崎弘宜, 深谷和宏, 久保淳人, 深澤良彰, パターン適用前のソースコードを用いたデザインパターン検出, コンピュータソフトウェア, Vol.27, No.2, pp.136-141, 2010.
[75] 鷲崎弘宜: 小特集 ソフトウェアパターン- 時を超えるソフトウェアの道 -, 情報処理, Vol.52, No.9, 2011.
[76] 鷲崎弘宜: ソフトウェアパターン概観, 情報処理, Vol.52, No.9, pp.1119-1126, 2011.
[77] T. Winn and P. Calder: Is This a Pattern?, IEEE Software, 19(1), pp.59-66, 2002.
[78] Roel Wuyts: Declarative Reasoning about the Structure of Object-Oriented Systems, Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS'98), pp.112-124, 1998.
[79] 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.
[80] Nobukazu Yoshioka, Hironori Washizaki, Katsuhisa Maruyama: A survey on security patterns, Progress in Informatics, No.5, pp.35-47, 2008.