連載解説

パターン

ーソフトウェア開発ノウハウの再利用

第1回　パターンの発展と現状

中山裕子　（株）富士通研究所
山本里枝子　（株）富士通研究所
吉田裕之　富士通（株）

ソフトウェアのパターンとは

ソフトウェア開発に携わる方は，最近「パターン」という言葉をよく聞かれるのではないだろうか．しかし，ソフトウェア開発のパターンと聞いて，どんなものか想像がつかないと思っておられる方も多いだろう．今号では，ソフトウェアのパターンとは何か，なぜ注目を集めているか，またどのような経緯で発展してきたかを解説したい．

一般にパターンという言葉には，型，思考や行動の様式，見本などの意味がある．ソフトウェアのパターンも，こうした一般的な意味でも解釈できるが，特に重要なのは「繰り返し使われる」ということである．

長年ソフトウェアを開発してきた方なら，他の人から苦労話を聞かされて，それは自分も経験したと思ったことが一度や二度はあるのではないだろうか．そして，その人も長い時間をかけて自分と同じ解決策に至ったことを知り，「先に話を聞いていれば，助けになったかもしれないのに」と考えたことも何度かあるのではないだろうか．こうしたソフトウェア開発で繰り返し起こる問題と，それに対して皆が繰り返し使っている解法を共有し再利用するのがパターンの目的である．

パターンは特に，今まで熟練者頼みにしてきた知識をうまく再利用できる方法として注目されている．ソフトウェア開発に必要な知識のうち，一部の知識，たとえばアルゴリズムやデータ構造などは，すでに広く共有され大学の教育課程にも組み込まれている．しかし他の知識，たとえば分析のノウハウや設計のテクニック，プロジェクト管理の知恵などは，各自が経験の中から学んでいるのが実状で，なかなかうまく共有できていない．パターンは，こうした知識を理解しやすい表現で簡潔に書いたものである．

パターンはどこで生まれたか

知識をパターンという形で再利用しようというアイデアは，元は建築分野で生まれた．1970年代後半，University of California at Berkleyの建築学の教授Christopher Alexanderが，パターン言語（pattern language）に関する4冊の本を発表した．その中の1冊，「パタン・ランゲージ」[1]には，253のパターンが入っている．Alexanderがランゲージ（言語）という言葉を使ったのは，構成要素である各パターンがボキャブラリとなり，パターン間の組合せやアレンジのためのルールが文法となる，と考えたためである．「パタン・ランゲージ」では，1つのパターン言語で，地域や町の計画から部屋の内装など細部に至るまでを網羅している．各パターンには，そのパターンを適用するための前提，課題，解法，および関連パターンを数ページで簡潔に書いてあ

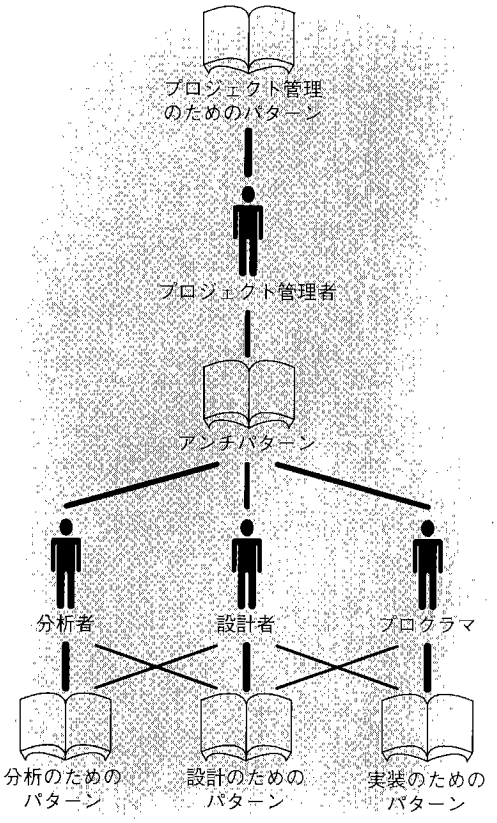


図-1　パターンの利用者とパターンの分類

る．また，ある課題に着目すると，その課題を解くパターンからより小規模な対象を扱う関連パターンへと連鎖的に考えをブレークダウンできるようになっている．「パタン・ランゲージ」は，Alexanderと共著者たちの知識をエッセンスにした，まさにノウハウの集大成といえる．Alexanderの一連の著作は建築分野に大きな影響を与え，今日に至るまで建築関係の学科の教科書として広く使われているようである．

Alexanderのアイデアが，ソフトウェアの研究者たちにも影響を与え始めたのが，1980年代の半ばである．まずTektronix研究所のグループが，オブジェクト指向技術の国際学会OOPSLA（Object-Oriented Programming, Systems, Languages and Applications）で，オブジェクト指向開発のためのパターン言語を作ることを提案した．また同じころ，Erich Gammaは，自分の使った設計手法の中に他の人たちも繰り返し使っているものがあることに着目した．彼は，その設計手法を「デザインパターン」と呼び，博士論文の中にその一覧を示した（Gammaは，後年「デザインパターン」[2]の著者の1人になる）．

1990年代に入るとこれらの成果がOOPSLAに集まり，パターン言語に対する取組みが一気に加速した．1994年には，パターンのための会議，PLoP（Pattern Languages of Programs）Conference[3]が始まった．PLoPは毎年夏に米国イリノイ州で開催され，パターンの発展に中心的な役割を果たしている．

どのようなパターンがあるのか

では，パターンとは具体的にどのようなものか，ここでいくつか例を紹介したい．パターンというと，「デザインパターン」[2]を思い浮かべる方が多いと思う．確かに「デザインパターン」は最も優れたパターン集の1つだが，他にも多種多様なパターンがある．以下では，パターンが対象にしている作業を，分析，設計，実装，プロジェクト管理の4つに分類し，それぞれ代表例を紹介する．また，アンチパターンという新しい概念も簡単に紹介する．

図-1に示したように，分析，設計，実装のためのパターンは，主にこれらの作業の担当者を対象に書かれたものである．また，プロジェクト管理のためのパターンはプロジェクト管理者を対象にしている．アンチパターンには，ソフトウェア開発，ソフトウェアのアーキテクチャ，プロジェクト管理の3種類のパターンが入っていて，すべての開発関係者を対象にしたパターン集になっている．

分析のためのパターン

経験の浅いチームが分析をすると，必ずといっていいほど「何をオブジェクトやエンティティにすればいいか分からない」という声が上がる．また，一通り方法論や表記法を勉強した人が次に発する質問も，たいていこれだといっていい．特にオブジェクト指向開発では，分析で作るモデルがその後の作業の土台になるので，それをどう作っていいか分からないというのは大問題である．良いモデルを作るには，ソフトウェアが対象とする分野や業務（ドメインと呼ぶ）とモデリング技法（UML（Unified Modeling Language）やEntity-Relationship図など）の両方の知識がなくてはならない．しかし現実には，この両者に精通した熟練者は数少ないし，新たに育てるだけの時間的，経済的な余裕もない．分析のためのパターンは，この問題を解決するために現れた．

分析のためのパターンには，対象

世界のどこをどういう視点で見て，どうモデルに表せばいいかが書いてある．特定の分野や業務の世界そのものが課題になるので，パターンもそのドメインに特化したものになる傾向がある（特定ドメインを対象にしたパターンについては，「ドメイン指向パターン」[4]に詳しく述べた）．ここでは，代表的なパターン集として，Fowlerの「アナリシスパターン」[5]を紹介する．

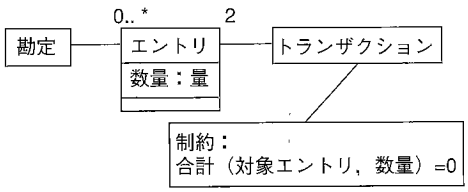
「アナリシスパターン」は，ビジネスアプリケーションを対象に，オブジェクト指向モデルを作るためのノウハウを集めたものである．在庫管理や会計，金融派生商品などでの経験を基にしているが，パターンは複数のビジネスアプリケーションドメインで共通に使えるように抽象化してある（中には，「先物取引」パターンのように，特定のドメインを対象にしたものもいくつかある）．

図-2の例は，「トランザクション」パターンの「エントリを2つ持つトランザクション」モデルである（FowlerはクラスをMartin/Odellの表記法で書いているが，図-2ではUMLで書き直した）．ここでのトランザクションは，ビジネス・トランザクション，つまり企業と顧客との間のやりとりのことで，トランザクション処理（TP）のトランザクションではない．

Fowlerは，企業活動という対象世界を見るときのノウハウとして，「金やものを入れるいくつかの入れ物があり，金やものが入れ物の間をどう移動するかを記録するのが基本である」としている．トランザクションは，この移動を記録するためのオブジェクトである．図-2の「エントリを2つ持つトランザクション」は，勘定の払出（移動元）と他の勘定への繰入（移動先）がそれぞれ1つずつの場合は，図中のクラス図のようにモデルを作るといい，と教えている．

図中の「勘定」は，金額やものの個数を持っていて，「エントリ」は勘定の増減を記録している．また，トランザクションオブジェクトは，金やものの移動の記録（移動元と移動先の2つのエントリを関連づけること）に責任があると同時に，移動間の差引きが必ず0になることをチェックする責任がある．たとえば図中の例のように，ある当座預金口座から500ドルの引落し（移動元のマイナスのエントリ）があったら，対応して500ドルの支払い（移動先のプラスのエントリ）があり，2つのエントリの数値は必ず差引き0ドルになることを保証する．

Fowlerのアナリシスパターンには，この例のように，さまざまな業務に共通に使える抽象度が高いものが多い．本連載の第4回では，Fowlerのパターンと他のパターンを比較しながら，アナリシスパターンをさらに詳しく解説する．



（例）　筆者は航空券を買うためにボストン航空に500ドルをクレジットカードで払う．これは，クレジットカード勘定からボストン航空勘定への500ドルのトランザクションである．後日，筆者は当座預金からクレジット勘定へ，クレジット勘定のバランスを0にするためのトランザクションを作成することになるだろう[5]．

（例）　アロマコーヒーメーカ（ACM）社は，5トンのアラビアンモカをニューヨークからボストンへ移送する．これは，ニューヨーク勘定からボストン勘定への5トンのトランザクションになる[5]．

（モデリングの原則）　勘定を使うときは，保全原理（勘定の対象項目は，生成も消滅もせず，場所が移り変わるだけである）に従え．これによって，漏れの発見および防止が容易となる[5]．

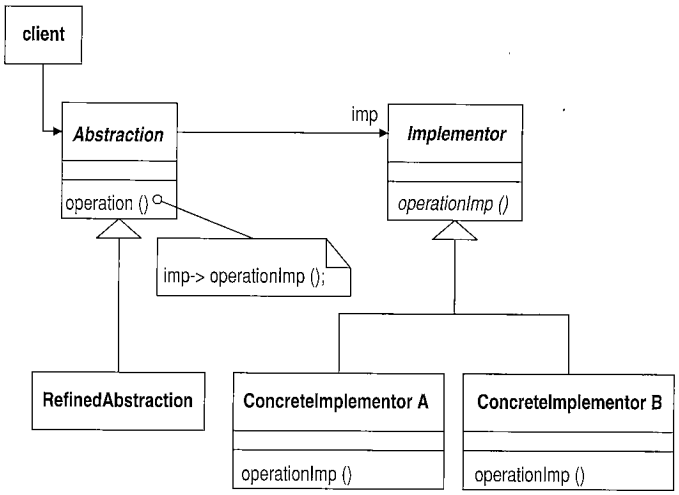
図-2　エントリを2つ持つトランザクション

設計のためのパターン

設計のためのパターンには，大別して，システム全体の構造に関するパターン（アーキテクチャパターンとも呼ぶ）と，部分的な構造に関するものの2種類がある．ここでは後者の代表例として，Gamma, Helm, Johnson, Vlissidesの4人（通称Gang of Four）による「デザインパターン」[2]を紹介する．この本は，変更が容易で再利用性の高いオブジェクト指向ソフトウェアを設計するためのテクニックを集めたものである．

「デザインパターン」が出版されて5年以上が経ったいま，その中のパターン（SingletonやBridge，Strategyなど）は，オブジェクト指向設計のボキャブラリとして一般常識化しつつある．なぜここまで広くかつ迅速に普及したのか，その理由は大きく3つある．1点目は，従来使われていたテクニックに名前をつけ，皆が共通のボキャブラリを持てるようにしたこと，2点目はたいへん使いやすい形で提供したこと，そして3点目は当然ながら内容が優れていたこと，である．

1点目のボキャブラリであるが，皆が繰り返し使う設計テクニックに名前がついたことで，コミュニケーションが格段に楽になった．ちょうどアルゴリズムについて話すときにヒープソートやクイックソートといえば話が通じるように，設計テクニック（クラス構造とその間の協調関



協調関係：Abstractionクラスは，クライアントからの要求をImplementorのオブジェクトへ転送する．

図-3　Bridgeパターンの構造と協調関係

係）とそれが持つ意図や効果を一言で伝えられるようになったのである．かつては「ここをスーパークラスにしてその下にサブクラスを作って，こういうメソッドの呼出しをすれば」と絵を描いて長々説明しなければならなかったのに，今では「Bridgeパターンを使えば」といえば済むようになった．

2点目の使いやすさは，パターンを提供するスタイルを確立することで実現された．「デザインパターン」のスタイルは，「カタログ」と呼ばれ，後の多くのパターン集に影響を与えた．ここでは，パターンの利用法を示しながら，そのスタイルを説明する．

パターンの利用者は，たいていまず「デザインパターン」の見返しを開く．そこには3つのカテゴリ（生成，構造，振る舞い）に分類した23個のパターンの一覧が載っている．一覧にはパターンの名前と目的，掲載ページを示す数字が書いてある．利用者はこの中から適当なパターンを選び，掲載ページを開く．開いた先のページには，各パターンが統一したフォーマット（テンプレートと呼ぶ）に則って記述してある．パターン1件あたり10数ページの量なので，短時間に理解できる．

「デザインパターン」のテンプレートには，「パターン名と分類」，「目的」，「別名」，「動機」，「適用可能性」，「構造」，「構成要素」，「協調関係」，「結果」，「実装」，「サンプルコード」，「使用例」，「関連するパターン」の13項目がある．**図-3**にBridgeパターンを例として，構造と協調関係の部分を示す．Bridgeパターンの目的は「抽出されたクラスと実装を分離して，それらを独立に変更できるようにする」[2]ことである．図中のイタリックで示したクラスは抽象クラス，同じくイタリックで示したメソッドは抽象メソッドである．Abstractionクラスでインタフェースを定義し，インタフェースの実装をImplementorクラスに分離する．Implementorのサブクラスを実行時に決定できるので，実装の変更や拡張が容易になる．

3点目の優れた内容ということに関しては，まず，テーマを「変更が容易で再利用性の高いオブジェクト指向ソフトウェアの設計」に絞って，利用頻度の高いパターンを集めた点にある．また，パターン間の関連を明確にして，いくつかのパターンを比較したり組み合わせたりすることを容易にしている．さらに重要な点は，すべてのパターンが一貫して次の2つの原理に基づいていることである．

1. インタフェースに対してプログラミングするのであり，実装そのものをプログラミングするのではない[2]．
2. クラス継承よりもオブジェクトコンポジションを多用すること[2]．

見方を変えれば，各パターンがこの2つの原理を学ぶための例になっている．「デザインパターン」は，オブジェクト指向設計の真髄を示した優れた教科書でもある．

実装のためのパターン

実装（プログラミング）のためのパターンは，あるプログラミング言語に特有のノウハウを書いたものである．イディオムとも呼ぶ．すでに，C++, Java, Smalltalkを対象としたパターン集が出版されている．

実装作業でも，プログラミング言語の文法を知っているからといって，質の高いプログラムが書けるわけではない．実装のためのパターンは，良いプログラムを書くために次の2種類の知識を示している．

1. プログラミングのテクニック．実行性能や変更容易性などの優れたプログラムを書くための方法，ライブラリの効果的な使い方などを含む．
2. 記述のスタイル．理解しやすいプログラムにするための，名前の付け方，空白の使い方などを含む．

ここでは，Kent Beckが書いた「Smalltalk Best Practice Patterns」[6]を紹介する．

Beckが各パターンを書くのに使っているテンプレートには，「タ

イトル」,「先行パターン」,「問題」,「影響する事柄」,「解法」,「議論」,「後続パターン」の7つの項目がある.「デザインパターン」[2]のテンプレートと比較すると項目数が少ない.これは対象をSmalltalkに限定しているので細かい説明が不要なためと,扱う問題の規模が小さいためである.

各パターンは,テンプレートに則った簡潔な表現で,数ページに収まっている.また,Alexanderの「パタン・ランゲージ」[1]と同様,課題をブレークダウンしながらパターンを順々に適用できる.たとえば,Collectionカテゴリの最初のパターンCollectionは,「1対多の関係をどう表したらいいか?」が問題で,その解は「Collectionを使いなさい」というきわめて基本的なものである.このCollectionパターンを出発点に,要素の数は固定か可変か,要素がユニークでなくてはならないか,など問題をさらに掘り下げながら最適なパターンを探せるようになっている.

「Smalltalk Best Practice Patterns」は,従来の文法書とは異なる,新たなプログラミングの教科書の好例といえる.今後,他のプログラミング言語でも,このようなパターン集が出版されることを期待したい.

プロジェクト管理のためのパターン

以上に紹介した3種類のパターンは,いずれも開発の成果物に関するものである.次に紹介するのは,プロジェクトを成功に導くためのノウハウを集めたパターン集である.開発プロセスやチームの構成などについて記述しているので,「組織とプロセスのパターン」とも呼ぶ.プロジェクト管理についてのノウハウは,今まで多くの本の中で語られてきた.それをパターンの形にしたことの意義は,ハンドブックとしてより使いやすくなったことだろう.ここでは,代表例として,James Coplienの「A Generative Development - Process Pattern Language」[7]を紹介する.

タイトルの"Generative"(生成的)とは,良いプロセスや組織そのものではなく,それを生成する方法を示すことを意味している.どのようなプロセスや組織が良いかは状況によって違うので,固定的な形では示せない.しかし,生成にあたって考慮すべきことは再利用可能である.Coplienはこれを,統一したテンプレートを用いて,42のパターンに示した.このうち,11が開発プロセスに関するもの(process pattern)で,残りの31が組織に関するもの(organizational pattern)である.

開発プロセスに関するパターンには,たとえば,"Size the Schedule"(スケジュールの長さを決める)がある.このパターンの課題は,スケジュールが長すぎても短すぎても市場の要求を満たすものは作れないことである.そして,これに対して解決策を2つ示している.1つは開発者がスケジュールを守ったときに報酬を与えること,もう1つは市場用と内部の開発者用との2つのスケジュールを立て,後者を前者より短くしておくこと,である.

また組織に関するパターンには,たとえば,"Size the Organization"(組織の大きさを決める)がある.このパターンの課題は,組織についても,大きすぎても小さすぎても開発がうまくいかないことである.これに対する解決策には,Coplienが経験から得た具体的な数字を示している.彼の経験によると,教育を受け,かつプロジェクトに適した10人の開発者を選んでチームを作る場合,1,500KSLOC(Kilo Source Line Of Codes)のプロジェクトなら31カ月,200KSLOCなら15カ月,60KSLOCなら8カ月で開発できる[7].また,開発者に割り当てられた1つの役割につき,他の6つないし7つの役割とは相互関係を維持できる[7].したがって,10人のプロジェクトならほぼ全員でコミュニケーションをとることが可能で,これによって皆が「グローバルな知識」[7]を持てる.

アンチパターン

パターンがソフト開発で多くの人が直面する課題とそれに対する解法なのに対し,アンチパターンはソフト開発で多くの人が陥ってしまう失敗とそれに対する対策を,テンプレートに則って記述する(「アンチ」はパターン自体に反対すること意味するものではない).

アンチパターンは比較的新しい概念で,1998年にBrownらが「アンチパターン」[8]を出版した.アンチパターンには,たとえば,「肥満児(The Blob)」(なんでも抱え込んだ巨大なクラスを作ってしまうこと)や「分析地獄(Analysis Paralysis)」(分析の完全さにこだわって,分析作業にはまり込んでしまうこと)といったものがある.パターンの名前も文章もユーモアに富んでいるが,内容は辛口である.耳の痛い話ばかりなので,ユーモアのオブラートで包まなければ受け入れ難いものになってしまったろう.

アンチパターンの重要な点は,失敗から回復する方法を順序立てて説明していることである.これには2つの使い方がある.1つは,完全に失敗してから修正するために適用するもので,もう1つは,兆候が現れた時点で解決策を使うものである.後者の方が良いことはいうまでもない.

なお,アンチパターンの基になった「リファクタリング」という概念を,本連載の第3回で解説する.

以上5つのパターン集を紹介してきたが，他にも数多くのパターン集が発表されている．そのうちの多くは，PLoP Conference Home Page[3]やPatterns Home Page[9]で紹介されており，一部はWWWで見ることができる．

パターンの特徴

ここまでパターンの例を見てきて，一口にパターンといっても内容や形式はさまざまだということがお分かりいただけたと思う．パターンを厳密に定義するのはなかなか難しいのだが，Ralph E. Johnson（「デザインパターン」[2]の著者の1人）が「パターンとフレームワーク」[10]の中でパターンの特徴を分かりやすく整理している．今回の解説のまとめとして，以下にその要約を示す．

1. **一種の作品集（literature）である．**
   パターン集は，ソフトウェアアーキテクトのためのハンドブックになる．検索しやすく，また理解しやすいものにするには，すべてのパターンを同じフォーマット（テンプレートと呼ぶ）で書いておくことが重要である．また，パターン間の関係を示しておくことも大切である．パターンは，こうした独特の著作物としての形式（literary style）を持つ．
2. **トレードオフを示している．**
   パターンは皆が繰り返し使っている解法であるが，万能ではない．たとえば，あるパターンはソフトウェアの変更を容易にするが，代わりに実行速度を若干落とすかもしれない．パターンには，このようなトレードオフが明確に書かれ，利用者が最適なパターンを選べるようになっている．
3. **著者による使用を除き，少なくとも3つの使用例がある．**
   これを「3の法則（Rule of Three）」と呼ぶ．パターンは皆が繰り返し使っている方法のはずだから，それを証明しなくてはならない．パターンには，必ず3つ以上の使用例を書かなくてはならない．Johnsonは，「パターンは発見するものであり，発明するものではない」[10]としている．
4. **抽象化された（広義の）テンプレートである．**
   パターンは，ある技法の型（template）である（ここでのテンプレートは，パターンのフォーマットを指すものではない）．たとえば，「デザインパターン」[2]は，クラス間の静的な構造と動的な協調関係を抽象化したものである．適用する際には，クラスや操作に名前を与えたり，クラスを追加したりして，具体化しなければならない．
5. **パターンは開発を自動化するものではない．**
   一部のパターンは自動化可能かもしれないが，すべては自動化できない．パターンを適用するには，課題やトレードオフを理解し，抽象化された解法を具体化しなければならない．パターンは開発者の思考力を高めるもので，考えずに済ますためのものではない．
6. **ソフトウェア開発のためのボキャブラリを作る．**
   皆がソフトウェア開発に利用してきた解法に名前を付けることで，短時間でスムーズにコミュニケーションできる．前述したように，「ここにBridgeパターン[2]を使う」というだけでクラス構造や協調関係，効果などを伝えることができる．また，ボキャブラリの数で，知識の量を計れるようにもなる．
7. **経験から得た知識の再利用を可能にする．**
   ソフトウェア開発を成功させるには，広い知識を持ちシステム全体を理解できるアーキテクトが必要である．パターンは，経験的に得られた解法を再利用しやすい形式で示し，より短い期間でアーキテクトが必要な知識を得られるようにする．

参考文献

1) Alexander, C. I., Silverstein, M. with Jacobson, M., Fiksdahl-King, I. and Angel, S.: A Pattern Language: Towns, Buildings, Construction, Oxford University Press, New York (1977). (平田翰那 (訳): パタン・ランゲージ, 鹿島出版会 (1984)).
2) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, Massachusettsu (1995). (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, ソフトバンク, 東京 (1995)).
3) Pattern Languages of Programs (PLoP) Conference Home Page: <http://st-www.cs.uiuc.edu/~plop/>
4) 吉田裕之, 中山裕子: ドメイン指向パターン, 情報処理, Vol.40, No.12, pp.1186-1191 (Dec. 1999).
5) Fowler, M.: Analysis Patterns: Reusable Object Models, Addison-Wesley, Reading, Massachusetts (1997). (堀内一, 児玉公信, 友野晶夫 (訳): アナリシスパターン: 再利用可能なオブジェクトモデル, 星運社, 東京(1998)).
6) Beck, K.: Smalltalk Best Practice Patterns, Prentice Hall, Upper Saddle River, New Jersey (1997).
7) Coplien, J.: A Generative Development - Process Pattern Language, Coplien, J. and Schmidt, D. (eds.): Pattern Languages of Program Design, pp.183-257, Addison-Wesley, Reading, Massachusetts (1995). <http://www.bell-labs.com/people/cope/Patterns/Process/>
8) Brown, W.J., Malveau, R.C., McCormick III, H. W. and Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures and Project in Crisis, John Wiley & Sons, New York (1998). (岩谷 浩 (訳): アンチパターン: ソフトウェア危篤患者の救出, ソフトバンク, 東京 (1999)).
9) Patterns Home Page: <http://hillside.net/patterns>
10) Johnson, R. E., 中村宏明, 中山裕子, 吉田和樹: パターンとフレームワーク, 共立出版, 東京 (1999).

（平成11年11月30日受付）

次回「デザインパターンの適用」では，実例を基にデザインパターンの使い方を具体的に解説する．設計で遭遇する問題に対して適切なパターンを選ぶにはどうすればよいか，いくつかのよく知られたデザインパターンを使って説明する．