ソフトウェア工学　137－5
（２００２．　５．　１０）

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

鹿糠 秀行‡　　三部 良太†　　矢島 敬士‡,††

‡東京工業大学大学院総合理工学研究科 知能システム科学専攻
†日立製作所 システム開発研究所
††日立製作所 情報システム事業部

ソフトウェア開発において，ソフトウェアパターンを活用することは有用である．しかし，ソフトウェアパターンを活用するためには様々なスキルが必要であり，ソフトウェアパターンの適用時やその後の拡張時にソフトウェアパターンの意図から逸脱しがちである．そこで，ソフトウェアパターンの適用において，予め起こりうる逸脱とそれに対する解決策や防止策を整理し，逸脱パターンとして明文化することを提案する．逸脱パターンを蓄積し活用することによって，ソフトウェアパターンの使いやすさを向上させ，スキルの不足に起因する問題の解決と防止に有効であることを議論する．GoF のデザインパターンを題材に逸脱パターンの具体例を示す．

Proposal of Deviation Patterns for Applying Software Patterns

Hideyuki Kanuka‡　　**Ryota Mibe†**　　**Hiroshi Yajima‡,††**

‡Department of Computational Intelligence and Systems Science,
Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology.
†Systems Development Laboratory, Hitachi, Ltd.
††Information Technology Division, Hitachi, Ltd.

In software development, it is helpful to apply software patterns. However, when software patterns are applied, the pattern user who doesn't have the skill to apply them fully tends to make the intention of the software patterns deviate. We propose "Deviation Patterns" which provides solutions toward possible deviation cases applying software patterns. We discuss effectiveness of deviation patterns leading to proper application of software patterns with the example of deviation patterns for GoF design patterns.

1　はじめに

ソフトウェア開発の様々な局面において，繰り返し現れる課題とそれに対してよく使われる解決策を明文化したものがソフトウェアパターンである．ソフトウェアパターンの蓄積と適用によって，ソフトウェア開発における有用なノウハウを共有し再利用することで，ソフトウェアの生産性や品質を向上させることが期待できる．しかし，概して蓄積と適用は容易ではなく，いかに蓄積をして適用するかが問題である．蓄積に関しては，これまでに様々なソフトウェアパターンがカタログ化されているので，本稿ではソフトウェアパターンの適用に関する問題を中心に議論する．

ソフトウェアパターンの適用とは，これまで蓄積されてきたソフトウェアパターンの中から直面する課題に対して適するソフトウェアパターンを探索し，選び出したソフトウェアパターンの解決策を適用することによって課題を解決することである．

ところが，ソフトウェアパターンを活用するためには様々なスキルが必要であり，パターン適用時やその後の変更時や拡張時に，ソフトウェアパターンの意図から逸脱しがちである．これは，ソフトウェアの生産性や品質を低下させる原因になる．

本稿では，ソフトウェアパターンの適用において，予め起こりうる逸脱とその解決策や防止策を整理し，**逸脱パターン (Deviation Paterns)** として明文化す

ることを提案する．逸脱パターンの蓄積と活用によって，ソフトウェアパターンの使いやすさを向上させ，スキルの不足に起因する問題の解決や防止に有効であることを議論する．

具体例として，GoFの各デザインパターンに対する逸脱パターンを**DP逸脱パターン(Deviation Patterns from Design Patterns)**として明文化し，その活用法と効果について述べる．

2 ソフトウェアパターンについて

ソフトウェアパターンとは，ソフトウェア開発の各局面において繰り返し現れる課題とそれに対してよく使われる解決策を明文化したものである．例えば，分析段階のアナリシスパターン，システム全体のアーキテクチャを決定する段階のアーキテクチャパターン，設計段階のデザインパターン，コーディング段階のイディオム，他にもこれまでに多くのソフトウェアパターンが提案されカタログ化されている．

これらカタログ化されているソフトウェアパターンの多くは，過去に実績のあるノウハウを抽出し蓄積されてきたもので，適切に適用することでソフトウェアの生産性や品質を向上させることが期待できる．

例えばGoFのデザインパターン[1]は，オブジェクト指向設計において頻出する２３個の設計指針を明文化したものとして有名である．デザインパターンは，継承や集約などのオブジェクト指向特有の技術を使いこなし，再利用性や柔軟性に優れた設計の実現を支援するものとして注目されている．

2.1 ソフトウェアパターンの適用プロセス

[2]では，いずれのソフトウェアパターンも，課題，コンテキスト（課題の背景と前後関係），フォース（課題解消における制約），そして解決策の組み合わせからなるとしている．また各々の関係は，ソフトウェアパターンのメタモデルとして，図１のように示すことができるとしている．

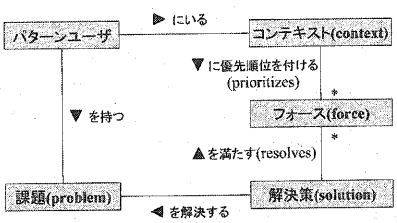

図 1: ソフトウェアパターンのメタモデル

図１のモデルを用いて，ソフトウェアパターンの適用プロセスとその難しさについて述べる．

まず，自分が直面している課題とそのコンテキストを把握し，それとソフトウェアパターンのコンテキストを照合し，いくつか該当しそうなソフトウェアパターンを見つけ出す．それから，ソフトウェアパターンのフォースを比較し，最適だと考えられるソフトウェアパターンを選び出し．最後に選び出したソフトウェアパターンの解決策を，対象とするプロダクトに対して適用することで，課題を解決させる．

したがって，パターンの対象とする領域の種類や表現力によるが，概してソフトウェアパターンの適用には，様々なスキルを必要とし難しいと考える．

2.2 ソフトウェアパターンの不適切な適用

ソフトウェアパターンを効果的に機能させ，ソフトウェアの生産性や品質を真に向上させるためには，直面する課題に対して適切にソフトウェアパターンを適用する必要がある．しかし，前節で述べたようにソフトウェアパターンの適用には，様々なスキルを必要とし難しいために，ソフトウェアパターンの不適切な適用を招きやすい．これによって，課題が解決されないだけでなく，逆にソフトウェアの生産性や品質を低下させるなど，ソフトウェア開発を阻害する新たな課題を生じる原因となってしまう．

3 逸脱パターン

3.1 ソフトウェアパターンの意図からの逸脱

ソフトウェアパターンの不適切な適用とは，ソフトウェアパターンが意図通りに使われないことである．言い換えれば，ソフトウェアパターンの意図からの逸脱である．

逸脱の原因を生じるのは，以下３つの段階である．

1. **パターン選択時**
   直面する課題に対して，適するソフトウェアパターンを探索し選択する時．
2. **解決策の適用時**
   ソフトウェアパターンの解決策を，プロダクトに対して適用する時．
3. **プロダクト変更時・拡張時**
   ソフトウェアパターンを適用したプロダクトを，変更や拡張する時．

各段階において，次のような逸脱の原因を生じると考えられる．１において，課題に対して不適切なソフトウェアパターンを選ぶ．２において，選び出されたソフトウェアパターンを，直面する課題に対して解決策を不適切に適用する．３において，ソフトウェアパターンを適用したプロダクトを，その後の過程でソフ

トウェアパターンが意図しないような変更や拡張をしてしまう．

逸脱とその原因の全てを，レビューやテストの際に確認し是正することができれば問題はない．しかし，逸脱を確認するためには，ドキュメントやプロダクトから判断しなければならず，特に第三者が逸脱を確認することは，その多くの場合難しい．

そこで [3][4][5] は，パターンが正しく適用されているかをツール支援によって判断することを提案している．しかし，これらのツールを用いて判断できるパターンは限られている．

3.2 逸脱パターンの定義

我々は，各ソフトウェアパターンに対して，起こりうるソフトウェアパターンの意図からの逸脱を整理するために，各逸脱に対して，それを是正する解決策や防止策を明文化するアンチパターン [6] として**逸脱パターン**を提案する．

アンチパターンとは，頻出する悪い問題におけるコンテキストとそこから生じる結果，それに対する解決策や防止策を明文化したものである．

3.3 逸脱パターンの記述フォーマット

アンチパターンの記述フォーマットを基準にして，定義した逸脱パターンの記述フォーマットを表 1 に示す．

表 1: 逸脱パターンのフォーマット

逸脱パターン名	パターンの名称
対象ソフトウェアパターン	対象とするソフトウェアパターン名
逸脱タイプ	逸脱のタイプ
一般形式	一般的に記述した逸脱パターンの特徴 (症状と結果を引き起こす原因となるもの)
症状と結果	逸脱の症状とそこから生じる問題
解決策	逸脱を是正する解決策
防止策	逸脱を生じさせないための防止策
逸脱例	逸脱の事例 (プログラミング言語で記述されたサンプルコードなど)
関連する逸脱パターン名	関連する逸脱パターン名
備考	追加事項など

表 1 のフォーマットにしたがって，各々のソフトウェアパターンに対して逸脱パターンを抽出し記述する．

3.4 逸脱パターンの活用法

3.1 節で述べた各段階に沿って，逸脱パターンの活用法を説明する (図 2)．

ソフトウェアパターン選択時に，自分の適用目的が適用候補のソフトウェアパターンの意図と異なっていないかの判断を，ソフトウェアパターンとその逸脱パターンを参照して確認する．時には，逸脱パターンの解決策が別のソフトウェアパターンへ誘導し，これを元にしてより最適なソフトウェアパターンを探す．

解決策適用時には，逸脱パターンを参照しながら，ソフトウェアパターンの解決策を適用する．逸脱パターンに適合した時には，逸脱パターンの解決策で修正する．

プロダクト変更時や拡張時には，適用されているソフトウェアパターンの逸脱パターンを参照しながら，変更と拡張を行う．逸脱パターンに適合したときには，逸脱パターンの解決策で修正しながらプロダクトを変更や拡張する．時には，解決策が別のソフトウェアパターンの適用を誘導し，これを元にしてより適するソフトウェアパターンへ変えるかを考える．

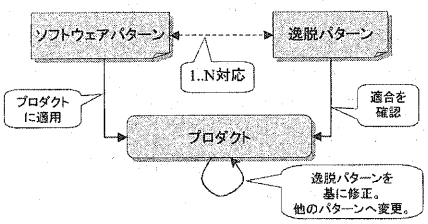

図 2: 逸脱パターンの活用法

3.5 逸脱パターンの効果

2.1 節で述べたように，パターンの対象とする領域とパターンの表現力に依存するが，一般的にソフトウェアパターンの活用は，様々なスキルを必要とするために難しい．すなわち，ソフトウェアパターンの使いやすさ（ユーザビリティ）は，概して良くないといえる．

逸脱パターンを活用することによる効果は，ソフトウェアパターンのユーザビリティを向上させることである．[7] で提案されているユーザインタフェースの 5 つのユーザビリティ特性 (ただし**主観的満足**の項目は除く) を，ソフトウェアパターンのユーザビリティ特性に言い換え，逸脱パターンの効果をまとめる．

1. **学習しやすさ**：逸脱パターンを習得することで，ソフトウェアパターンの活用において「こうしてはいけない」ということを簡潔に把握することを

可能にし，ソフトウェアパターンの習得を促進することができる．

2. **効率性**：逸脱パターンを参照することによって，ソフトウェアパターンを効率的に活用することができるようになる．

   例えば，パターン選択時やプロダクト変更時・拡張時に，逸脱パターンの解決策が別のパターンへ誘導することで，適応的にソフトウェアパターンを適用できる．また，解決策適用時に，逸脱パターンを参照することで，逸脱する原因を予め把握することができ，解決策を適用しやすくなる．

3. **記憶しやすさ**：概してソフトウェアパターンカタログから逸脱の原因となる内容を読み取るのは容易ではない．逸脱する原因と解決策や防止策を「逸脱パターン」として明文化することによって記憶しやすく，例えば逸脱の確認と逸脱の解決に使うことができる．

4. **エラー**：ソフトウェアパターンの不適切な適用によるエラーの発生率を低くし，エラーが起こっても逸脱パターンを適用することによってすぐに是正することができる．

   例えばパターン選択時に，適用候補のソフトウェアパターンを適用しても，課題を解決しない可能性を事前に知ることができる．また，解決策適用時やプロダクト変更時・拡張時に，予めしてはいけないことを確認し，逸脱が生じた場合にも解決策にしたがって是正することができる．さらに，逸脱パターンの防止策を適用することで，逸脱を防止することができる．

4 逸脱パターンの例

本節では，ソフトウェアパターンの一つであるデザインパターンを題材にして，逸脱パターンに関して具体的に議論する．

デザインパターンを適用することによってプロダクトの変更箇所や拡張箇所が明確になり，変更や拡張を容易にするといわれてきた．しかし，いくつかのデザインパターンでは，デザインパターンを適用しない場合よりも，むしろ適用した場合のほうが変更や拡張においてエラーを発生しやすく，また時間もかかるというが報告ある [8]．これは，デザインパターンのユーザビリティが良くないことを示した良い例である．

4.1 DP 逸脱パターンの定義

デザインパターンの適用における逸脱パターン（以降，**DP 逸脱パターン**と記する）を抽出し，蓄積し活用することによって，デザインパターンのユーザビリティを向上させることを目標とする．

DP 逸脱パターンを，表 1 で定義したフォーマットに従って記述する．逸脱タイプについては，逸脱の原因を分析し (表 2 右列)，「目的・協調関係・構造・実装」に分類することにした (表 2 左列)．この分類にした理由は，デザインパターンの特徴を示している項目が「目的・協調関係・構造」であり，さらにデザインパターンの解決策の適用には，しばしば「実装」が伴い重要だと考えたためである．

表 2: 逸脱タイプ

逸脱タイプ	逸脱の原因
目的	• 適用するデザインパターンがその意図とは違う目的で利用．
協調関係	• パターンを構成しているメソッドの不正な呼び出し． • パターンを構成しているクラスを不正に生成．
構造	• クラス関係（継承，委譲）が不正． • 不正なメソッド再定義． • 必要（不必要）メソッドの不定義（定義）． • 要素（クラス）個数が不正．
実装	• プログラミング言語上の問題．

4.2 DP 逸脱パターンの活用と効果

例として，GoF デザインパターンの一つである Template Method パターンを取り上げる．

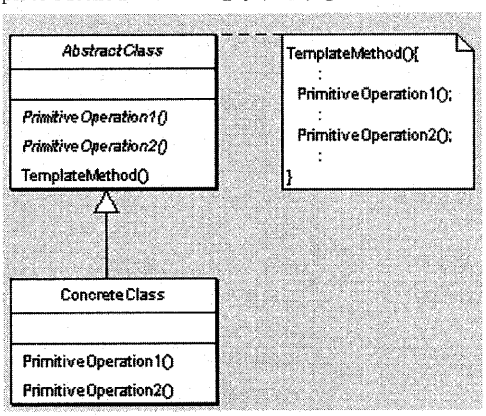

図 3: Template Method パターン

Template Method パターンは，抽象クラスである AbstractClass クラスと，それを継承する具象クラス

ConcreteClass クラスからなる (図 3).

AbstractClass クラスは，アルゴリズムのスケルトンである TemplateMethod メソッドとその内部で用いる PrimitiveOperation メソッドからなる．ConcreteClass クラスでは，PrimitiveOperation に対する任意の実装をする．

アルゴリズムの流れは決まっているが，その処理について詳細に決まっていないときに便利なので，フレームワークの構築にしばしば利用されるデザインパターンである．

フレームワークで変更可能な部分をホットスポット，固定部分をフローズンスポットと称する [12]．テンプレートメソッドでは，ホットスポットが TemplateMethod メソッドに，フローズンスポットが PrimitiveOperation メソッド（このメソッドをフックメソッドと称している）に対応する．

Template Method パターンの DP 逸脱パターンの例を表 3 と表 4 に示す．

表 3 や表 4 の逸脱パターンの「症状と結果」の項目を読むことによって，なぜ「一般形式」の項目に示されることをしてはいけないのかを納得し，デザインパターンの解決策の適用時，またはプロダクトの変更時や拡張時に，「一般形式」の項目で示される Template Method パターンの意図からの逸脱を防止できる．

また，例えば表 3 の解決策 2 を参照することによって，直面する課題を解決するためにより適するソフトウェアパターンへ誘導してくれることによって，パターン選択やプロダクト変更や拡張の一つの指針として検討することができる．

この解決策 2 では Template Method パターンから Strategy パターンへ変更することを促しているが，これは外部的な振る舞いを変えることなく，どのような手順で変更させるかというリファクタリング [14] の問題である．既存のプロダクト (プログラム) にデザインパターンを適用するリファクタリングはデザインパターンにもよるが複雑な場合が多く，その手法は [15][16][17] など多種多様に存在するので，逸脱パターンの解決策ではその手順の詳細までを言及しないことにする．

表 3: DP 逸脱パターンの例 1

逸脱パターン名	テンプレートメソッドの再定義
対象デザインパターン	Template Method パターン
逸脱タイプ	構造
一般形式	ConcreteClass クラスで TemplateMethod() を再定義する
症状と結果	Template Method パターンは，アルゴリズムの流れを固定し，その流れの中で使われる処理を変更することで使用される．したがって，フレームワークの構築に使われる．AbstractClass クラスの TemplateMethod() を再定義しようとする場合，それは TemplateMethod 内部のアルゴリズムを変えようとすることを意味しパターンの目的に逸脱する．
解決策	1. 頻繁に変更される状況であれば，TemplateMethod() のアルゴリズムを見直す．例えば，PrimitiveOperation() の内容を TemplateMethod() へ移動する． 2. TemplateMethod() のアルゴリズムごと変更したいのであれば，Strategy パターンへの変更を考える．
防止策	TemplateMethod() を ConcreteClass で再定義させない，つまりアルゴリズムの流れを変えないようにする．例えば Java では，TemplateMethod() を final 宣言することで再定義を禁止することができる．
逸脱例	省略
関連する逸脱パターン名	省略
備考	解決策 1 で提案したメソッドの再構成は，特に実装した後にそれを人手で行うことは特に難しい．そこで文献 [13] では，過去のアプリケーション開発時のメソッド変更履歴に基づいて，メソッドを自動的に再構成し，フレームワークの成熟化を実現する方法を提案している．

表 4: DP 逸脱パターンの例 2

逸脱パターン名	フックメソッドの直接呼び出し
対象デザインパターン	Template Method パターン
逸脱タイプ	協調関係
一般形式	パターン外部から直接 PrimitiveOperation() を呼び出す
症状と結果	Template Method パターンは，TemplateMethod() を介して PrimitiveOperation() を呼び出すことを想定している．例えば Client という外部のクラスがあって，そのクラスから直接 PrimitiveOperation() を呼び出していた場合に，ConcreteClass 及び PrimiveOperation() の内容が，Client が知らないうちに変わる可能性がある．そのために Client に不具合が生じる可能性がある．
解決策	PrimitiveOperation() を単独で呼び出す可能性があらかじめ想定できるならば，AbstactClass に PrimtiveOperation() を単独で呼び出す TemplateMethod() を定義する．利用者は，これを介してのみ PrimitiveOperation() を呼び出すようにする．
防止策	PrimitiveOperation() が TemplateMethod() からのみ呼び出せるようにする．例えば C++では，TemplateMethod() が呼び出す PrimitiveOperation() を保護的メンバとして宣言することで実現できる．
逸脱例	省略
関連する逸脱パターン名	省略
備考	省略

5 関連研究

ソフトウェアパターンの適用を支援するために，今までにいくつかのツールが開発されてきた．その多くはデザインパターンを対象としている．

- デザインパターンの選択を支援するために，ハイパーテキストでデザインパターンを記述し，各パターンの目的やその他をわかりやすく提示することで，選択を支援するツール [10][11].
- 解決策の適用を支援するために，デザインパターンから自動的にコードを生成するツール [4][9][10].

しかし，これらのツールを使用しても，最終的なパターンの選択，解決策の詳細な適用，そして適切な適用であるかの確認は人が行わなければならず，デザインパターンを活用するスキルがなければ，使わない場合と同様の逸脱を生じてしまう．

もちろん，これらの支援ツールは重要であり，逸脱パターンと相補的に活用することによって，より効率よくソフトウェアパターンを適用することができると考える．

6 むすび

ソフトウェアパターンを活用するためには，様々なスキルを必要とする．したがって，ソフトウェアパターンの不適切な適用や，その後に不適切な変更や拡張をしがちである．言い換えれば，ソフトウェアパターンの意図からの逸脱であり，ソフトウェアの生産性や品質を低下させる原因となってしまう．

そこで本稿では，ソフトウェアパターンの意図から，予め起こりうる逸脱とその解決策や防止策をまとめ，逸脱パターンとして明文化することを提案した．逸脱パターンを活用することによって，ソフトウェアパターンの使いやすさ (ユーザビリティ) を向上させ，スキルの不足に起因する問題の解決に有効であることを議論した．

具体例として，GoF のデザインパターンに対する逸脱パターンを，DP 逸脱パターンとして明文化した．デザインパターンは，拡張性や変更性に優れたオブジェクト指向設計を容易に実現するものとして注目されている．しかし，初心者には活用することが難しいとされ敬遠されている．DP 逸脱パターンを抽出・蓄積・共有・活用することによって，今までとは異なった視点で，デザインパターンの使いやすさを向上させることができると考える．そのためには，まず DP 逸脱パターンカタログを完成させることが課題である．そして，実験を通じて DP 逸脱パターンの効果を考察する必要がある．

また今後，他のソフトウェアパターンの適用における逸脱パターンも考える予定である．

参考文献

[1] E.Gamma, R.Helm ,R.Johnson, and J.Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, (1995).

[2] G.Maeszaros and J.Doble, "A Pattern Language for Pattern Writing", Pattern Languages of Program Design 3, pp529-574, Addison-Wesley, (1997).

[3] Mohlalefi Sefika, Aamod Sane, Roy H. Campbell, "Monitoring Compliance of a Software System With Its High-Level", Proceedings of the 18th International Conference on Software Engineering, (1996).

[4] 山本純一, 松本一教, "CASE ツールによるデザインパターン適用支援", 情報処理学会ソフトウェア工学研究会, No111-6, (1996).

[5] 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, No121-20, (1998).

[6] William J. Brown, Raphael C. Malveau, William H. Brown, Hays W., III McCormick, and Thomas J. Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis", WILLEY, (1998).

[7] Jakob Nielsen, "Usability Engineering", Morgan Kaufmann Publishers, (1994).

[8] L. Prechelt, B. Unger, W.F. Tichy, P. Brossler, L.G. Votta, "A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions", IEEE Transactions on Software Engineering, Vol.27, No12, pp.1134-1144, (2001.12).

[9] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, P. S. Yu, "Automatic code generation from design patterns", IBM Systems Journal, Vol.35, No.2, (1996).

[10] 大月美佳, 瀬川純一, 吉田紀彦, 牧之内顕文, "デザインパターンの SGML に基づく構造化文書化とその閲覧", 情報処理学会論文誌, Vol.39, No.3, (1998.3).

[11] 原田実, 長田英嗣, "設計図の融合機能を持つデザインパターン適用支援ツール OOPAS", 情報処理学会オブジェクト指向 2000 シンポジウム論文集, pp157-164, (2000.8).

[12] W.Pree, "Design Patterns for Object-Oriented Software Development", ACM Press, (1995).

[13] 丸山勝久, 島健一, "重み付き依存グラフを用いたメソッドの再構成", 情報処理学会論文誌, Vol.41, No.6, (2000.6).

[14] Martin Fowler, "Refactoring: Improving the Design of Existing Code", Addison-Wesley, (1999).

[15] Mel Ó Cinnéide, "Automated Application of Design Patterns: a Refactoring Approach", PhD thesis, University of Dublin, Trinity College, (2001).

[16] 加茂昌彦, 村木太一, 佐伯元司, "リファクタリングプロセスにおけるデザインパターン適用支援", 情報処理学会第 62 回全国大会, 2H-05, (2001.3).

[17] 青柳哲, 伊藤友昭, 中井戸健至, 木村耕, "粗粒度なパターン指向リファクタリングの考案", 情報処理学会第 64 回全国大会講演論文集 (分冊 1), pp119-120, (2002.3).