社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－SE－151（9）
2006／3／23

評価対象としてのデザインパターンの使用に関する考察

下滝 亜里†

デザインパターンを評価・比較のための例題やケーススタディとして用いて，色々な技術が提案・記述される傾向がある．本稿では，この傾向を具体的な例を挙げることにより明示的にし，考察する．結果として，技術を評価するためのツールとしてデザインパターンは効果的であるが，うまく活用されていないこと指摘し，デザインパターンをベンチマークとして活用すること提案する．

Need for a Design Pattern based Benchmark

ASATO SHIMOTAKI†

Using design patterns for evaluating or motivating a technology is now common practice and prevalent, but not well understood. In this paper, based on an analysis of literature, we argue that we should seek to find a more effective way of using design patterns for evaluating technologies. Our answer is to construct a benchmark based on design patterns and we expect that the benchmark will facilitate comparison and be used for more rigorously validating the technologies.

1. はじめに

プログラミング言語，ライブラリ，リファクタリング，テストに関連するツールなど色々な技術が，評価・比較の対象（例題やケーススタディ）としてデザインパターン[3]を用いて，提案・記述される傾向が見られる．評価や比較は，提案技術の有効性を示す重要な項目であるため，この傾向の分析と理解は重要である．

本稿の目的は以下である．(1) 傾向を具体的な例を挙げることにより，明示的にすること．(2) 傾向を分析・考察し，現状を把握・理解すること．(3) 評価対象としてのデザインパターンという「ツール」の使い方はアドホックであり，うまく活用されていないことを指摘すること．(4) ベンチマークとしてのデザインパターンの活用を提案すること．

本研究の長期的な目的は，デザインパターンを基にしたベンチマークを構築することであるが，本稿では，このようなベンチマークの必要性の動機の部分に重点を置く．ベンチマーキングへの早期のコミットメントは失敗を招くため，ベンチマーキングを実施する準備ができているかどうか確かめることが重要であるからである[17]．Simはベンチマーク構築にとりかかる前の必要条件として 3 つ（分野における最低限の成熟，比較の傾向があること，コミュニティにおける協力の姿勢があること）を挙げているが[17]，本稿では実際に例を挙げることにより，ベンチマーク構築の準備が少なくともできつつあること示す．

2 節では，具体的な例を挙げながら，評価対象としてデザインパターンがどのようにして用いられる傾向があるのか述べる．3 節ではこの傾向について分析・考察する．4 節では，議論を行うとともにデザインパターンを基にしたベンチマークの構築の提案とその目的について簡単に述べる．5 節はまとめである．

2. 評価対象としてのデザインパターン

提案技術の評価・比較を行う対象としてデザインパターン（特に GoF のデザインパターン）を用いた研究が多く報告されている．本節ではそのような報告の例を述べる．3 節では，この調査を基に考察を行う．

2.1. 調査内容

紙面の都合上，デザインパターンを評価対象として用いているすべての報告を紹介することはできない．しかし，どのような傾向があるのかを知るには十分である．調査するにあたって，次の項目に注目した．

- 提案している技術の種類
- 用いられたデザインパターン
- 比較対象（もしあれば）

なお，どのような評価基準を用いて技術が評価・比較されているのかについて調査することは重要であるが，今回の調査では取り扱ってない．これは，用いられたデザインパターンやそのサンプルコードを特定することに比べると，そのような基準を特定するのが簡単でないためである．

また，各技術は，デザインパターン以外を評価対象（例題やケーススタディ）として用いることもあるが，以下では評価対象として用いられたパターンだけに焦点を当てる．

2.2. 調査結果

表 3 は，調査結果のまとめである．以下では，読みやすくするために提案技術の種類に従って各技術を述べる．

†大阪産業大学大学院
Osaka Sangyo University

2.2.1. プログラミング言語/モデル/コンセプト

AspectJ (Hannemann and Kiczales, 2002): Hannemannと Kiczales（以下，H&Kと呼ぶ）はGoFの 23 種のデザインパターンを，JavaとAspectJ[29]により実装することを試みている[6]. AspectJは，アスペクト指向プログラミング[8]のためのJavaを拡張である.

Classpects(Rajan and Sullivan, 2005): Rajan と Sullivanは，H&Kと同様に，GoFの 23 種のデザインパターンを用いて，彼らの提案しているClasspectsモデルの評価を行っている[15]. 具体的には，H&Kのサンプルコードを用いることにより，ClasspectsとAspectJの比較を行っている.

MixJuice（田中と一杉，2003）: 田中と一杉は，MixJuice言語によるGoFデザインパターンの改善を報告している[26][43]. MixJuiceは，Javaを改良した言語であり，「差分ベースモジュール」と呼ばれるモジュール機構に特徴がある.

ObjectTeams (Herrmann, 2003): Herrmann は，ObjectTeams (OT) モデルを提案しており，ObjectTeams/Java[33]はこのモデルのJavaにおける実現である. OTは，役割やコラボレーションといったコンセプトを明示的にサポートしている点に特徴がある. [7]では，ObectTeamsの適用例としてデザインパターン（Observer, Decorator, Factories）を挙げている. OTにより，いくつかのパターンは不要になるか，再利用可能な形で実装可能となる.

Keris (Zenger, 2002): Zengerは，Keris言語を提案している[23]. Kerisはソフトウェア進化を明示的にサポートする言語である. Observerを例として用いている.

Josh (Chiba and Nakagawa, 2004): ChibaとNakagawaは，AspectJに似た言語であるJoshを提案している[2]. Joshでは，AspectJと違ってユーザがpointcut指定子を新たに定義できる点に特徴がある. 動機となる例の一つとしてVisitorパターンの実装を挙げている. 比較対象としてAspectJにおけるVisitorパターンの実装を用いている.

Sally (Hanenberg and Unland, 2003): Hanenberg と Unlandは，Singleton，Decorator，Visitorを用いてAspectJやHyper/Jの欠点を動機として，Sally言語を提案している[5].

LAVA (Kniesel, 2000): Knieselは，DARWINモデルを提案している[10]. LAVAはJavaを拡張することによるDARWINの実装である. LAVAは，静的や動的なdelegationをサポートする. Knieseは，LAVAの具体的な適用例としてDecoratorパターンとStrategyパターンの実装を挙げている.

LogicAJ (Kniesel et al., 2004): Knieselらは，Decoratorパターン適用後に発生するある種の機能要件は，従来のAOP言語ではうまく取り扱えないことを動機にLogicAJという言語の必要性を述べている[9]. LogicAJはAspectJの拡張であり，genericなアスペクトの記述をサポートする.

Caesar (Mezini and Ostermann, 2003): MeziniとOstermannは，Caesarモデルを提案している[13]. CaesarJは，Javaをベースとしたアスペクト指向言語である. [13]では，H&Kの再利用可能なObserverの実装を例として挙げ，Caesarとの比較を行っている.

EpsilonJ (Tamai et al., 2005): Tamaiらは，役割をベースとしたモデルであるEpsilonを提案している[20]. EpsilonJは，このモデルのJavaにおける実現である. [20]ではEpsilonJ とCaesarをObserverパターン実装の視点から比較している.

アスペクトのパラメータ化（Alvarez, 2004）: Alvarezは「アスペクトのパラメータ化」を提案しており，AbstractFactoryを例として用いて有効性を示している[1].

アスペクトのグループ化（下滝, 2005）: [24]では，お互いに排他的なアスペクトを表す「アスペクトのグループ化」のコンセプトを提案しており，AspectJのライブラリとして実現している. 適用例としてStrategyパターンとStateパターンの実装の改善を述べている. 改善の比較対象としてH&Kの実装を用いている.

2.2.2. ライブラリ/フレームワーク/システム

JAML（Lopes and Ngo, 2004）: Lopes とNgoは，JAML（Java Aspect Markup Language）を提案している[11]. JAMLは， XMLをベースとしたアスペクト指向言語である. JAMLでは，ドメインに特化した横断的関心事をプラグインとして定義することが可能であり，[11]ではプラグインの例としてSingletonとObserverの実装を挙げている. また，比較対象としてH&Kのサンプルコードを用いている.

Reflex (Tanter et al., 2003): Reflex [32]は，部分的動作リフレクション（Partial Behavioral Reflection）をサポートするJavaライブラリである. Tanterらは，Reflexによる Observerパターンの実装例を示している[21]. また，その実装をH&Kの実装と比べている.

Spring (Priolo, 2005): Priolは，Springフレームワーク[30]によるObserverパターンの実装例を示している[28]. Springは， IoC(Inversion of Control)コンテナの代表的な一つである.

dynaop (Lee): dynaopはプロキシをベースとしたAOPフレームワークである[31]. dynaopのマニュアルでは，Observerパターンの実装を例として用い，従来のオブジェクト指向における実装方法とdynaopを用いることによる実装方法を比較している.

2.2.3. その他

定量的な評価(Garcia et al., 2005): Garciaらは，定量的な視点から（コードの行数やコンサーンの分離に関連するメトリクスなど），H&KらのAspectJによるデザインパターン実装の再評価を行っている[28].

AO Refactoring (Monteiro, 2005): Monteiroらは，AspectJを用いたリファクタリングのケーススタディの対象として，H&Kのサンプルコードを用いている[12].

Aspectra (Xie and Zhao, 2006): XieとZhaoは，Aspectraを提案している[22]. Aspectraは，アスペクトの振る舞いをテストするための入力を自動的に生成するフレームワークである. Aspectraの有効性を評価する対象としてAspectJで書かれた 12 のプログラム (彼らは「ベンチマーク」と呼んでいる) を用いている. その中の一つにH&KのStateパターンの実装が含まれている.

AJTest (下滝, 2006): AJTest[34]はAspectJ5 におけるアスペクトのためのテスティングフレームワークであり，現在著者によって開発中である. AJTestの設計と実装は，H&Kのサンプルコードを基に行っている.

3. 考察

2 節での調査を基に，この節では考察を述べる.

3.1. 評価対象となったパターン

表 1 に，各パターンが評価対象となった回数を示す. GoFの他にもデザインパターンは提案されているが，今回調査した範囲ではGoFのデザインパターンが評価対象として使われていた. また，GoFの中でも一部のデザインパターンに偏って使われる傾向がある. 中でもObserverパターンが最も用いられている.

表 1 評価対象となった回数

パターン	回数	参照
Observer	8	[20] [11] [7] [23] [21] [13] [20][31]
Decorator	5	[5][1] [7] [10] [9]
Visitor	2	[5] [2]
Strategy	2	[24] [10]
State	2	[24][22]
AbstractFactory	1	[1]
Singleton	1	[5]
Factory	1	[7]
GoF すべて	5	[28]**エラー! 参照元が見つかりません。** [15][34][26]

3.2. パターンの実装は技術依存

デザインパターンの実装は用いる言語に依存するだけでなく，用いる技術に影響を受ける. たとえば Lopes と Ngo の JAML は，従来の意味での言語拡張は行っていない. また，Reflex，Spring，Dynaop などはライブラリやフレームワークであるがパターンの実装方法に影響を与える.

デザインパターンの実装が言語だけなく，たとえばアスペクトのグループ化などのライブラリに影響を受けることは，比較の視点から重要となる. たとえば AspectJ を使っているなら，従来の Strategy，H&K の Strategy，そしてアスペクトのグループ化による Strategy の実装方法が設計上の選択肢として現れる.

3.3. サンプルコードの利用性

第三者が再評価を行えるようにしたり，容易に比較ができるようにするために，評価・比較に用いられたサンプルコードを入手しやすくすることは重要である. この節では，サンプルコードの入手容易性について行った調査について述べる. 表 2 は，調査結果のまとめである. 調査するにあたって，2 つの項目に注目した.

- **サンプルコードの配布:** 論文中でデザインパターン実装の例題として用いたコードが配布されているかどうか.
- **比較に用いたサンプルコードの配布:** 論文中で比較対象として用いたサンプルコードが配布されているかどうか.

表 2 サンプルコードの利用性

	コードの配布	比較のコードの配布
AspectJ (H&K)	○	○
Caesar	○	なし(H&K)
JAML	○	なし(H&K)
Spring	○	×
LogicAJ	○	×
dynaop	○	×
MixJuice	△	△(GoF)
アスペクトのグループ化	△	なし(H&K)
ObjectTeams	△	×
Sally	△	×
Keris	△	×
Reflex	×	なし(H&K)
Classpects	×	なし(H&K)
EpsilonJ	×	なし(Caesar)
Josh	×	×
LAVA	×	×
アスペクトのパラメータ化	×	×

たとえば，H&Kは，JavaとAspectJのデザインパターン実装の比較を行っているが，その比較に用いたサンプルコード[36]は容易に入手可能である. この場合，「コードの配布」に対応するのは，AspectJのコードであり，「比較のコ

ードの配布」に対応するのはJavaのコードである.

同様に, 以下では 2 節で述べた各技術(「その他」に分類された技術は除く)について, サンプルコードが容易に取得できるか評価する.

ObjectTeams: ObjectTeams (OT) のサイト[33]では, OTの統合開発環境としてEclipseの拡張であるOTDT (Object Teams Development Tooling) が配布されている. OTのサンプルコードとしてObserverの実装例(再利用可能なObserverとその適用例)が含まれているが, [7]で述べているようなDecoratorやFactory関連の実装例は含まれていない. Observerの実装例は, [7]で明記されているバージョンとほとんど同じである. ただし, OTを用いないバージョン(Java)のObserverの実装例は配布されていない.

CaesarJ: CaesarJのサイト[35]では, [13]で用いられたObserver実装の例とほとんど同じバージョンが例として配布されている. 比較の直接的な対象となったAspectJのObserverの例は配布されていないが, H&Kのサンプルコード[36]を基にしているため, AspectJの例を入手するのは容易である. ただし, CaesarJのObserverの使用例は, H&KのObserverの使用例を少し改良したバージョンであり, 同一ではない. たとえば, CaesarJ では, Pointクラス(Subject), Lineクラス(Subject), Screenクラス(Observer), ColorObserver, PointObserverが主なモジュールであるが, H&Kでは, Pointクラス(Subject), Screenクラス(Subject, Observer), ColorObserver, CoordinateObserver, ScreenObserverが用いられている.

JAML: JAMLのサイト[37]では, ObserverとSingletonのサンプルコードが配布されている. これらのパターンの実装は, H&Kのサンプルコードを基にしている. ただし, JAMLでは, Pointクラス, Screenクラス, PointObserverImplクラス(H&KのColorObserverとCoordinateObserverに対応)の 3 つだけが実装例として配布されている.

Josh: Joshのサイト[38]では, [2]で動機となる例として挙げられたVisitorの実装例は配布されていない. また, 比較として用いられたAspectJの実装例も配布されていない.

LAVA: LAVAのサイト[47]では, [10]でLAVAの適用例として挙げられたStrategyとDecoratorの実装例は, サンプルコードとして配布されていない. また, LAVAを使用しない場合のそれらのパターンの実装例も配布されていない.

Reflex: [21]では, Reflexの例として(H&Kの例を基にすることによる)Observerの実装例が挙げられているが, Reflexのサイト[32]ではサンプルコードは配布されていない.

Keris: Kerisのサイト[39]では, Kerisの配布と共に再利用可能なObserverの実装は配布されているが, [23]で例として用いられたサンプルコードは配布されていない.

EpsilonJ: [20]ではObserver実装におけるCaesarとの比較例が述べられている. EpsilonJのサイト[40]では, Observerのサンプルコードは配布されていない.

アスペクトのグループ化: [24]では適用例としてStrategyとStateの実装の改善を述べている. 比較対象としてH&Kの実装を挙げている. [31]ではStrategyのサンプルコードは配布されているがStateのコードは配布されていない.

アスペクトのパラメータ化[1]: 具体的な実装はまだ配布されておらず, そのためサンプルコードも配布されていない.

dynaop: dynaopのマニュアル[31]では, 抽象的なコード例(MyClassImplクラスやobservedMethodメソッドなど)を用いて説明しているが, 実際に配布されているサンプルコードは具体的(BookTitleObserverなど)である. ただし, dynaopを用いない場合のJavaのサンプルコードは配布されていない.

LogicAJ: [9]ではDecoratorが用いられており, LogicAJのサイト[45]では, Decoratorだけでなく Visitorの実装例が配布されている.

Sally: [5]では, Singleton, Visitor, Decoratorを例として用いていたが, Sallyのサイト[44]では, Observer, Visitor, Singletonがサンプルコードとして配布されている. [5]ではAspectJやHyper/Jが比較対象として挙げられているが, これらのサンプルコードは配布されていない.

MixJuice: [43]では, 抽象的なコード(AbstractFactoryクラスcreateProductAメソッドなど)として, Mixjuiceによるパターンの改善例が載っている. 具体的なコードを示しているわけではないため, 従来のパターンの構造(改善前)が比較の対象になると考えられる.

Classpects/Eos: Eosのサイト[46]では, H&Kのようには, サンプルコードは公開・配布されていない.

Spring: [28]では, SpringによりObserverを実装する例がサンプルコードとして載っている. Springを使わない場合のコードは載っていない.

3.3.1. 考察

表 2 から分かるように, 17 件中 11 件がデザインパターン実装のサンプルコードを配布している. 17 件中 5 件がH&Kの実装と比較を行っているが, 5 件中 3 件はH&Kのサンプルコードと同一シナリオ(パターンの具体的な実装例)ではない. また, 比較のサンプルコードを配布しているのはH&Kだけである.

3.4. サンプルコードの再利用

H&Kがパターン実装の例として用いたサンプルコードが異なる研究者によって再利用される傾向がある. LopesとNgoは, JAMLのプラグイン例として, ObserverとSingletonのサンプルコードを用いている[11]. Tanterらは, ReflexによるObserverの実装例を示すために用いている[21]. Monteiroは, アスペクト指向におけるリファクタリング

のためのケーススタディの一つとしてH&Kのコードを用いている[12]. Xieらは, Aspectraの有効性を評価するための対象の一つとしてH&KのStateの実装を用いている[22]. 下滝は, アスペクトのグループ化のコンセプトの適用対象としてH&Kのコードを用いている[24]. また, 下滝はAJTest[34]の設計と実装を行うにあたって, H&Kのアスペクトを対象としている. Garciaらは, 定量的な視点から, H&Kの結果を再評価している[28].

[6]におけるH&Kの研究目的は, 新しい言語としてのAspectJが従来のデザインパターンの実装に与える影響を調査することであったため, この傾向は予期せぬものであったと考えられる. 再利用される理由としては (1) Javaという主流の言語をベースとしていること, (2) GoFの一部のパターンではなく 23 種のすべてのパターンを対象としていること, (3) 第三者が利用・入手しやすいようにサンプルコードが公開・配布されていることなどが考えられる. たとえば, アスペクト指向におけるリファクタリングのためにMonteiroがJavaとAspectJで書かれた例を探している時, 彼の探している条件に最も近いのがH&Kのサンプルコードであった[12].

なお, ある例題が異なる研究者によって何度も使用されることはデザインパターンに限ったことではない. たとえば, AOPのコミュニティでは図形エディタの例が用いられることが多い[8][2][11][14]. 他にもSullivanが例として挙げた統合システムがある[19][18][16][20].

3.5. デザインパターンを評価対象として用いる利点

デザインパターンを評価・比較の対象として使う利点はいくつか考えられる.

(1) GoFのデザインパターンは良く知られているため, 実装の詳細や問題の文脈などを詳しく説明する必要がない[5].

(2) 言語(モデル)の欠点や制限を具体的に明らかにする. たとえば, RajanらはClasspectsモデルの制限が明らかになったと報告している[15].

(3) デザインパターンが解決する課題は, 実践的に繰り返し発生すると実証されているため, たとえば従来のデザインパターンの実装を改善するという提案は, そのまま実践的であると考えられる. つまり, デザインパターンを評価対象として用いることは提案技術の有効性の「デモンストレーション」と「検証」の 2 つの特性を持っていると言える.

3.6. まとめ

- 一部のデザインパターンが評価対象として用いられる.
- デザインパターンの実装は言語に影響を受けるだけでなく, ライブラリやフレームワークにも影響を受ける.
- 論文中で用いられたサンプルコードが, 第三者が利用しやすいように配布されることは多くない. そのため, 再評価や比較を行うのが困難となる.
- しかし, 第三者が利用しやすいようにサンプルコードが配布されるなら, サンプルコードは再利用される.
- デザインパターンを評価・比較の対象として用いる利点がある.

4. 議論とアプローチ

3 節では, デザインパターンが評価対象として用いられる傾向について分析し考察した. この節では「デザインパターンを対象として技術を評価・比較する現状は, 最善でなくても良い傾向であり, この傾向は今後も続く」と仮定し, 「評価対象としてのデザインパターンの使用」という視点から議論を行う. まず, 現状における問題点や不十分な点は何であり, どのようにしてそれらの点を解決・改善するのかということについて簡単に述べる. 結果として, デザインパターンを基にしたベンチマーク構築の必要性を指摘する.

4.1. サンプルコード利用性の向上

3.1 節では, 一部のパターンが偏って評価対象として使用されていることを指摘した. 3.2 節では, パターンの実装は技術依存であるため, 実装の比較が重要となることを述べた. 3.3 節では, パターンの実装例としてのサンプルコードを利用しやすいように配布されていることは多くないことを指摘した. また, 評価の対象としているデザインパターンは同じでも, パターンインスタンス(適用のシナリオ)が異なる場合があることも述べた.

これらのことからは, 各技術の比較を行うことは重要であるが, 現状としては比較を行いやすい環境ではないと言える. したがって, 比較を行いやすいような環境を構築する必要がある.

4.2. パターンインスタンスの収集

3.4 節では, H&Kのサンプルコードが再利用される傾向があることを指摘した. ただし, H&Kがサンプルコードとして挙げているのは, 各パターンに対して一つの適用例(パターンインスタンス)だけであり, パターン実装のバリエーションやその他の適用例を広く取り扱っているとは言えない. たとえば, H&Kが用いたObserver実装のサンプルコード(適用のシナリオ)では, 彼らの再利用可能なObserverの実装(ObserverProtocol)は実際に再利用可能である. しかし, パターン適用のシナリオの微妙な違いが再利用性に影響を与えるため, Monteiroがリファクタリングの対象として用いたEckelのサンプルコードの例[27]では, ObserverProtocolを容易に再利用できなかった[12].

このことは, パターン適用の異なるシナリオにおいて各技術を評価する必要性があることを示している. たとえば, 再利用可能な Observer の実装に限ったことでいえば, CaesarJ と ObjectTeams でもそれぞれ ObserverProtocol を

提案しているため，同様に Eckel の例を用いてその再利用性を評価するのは興味深い．

4.3. アプローチ：DP-Bench

以上の二点を動機とし，本研究では，デザインパターンを基にしたベンチマーク（DP-Bench）の構築を検討している．DP-Bench[42]の目的は，デザインパターンを評価対象として用い，提案技術の再評価や比較を行いやすい環境を提供することである．

ベンチマークの構築には，時間がかかる[17]．まずはHannemannと Kiczalesの試み[36]を参考にする予定である．具体的には，比較のベース（Java）となるサンプルコードとして用いるためのパターンインスタンスの収集や作成，それからH&Kのサンプルコードを実装例として用いていない場合（たとえばObjectTeams）には，その実装を行い，公開していくことを予定している．

また，ベンチマークには比較のための共通の評価対象だけでなく，評価基準も必要である[17]．この点については今後の課題であるが，[25]ではDecoratorを具体的な例として挙げ，評価基準の必要性ついて簡単に述べた．

表 3 調査結果のまとめ

	パターン	技術	理由・動機	比較対象
Hannemann and Kiczales[6]	GoF	AspectJ	評価	Java
Rajan and Sullivan[15]	GoF	Classpects/Eos	評価	AspectJ
Garcia et al.[28]	GoF	AspectJ	評価	なし
田中，一杉[26][43]	GoF	MixJuice	評価	GoF
Alvarez[1]	AbstractFactory	アスペクトのパラメータ化	例	Java
Lopes and Ngo[11]	Singleton, Observer	JAML/プラグイン	例	AspectJ
Herrmann[7]	Observer, Decorator, Factories	ObjectTeams	適用例	Java
Hanenberg and Unland[5]	Singleton, Decorator, Visitor	Sally/Parametric Introduction	動機となる例	AspectJ, Hyper/J
Kniesel[10]	Decorator, Strategy	DARWIN /LAVA	適用例	Java
Kniesel et al.[9]	Decorator	LogicAJ	動機となる例	AspectJ
Zenger[23]	Observer	Keris	例	Java
Chiba and Nakagawa[2]	Visitor	Josh	動機となる例	AspectJ
Tanter et al.[21]	Observer	Reflex	例	AspectJ (H&K の例)
Mezini and Ostermann[13]	Observer	Caesar/ CaesarJ	問題説明	AspectJ
Tamai et al.[20]	Observer	Epsion/EpsionJ	比較	Caesar
Priolo[28]	Observer	Spring	比較	Java
Lee[31]	Observer	dynaop	例	Java
Monteiro[12]	GoF	AO Refactoring	ケーススタディ	なし
Xie and Zhao[22]	State	Aspectra	ベンチマーク	なし
下滝[24]	Strategy, State	AspectJ/アスペクトのグループ化	適用例	AspectJ (H&K の例)
下滝[34]	GoF	テスティングフレームワーク/AJTest	設計と実装	なし

5. まとめ

色々な技術が評価・比較の対象としてデザインパターンを使用する傾向がある．デザインパターンを評価対象として使用する利点は，デザインパターンが現実的なシステ

ムにおいて繰り返し発生するとして抽出された構造を表現していることに関係する．そのため，デザインパターンを評価対象として用いて得られた有効性は，現実的なシステムにおいても有効であることが期待できる．

しかしながら，評価対象としてのデザインパターンという「ツール」の使い方はアドホックであり，うまく活用されていない．本稿では，このツールを，ベンチマークの観点から活用できることを指摘した．

参考文献

[1] J. Alvarez. Parametric Aspects: A Proposal. RAM-SE'04 ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution. 2004.

[2] S. Chiba and K. Nakagawa. Josh: An Open AspectJ-like Language. AOSD'04.

[3] E. Gamma, et al. 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．ソフトバンクパブリッシング．

[4] A. F. Garcia, et al. Modularizing Design Patterns with Aspects: A Quantitative Study. AOSD'05.

[5] S. Hanenberg and R. Unland. Parametric Introductions. AOSD'03.

[6] J. Hannemann and G. Kiczales. Design Pattern Implementation in Java and AspectJ. OOPSLA' 02.

[7] S. Herrmann. Object Confinement in Object Teams -- Reconciling Encapsulation and Flexible Integration. Third GI Workshop on Aspect Oriented Software Development. 2003.

[8] G. Kiczales. J. Lamping, A.Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. ECOOP'97.

[9] G. Kniesel, T. Rho and S. Hanenberg. Evolvable Pattern Implementations Need Generic Aspects. RAM-SE'04 ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution. 2004.

[10] G. Kniesel. Dynamic Object-Based Inheritance with Subtyping. PhD Thesis, Computer Science Department III, University of Bonn. July 2000.

[11] C. V. Lopes and T. C. Ngo. The Aspect Markup Language and its Support of Aspect Plugins. ISR Technical Report UCI-ISR-04-8, October 2004.

[12] M.P. Monteiro. Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts. Ph.D. Thesis. 2005.

[13] M. Mezini and K. Ostermann. Conquering Aspects with Caesar. AOSD'03.

[14] K. Ostermann, M. Mezini, and C. Bockisch. Expressive Pointcuts for Increased Modularity. ECOOP'05.

[15] H. Rajan and K. Sullivan. Design Patterns: A Canonical Test of Unified Aspect Model. Technical Report 00000389, Department of Computer Science, Iowa State University, Oct 2005.

[16] K. Sakurai, H. Masuhara, N. Ubayashi, S. Matsuura, S. Komiya. Association Aspects. AOSD'04.

[17] S. E. Sim. A Theory of Benchmarking with Applications to Software Reverse Engineering. Ph.D. Thesis, Department of Computer Science, University of Toronto. 2003.

[18] K. Sullivan, L. Gu and Y. Cai. Non-Modularity in Aspect-Oriented Languages: Integration as a Crosscutting Concern for AspectJ. AOSD'02.

[19] K. J. Sullivan. Mediators: Easing the Design and Evolution of Integrated Systems. Ph.D. Dissertation, University of Washington Department of Computer Science and Engineering, Technical Report UW-CSE-94-08-01, August, 1994.

[20] T. Tamai, N. Ubayashi and R. Ichiyama. An Adaptive Object Model with Dynamic Role Binding. ICSE'05.

[21] É. Tanter, J. Noyé, D. Caromel, P. Cointe. Partial Behavioral Reflection: Spatial and Temporal Selection of Reification. OOPSLA'03.

[22] T. Xie and J. Zhao. A Framework and Tool Supports for Generating Test Inputs of AspectJ Programs. AOSD'06.

[23] M. Zenger. Evolving Software with Extensible Modules. International Workshop on Unanticipated Software Evolution. 2002.

[24] 下滝 亜里. AspectJ におけるアドバイスの動的なオン・オフ化. FOSE 2005.

[25] 下滝 亜里. ベンチマークとしてのデザインパターンの使用に向けて. ウィンターワークショップ 2006・イン・鴨川. 2006.

[26] 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:プログラミング, Vol.44 No.SIG 4(PRO 17), pp25--46, Mar. 2003.

[27] Bruce Eckel. Thinking in Patterns Revision 0.9 - May 20, 2003, http://www.pythoncriticalmass.com/downloads/TIPatterns-0.9.zip

[28] S. Priolo. Spring Loaded Observer Pattern. http://www.theserverside.com/articles/article.tss?l=SpringLoadedObserverPattern, 2005.

[29] AspectJ, http://www.eclipse.org/aspectj/

[30] Spring, http://www.springframework.org/

[31] dynaop, https://dynaop.dev.java.net/

[32] Reflex, http://reflex.dcc.uchile.cl/

[33] Object Teams, http://www.objectteams.org/

[34] AJTest, http://www.ncfreak.com/asato/software/ajtest.html
[35] CaesarJ, http://caesarj.org/
[36] Aspect-Oriented Design Pattern Implementations, http://www.cs.ubc.ca/~jan/AODPs/
[37] JAML, http://www.ics.uci.edu/~trungcn/jaml/
[38] Josh, http://www.csg.is.titech.ac.jp/%7Enakagawa/josh/
[39] Keris, http://zenger.org/keris/
[40] EpsilonJ, http://kumiki.c.u-tokyo.ac.jp/~ichiyama/cgi-bin/trac/trac.cgi/wiki/RoleModel4Java5
[41] Dynamic AspectJ Library , http://www.ncfreak.com/asato/software/dynaj.html
[42] DP-Bench, http://ncfreak.com/asato/dpbench/
[43] MixJuice によるデザインパターン改善カタログ, http://staff.aist.go.jp/y-ichisugi/ja/mj/design-pattern/index.html
[44] Sally, http://dawis.informatik.uni-essen.de/site/site/research/aosd/sally/
[45] LogicAJ, http://roots.iai.uni-bonn.de/research/logicaj/
[46] Eos, http://www.cs.virginia.edu/~eos/
[47] LAVA, http://javalab.cs.uni-bonn.de/research/darwin/project.html