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

2005－SE－149（11）
2005／7／29

ソフトウェア設計におけるベンチマーキングに向けての考察

下滝 亜里†

ソフトウェア設計に関する技術は，簡単な例題やケーススタディによって有効性が検証される傾向がある．このような例題は，提案手法がどのような問題を解決しようとしているのかを示すのに役に立つ．有効性をさらに検証するためには，広範囲かつ現実的な例題を用いることが望ましいが，そのような例題が適切に文章化されていることは稀である．本稿では，誰でも容易にアクセスでき，現実的であるコード例を表す，ソフトウェア設計のためのベンチマークの必要性を指摘する．

A First Step Towards Benchmarking in Software Design

ASATO SHIMOTAKI†

Validating a software design technology is not easy. The value of the technology is typically validated with toy examples and case studies. But there is often no realistic code for further validating the technology. This paper argues that we need benchmarks that are easy to access and reliable.

1. はじめに

ソフトウェア設計に関する技術は，簡単な例題やケーススタディによって有効性が検証される傾向がある．このような例題は，提案手法がどのような課題を解決しようとしているのかを示すのに役に立つ．より現実的な状況において提案手法の有効性を検証することが望ましいが，現実的である例題が適切に文章化されていることは稀である．そのため，提案手法の検証のために，実際にソフトウェアを開発するなどの比較的大きなプロジェクトが実施されるが，時間やコストがかかるという欠点がある．また，提案手法の比較を困難にするという欠点もある．この問題に対処するには，誰でも容易にアクセスでき，現実的であるコード例を表す，ソフトウェア設計のためのベンチマークがあることが望ましい．明確に定義されたベンチマークは，提案手法の有効性を検証するのに役に立つだけでなく，他の手法との比較を楽にすると期待できる．

ある分野で成功するベンチマークを構築するためには，個人としての試みだけでは不十分であり，コミュニティとして，ベンチマークの必要性の認識と同意が必要不可欠である[24]．本稿は，具体的かつ典型的な提案を例として挙げて，そのようなベンチマークの必要性を指摘することを目的としている．

以下，2 節では，提案する時の典型的な例を示す．3 節では，提案手法の評価方法自体の評価とベンチマークの必要性を議論する．5 節では関連研究を述べる．6 節では，まとめと今後の課題を述べる．

†大阪産業大学大学院
Osaka Sangyo University

2. 提案例

この節では，設計手法の提案や比較が典型的にどのように行われるのかを示す．そのために，具体的な提案例を用いる．この提案例を基に，3 節では，提案方法として何が問題となり，解決するには何が必要とされるのかを議論する．

提案例として，異なる 4 つのアスペクト指向[10]言語の比較を行う．比較に用いる言語は AspectJ(バージョン 1.2)[32]，abc (AspectBench Compiler)[1]，Caesar[19][34] AspectJ5[33]である．比較は，シンプルな例題を通して行う．また，数ステップに及ぶ進化の要求を考慮している．

2.1. アスペクトのオン・オフ化の進化シナリオ

以下のような進化のシナリオを基にする（図 1）．

(1) アスペクトの振る舞いをオン・オフできるようにする（Activatable Aspect 化）

(2) オン・オフ可能なアドバイスの追加（Activatable Advice の追加）

(3) ポイントカットのコード重複削除のリファクタリング（ActivationAspect の導入）

このシナリオは図 2 に示している単純なアスペクトから始まる．比較の基礎となる言語として AspectJ を用いる．以下の節では各進化のシナリオを詳しく述べる．

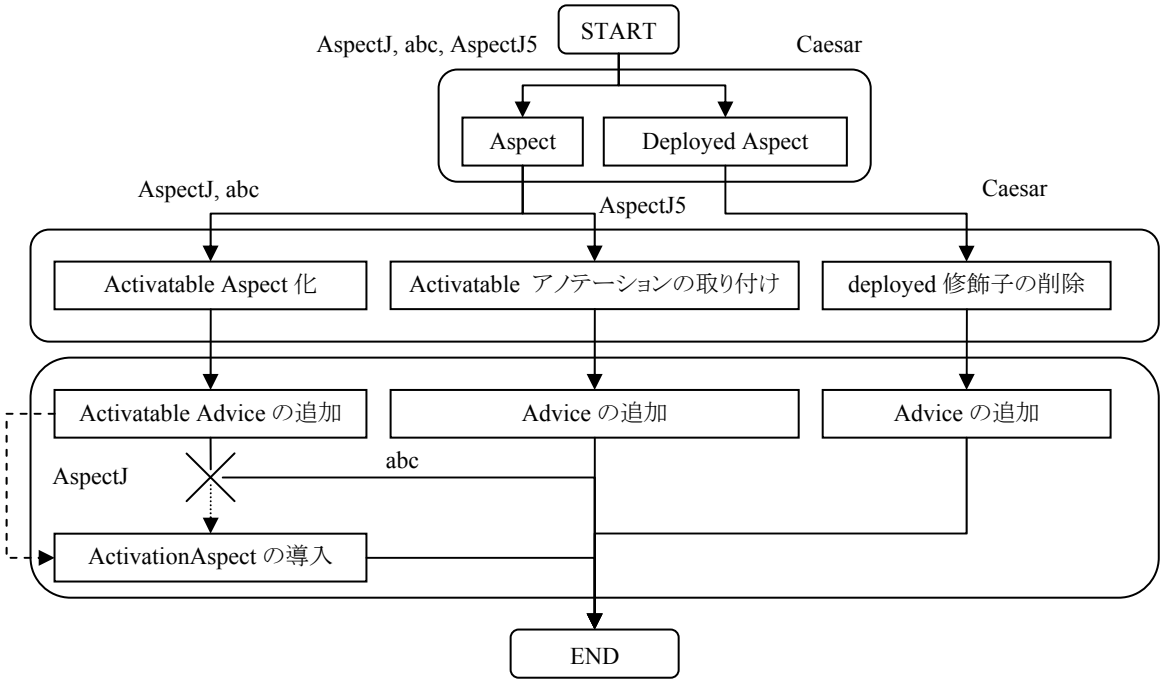

図 1 アスペクトのオン・オフ化の進化シナリオ

2.1.1. Activatable Aspect 化

状況: アスペクトがオンかオフかによって，アドバイスを実行するかどうかを実行時に制御できるようにしたい．AspectJ では，プログラマに対して，特定のアドバイスを任意に実行するかどうかの直接的な方法が提供されていないが，if pointcut を用いる(あるいはアドバイス内で if を使用する)ことにより，特定のアドバイスを実行するかどうかの制御を容易に実現できる．図 2 は，アドバイスがまだ常に実行される状況の時のコードを表している．図 3 には，その後、アドバイスの実行が，アスペクトがオンかオフに依存して決める要件に対してのコードの変化を示している．

```
public aspect PointLogging {
   before() : call( void Point.setX(int) ) { … }
}
```

図 2 Activatable Aspect 化：適用前

```
public aspect PointLogging {
   private static boolean isActive = true;
   before() : call( void Point.setX(int) ) && if(isActive) { … }
}
```

図 3 Activatable Aspect 化：適用後

2.2. Activatable Advice の追加

状況: 新たに定義するアドバイスは，アスペクトがオンかオフかによって実行するかどうかを決めたい． オン・オフが可能になったアスペクトがすでに存在する時に，次に新たに定義されるアドバイスも，アスペクトがオンかオフに応じてアドバイスを実行するかどうかを決めたい場合がある． 進化前のコードは図 3 であり，図 4 は新たにアドバイスを追加した後のコードである．

```
public aspect PointLogging {
   private static boolean isActive = true;
   pointcut isActive() : if(isActive);
   before() : call( void Point.setX(int) ) && isActive() { … }
   // 新たに追加されたアドバイス
   before() : call( void Point.setY(int) ) && isActive() { … }
}
```

図 4 Activatable Advice の追加:適用後

2.2.1. ActivationAspect の導入

状況: アドバイスをオン・オフにするために必要な if ポイントカットの重複をなくしたい. いくつかのアドバイスは, アスペクトがオンかオフかに依存した if ポイントカットにより, 実行するかどうかが決まる. 通常, それらの if ポイントカットは重複しているが AspectJ では容易にこの重複を削除することはできない. しかし, ActivationAspect ライブラリ[31]を用いることによりこれらの重複をなくすることができる. ActivationAspect ライブラリ適用前のコードを図 4 に, 適用後を図 5 に示す. Activatable インタフェースを実装したアスペクトは, 自動的にアスペクトをオンにするためのメソッド(activate())やオフにするためのメソッド(deactivate())が定義される.

```
public aspect PointLogging implements Activatable {
   before() : call( void Point.setX(int) ) { … }
   before() : call( void Point.setY(int) ) { … }
}
```

図 5 ActivataionAspect の導入:適用後

2.3. 比較

本節では, 進化の過程を表す上記の例を基にして, abc, Caesar, AspectJ5 の各言語がどのような異なる実装になるのかを示す. 最後に, 各言語の比較を行う.

2.3.1. abc

abc は AspectJ 1.2 の仕様をサポートしているが adviceexecution の取り扱いの違いのため[14], AspectJ のように ActivationAspect ライブラリを用いることによる if ポイントカットの重複の削除は難しい. したがって, 図 1 で示しているように, abc では ActivationAspect の導入のシナリオは存在せず, アスペクトのオン・オフ化の機能をうまくモジュール化することはできない.

2.3.2. Caesar

Caesar は, AspectJ に似ているがいくつかの異なる特徴を備えている. 上記のシナリオに最も関連する AspectJ との違いは, アスペクトのデプロイメントにある. AspectJ では, アスペクトの振る舞い(アドバイス)は静的であり, プログラムの実行時からオンになっている. 一方 Caesar では, AspectJ における静的なデプロイメントに加え, AspectJ にはない動的なデプロイメントが可能である. 図 6 は, アスペクトのオン・オフ化の要求を, Caesar がどのように取り扱うのかを示している. Caesar では, deployed 修飾子を用いることにより, AspectJ におけるアスペクトと同じ意味を表すことができる. つまり, プログラムの実行時から, アスペクトの機能はオンになっている. 一方, 実行時にアスペクトをオンやオフにしたい場合には, deploy 文を用いるか, 以下に示しているように DeploySupport のメソッドを呼び出すだけでよい. したがって, Caesar では, アスペクトのオン・オフ化を実現するために特別なことをする必要はない.

```
PointLogging l = new PointLogging();
DeploySuppor.deployLocal(l); // ログをオン
…
DeploySuppor.undeployLocal(l); // ログをオフ
```

```
public deployed cclass PointLogging {
   before() : call( void Point.setX(int) ) { .. }
}
                                  ↓
public cclass PointLogging {
   before() : call( void Point.setX(int) ) { … }
}
                                  ↓
public cclass PointLogging {
   before() : call( void Point.setX(int) ) { … }
   before() : call( void Point.setY(int) ) { … }
}
```

図 6 Caesar における進化の過程

2.3.3. AspectJ5

```
public aspect PointLogging {
   before() : call( void Point.setX(int) ) { … }
}
                                  ↓
@Activatable
public aspect PointLogging {
   before() : call( void Point.setX(int) ) { … }
}
                                  ↓
@Activatable
public aspect PointLogging {
   before() : call( void Point.setX(int) ) { … }
   before() : call( void Point.setY(int) ) { … }
}
```

図 7 AspectJ5 における進化の過程

AspectJ5 は, Java5 における代表的な新機能であるメタデータ(アノテーション)[34]とジェネリクスに対応した AsepctJ の最新バージョンである. この新機能を基にして, ActivationAspect の機能を書き直した(図 7).

実行時にアスペクトをオンやオフにしたい場合には，@Activatable を付加するだけでよい．AspectJ の時と同じように，アスペクトをオン・オフ化するためのメソッドが自動的に定義される．

2.4. 評価

表 1 に比較結果を示す．各言語の横の数字は順位を表している．まず，abc のみが，アスペクトのオン・オフ化の機能（横断的関心事の一つとして考えられる）をうまくモジュール化できなかった． AspectJ，Caesar，AsepctJ5 については，モジュール化の視点からは（ここではパフォーマンスは考慮しない），違いはほとんどないと考えられるが，Caesar がわずかに良い．これは Caesar におけるアスペクトは，特別なライブラリを必要することなく動的に扱えるためである．

表 1 アスペクトのオン・オフ化の比較

	モジュール性	簡潔さ
abc(4)	×	×
AspectJ(2)	○	△
AspectJ5(2)	○	△
Caesar(1)	○	○

2.5. さらなる進化シナリオの検討

もし，前節で述べた評価のみを考えるならば，Caesar の優勢という結論になるかもしれない．しかし，上記のシナリオに加えて新たな進化のシナリオを考えると，結論は異なる．この異なる結果は，ソフトウェアの進化をどのくらい考慮するかによって，いかに評価に影響を与えるのかを示している．

以下では，アスペクト単位でのオン・オフ化ではなく，アドバイス単位でのオン・オフ化を考える．具体的には，新たに追加するアドバイスは，アスペクトがオンかオフかに関わらず，常に実行されるアドバイスであると想定する．前と同様に，比較の基礎となる言語として AspectJ を用いる．

2.5.1. アドバイスのオン・オフ化の進化シナリオ

状況によっては，アスペクト単位でのオン・オフ化だけでなく，アドバイス単位でのオン・オフ化が有効であるかもしれない．図 8 には，アスペクトがオンかオフかに関わらず，常に実行されるアドバイスが追加された時のコードの変化を示している．

```
public aspect PointLogging {
   private static boolean isActive = false;
   pointcut isActive() : if(isActive);
   before() : call( void Point.setX(int) ) && if (isActive) { … }
   before() : call( void Point.setY(int) ) && if (isActive) { … }
   before() : call(void Point.setColor(String) ) { … }
}
```

図 8 アドバイスのオン・オフ化（AspectJ）

分かるように，AspectJ では，この要求をうまくモジュール化することは困難である．この要求を満たすには，Activatable インタフェースを取り外し，if ポイントカットの重複をなくすために行ったリファクタリング実行前のコードに戻す必要がある．

2.6. 比較

アドバイスのオン・オフ化の進化シナリオにおいて，同様の比較を行う．

2.6.1. abc

最終的に到達する abc の実装は，図 8 で示している AspectJ の実装と同じである．しかし，abc の方が AspectJ よりもより自然に進化できる．AspectJ の場合には，アドバイスのオン・オフ化機能を実現するためには Activatable インタフェースを外し，再び if ポイントカットをアドバイスに付け加える必要があった．一方，abc では，元々Activatable インタフェースは実装されていなかったので，単にアドバイスを追加するだけである．

2.6.2. Caesar

Caesar は，最も問題となる．理由の一つは，Caesar では，if ポイントカットがまだサポートされていないためである．そのため，アドバイスのオン・オフ化の機能を実現するためには，以下に示しているようにアドバイス内に if 文を直接埋め込む必要がある．結果としてアドバイスのオン・オフ化の機能は，複数のアドバイスに散らばっており，また，アドバイスの元々の機能とからまってしまっている．これは，横断的関心事をうまくモジュール化できないときの典型的な兆候である．

```
public deployed cclass PointLogging {
   private boolean isActive = false;
   before() : call( void Point.setX(int) ) {
      if (isActive) { … }
   }
   before() : call( void Point.setY(int) ) {
      if (isActive) { … }
   }
   before() : call( void Point.setColor(String) ) { … }
}
```

図 9 アドバイスのオン・オフ化（Caesar）

2.6.3. AspectJ5

AspectJ5 は，比較対象の中では最もよく進化が容易であるだけでなく，アドバイスのオン・オフ化の機能もうまくモジュール化できた．アスペクトがオンかオフかどうかに関わらず，常に実行されるようなアドバイスを実現するには単に @AlwaysActive アノテーションをそのアドバイスに付加す

ればよい.

```
@Activatable(active=false)
public aspect PointLogging {
   before() : call( void Point.setX(int) ) { … }
   before() : call( void Point.setY(int) ) { … }
   @AlwaysActive
   before() :call( void Point.setColor(String) ) { … }
}
```

図 10 アドバイスのオン・オフ化（AspectJ5）

2.7. 評価

評価結果を表 2 に示す. なお, 比較した各言語の括弧内の数字は, 2.4 節で評価を行ったときの順位を表している.

進化容易性は, 新しい要求をどのくらい容易に扱えたのかを表す. すでに述べたように, AspectJ が最も悪い. Caesar では, 新たにアドバイスを追加するだけでなく, 既存のアドバイス内に if 文を埋め込む必要がある. abc と AspectJ5 は新たにアドバイスを追加するだけである.

モジュール性は, アドバイスのオン・オフ化の要求をどのくらいクリーンにモジュール化できたのかを表す. AspectJ を除いて, 他の言語は, うまくモジュール化できていない.

表から分かるように, 前回の比較結果とは異なる順位となっている. 前回では, Caesar が最も良かったが, 今回は AspectJ5 が最良であった.

表 2 アドバイスのオン・オフ化の比較

	進化容易性	モジュール性
AspectJ(2)	×	×
Caesar(1)	△	×
abc(4)	○	×
AspectJ5(2)	○	○

3. 評価方法に関する考察と議論

2 節で述べた提案例からは, 実施した比較をより信頼できるような検証を行うためには, 以下の要素が必要なことが分かる.

- **例題は実際的であること.** 用いたコード例は非常にシンプルであった. 単に Point クラスの各メソッドの振る舞いをログすることを意図していただけである. シンプルであることの利点は, AspectJ に慣れていればコードの動作を理解することや, どんな課題を解決しようとしているのかの理解が容易であることである. 問題は, 現実性に欠けることである. たとえば, 各メソッドの振る舞いを個別にログすることが要求されるようなケースは, 現実的にどのくらいありえるだろうか.

  この提案例で扱った課題は, AOP 言語におけるアスペクト単位でのオン・オフ化である. アスペクトを動的にオン・オフ化すること（実際には, アスペクトの動的なデプロイメント）の有効性は, たとえば, Caesar では[18]において指摘されている. したがって, アスペクト単位でのオン・オフ化が場合によっては有効な時もある.

  提案例では, さらに, アドバイス単位でのオン・オフ化の課題も述べた. 結果として, 比較に用いた各言語がどのようにしてこの課題を扱うのかについての具体的な洞察を得ることができた. しかし, アドバイス単位でのオン・オフ化の要求が, 現実的に発生するかどうかはまだ明らかではない.

- **進化（変更タスク[9]）が現実的であること.** 連続するいくつかの進化的な要求を基に, 比較を行った. 結果として, 各言語がどのようにしてこの要求に対して異なる対処を行うのかについて具体的な洞察が得られた. たとえば, abc は AspectJ より進化が容易であったことなどである. このように, 数ステップにも及ぶ進化的な要求を基に比較が行われることは稀であるが, 2 節で示したように重要である. しかし, 例として示した進化的な要求が実際に発生するのかはまだ明らかではない.

3.1. ベンチマークの必要性

本稿で指摘したいのは, 十分に文章化された現実的なコード例（ベンチマーク）に容易にアクセス可能であるなら, シンプルな例題やケーススタディが持つ欠点を補完できるということである. たとえば, 2節で示した例で必要とされたのは, アドバイス単位でのオン・オフ化が要求されるような現実的な例である. あるいは, あるアスペクト内に二種類の異なるアドバイス（オン・オフ化できるアドバイスと常に実行されるアドバイス）が混合して定義されるような状況である.

以下では, 過去の研究を対象として, 2節での例以外でもそのようなコード例が必要とされていることを示す.

3.1.1. 不自然な例題

McEachen と Alexander は, アスペクトがすでにウィーブされた（織り込まれた）バイトコードに対して, さらにアスペクトをウィーブする時に起こりうる問題点を指摘している[17]. 彼ら自身が述べているように, この問題点を指摘するために用いられた例題はやや不自然である. そのため, 彼らは, 今後の課題として, 現実的なソフトウェア開発においてそのような問題点の起こりうる可能性を調査することを挙げている. もし, 現実的なコード例が十分に文章化されていているのならば, そのような調査のために利用できると思われる.

3.1.2. 暗黙のベンチマーク

GoF のデザインパターンは, 提案手法の有効性を示すために暗黙的にベンチマークとして用いられる傾向がある.

たとえば，Hanenberg と Unland は，Singleton，Visitor，Decorator の 3 つのパターンを実装する例を挙げ，Java，AspectJ，Hyper/J の実装上の欠点を指摘し，Sally 言語がどのようにしてこれらの欠点をなくすのかを議論している[7]. Observer パターンもよく利用される傾向がある[16][19][26][27].

HannemannらはAspectJを用いることによりGoFのデザインパターンすべての再実装を試みた[6]. Garciaらは，Hannemannらの定性的な試みを補完することを目的に，定量的な研究を行った[2]. 同じようにして，田中と一杉らは，MixJuice言語[38]を用いることによりGoFデザインパターンの改善を行った[30]. まだ報告されていないが，本稿で比較に用いた Caesar 言語や AspectJ5，あるいは ObjectTeams[35]などの他のアスペクト指向言語に関しても，GoFのデザインパターンの再実装を行うことによって，どのような異なる結果を生じるのか調査することは興味深い. しかし，当然ながら，GoFのデザインパターンだけがそのような改善や比較の調査を行う対象ではない.

多くの場合は，新しいテクニックの活用によるデザインパターンの単なる再実装であるが，デザインパターン化された設計から発生する明示的な進化シナリオの取り扱いを動機として，新たな言語が提案される場合がある. たとえば，Kniesel らは，Decorator パターンがすでに適用された状況から発生する進化的なシナリオは，オブジェクト指向言語ではうまく取り扱えないことを動機として，LogicAJ 言語[37]の利点を示している[13].

GoF におけるデザインパターンがよく使われる理由の一つは，これらのデザインパターンは十分な文章化がされており，論文を読む側だけでなく，例題としてデザインパターンを用いる側にも十分な情報が与えられるからだと思われる. デザインパターンは，実践において繰り返し発生することが分かっているため，実際に起きることが証明されている. したがって，デザインパターンを例題として扱う場合には，安心して利用することができる.

3.1.3. 再利用される例題

いくつかの例題は，同一の研究者や研究グループ内だけでなく，異なる研究者間でも再利用される.

たとえば，Sullivan が[25]において，統合システムにおける問題として示した例は，その後何度か再利用されている[21][22][26]. また，特に，AOP コミュニティでは図形オブジェクト（Point や Line）を用いた例が提案手法の有効性を示すために良く使われている[3][9][11][12][20].

3.2. まとめ

デザインパターンのように，実践で繰り返して発生するとして十分に文章化されたコード例が存在すれば，特定のデザインパターンや例題にオーバーフィットすることなく，より広い範囲で提案手法の有効性を検証するのに役立つ.

ベンチマークとして利用できる現実的なコード例を入手するのは容易ではない. オープンソースなソフトウェアでは，ソースコードを容易に入手できる. しかし，どのように設計が進化してきたのかについての情報を集めるのは困難である. 特に，新しく提案された言語によって開発されたソフトウェアのコードは入手困難となる.

4. 関連研究

[5]では，ソフトウェア進化のベンチマークの必要性を述べている. 彼らは，ソフトウェア開発全般におけるソフトウェアの進化におけるベンチマークに焦点を当てているが，本研究ではソフトウェア設計技術の検証に用いることのできるベンチマークの必要性を指摘している.

新しいプログラム言語の出現は，どのようにソフトウェアを開発・設計するのかに影響を与える. JHotDraw[4]は，Java で書かれたオープンソースの描画アプリケーションであり，アスペクト指向言語におけるテクニック（アスペクト・マイニングやアスペクト・リファクタリング）の有効性を試すための共通ベンチマークとして用いられることを目的としている.

[28]では， Object Team[35]の有効性を示すために MVC をベンチマークとして用いている. しかし，著者の知る限り，その後 MVC がベンチマークとして用いられたことはない.

5. まとめと今後の課題

ソフトウェア設計技術における検証方法は，現状よりも改善できる. 本稿では，典型的な提案例を用いて，ソフトウェア設計におけるベンチマークの必要性を指摘した.

本稿では，ソフトウェア設計技術に対する検証方法にどのような問題点があり，どこが改善できるのかを示した. しかしながら，ベンチマークの必要性を述べただけであり，ベンチマーク構築のために具体的にどうすれば良いのかは述べていない. これらは今後の課題であるがこの課題に向けての，信頼できるベンチマーク用のコードを収集するためのアプローチの一つのとしては，ソフトウェア設計における進化パターンを集めることを考えている[29]. 進化パターンは，設計の進化自体が繰り返し発生するパターンであるとして考え，それらを明示的に文章化しようという試みである. 複数の進化パターンを組み合わせることにより作成された進化シナリオは，ベンチマークとして扱えると期待している.

ベンチマーク構築にあたっての，本研究に関わる最も重要な研究は，Sim によるベンチマーキングの理論である[23][24]. 彼女の理論は，ソフトウェア設計におけるベンチマーキングを考える上でも役立つと思われるため，適用を検討している.

参考文献

[1] Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondřej Lhoták, Oege de Moor, Damien Sere

ni, Ganesh Sittampalam, and Julian Tibble. abc: An extensible AspectJ compiler. AOSD 2005.
[2] Alessandro F. Garcia, Cláudio N. Sant'Anna, Eduardo M. L. Figueiredo, Uirá Kulesza, Carlos J. P. Lucena, Arndt von Staa. Modularizing design patterns with aspects: a quantitative study. AOSD 2005.
[3] Shigeru Chiba and Kiyoshi Nakagawa. Josh: An Open AspectJ-like Language. AOSD 2004.
[4] Arie van Deursen, Marius Marin and Leon Moonen. AJHotDraw: A showcase for refactoring to aspects. Linking Aspect Technology and Evolution 2005.
[5] Serge Demeyer, Tom Mens, Michel Wermelinger. Towards a Software Evolution Benchmark. IWPSE 2001.
[6] Jan Hannemann and Gregor Kiczales. Design Pattern Implementation in Java and AspectJ. OOPSLA 2002.
[7] Stefan Hanenberg and Rainer Unland. Parametric Introductions. AOSD 2003.
[8] Stephan Herrmann. Object Teams: Improving Modularity for Crosscutting Collaborations. Net.ObjectDays 2002.
[9] Gregor Kiczales and Mira Mezini. Separation of Concerns with Procedures, Annotations, Advice and Pointcuts. ECOOP 2005.
[10] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin. Aspect-Oriented Programming. ECOOP 1997.
[11] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold. An Overview of AspectJ. ECOOP 2001.
[12] Gregor Kiczales and Mira Mezini. Aspect-Oriented Programming and Modular Reasoning. ICSE 2005.
[13] Gunter Kniesel, Tobias Rho and Stefan Hanenberg. Evolvable Pattern Implementations Need Generic Aspects. ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution.
[14] Sascha Kuzins. Efficient Implementation of Around-advice for the AspectBench Compiler. MSc thesis, Oxford University, September 2004
[15] Cristina Videira Lopes and Sushil Bajracharya . An Analysis of Modularity in Aspect-Oriented Design. AOSD 2005.
[16] Cristina Videira Lopes and Trung Chi Ngo. The Aspect Markup Language and its Support of Aspect Plugins. ISR Technical Report UCI-ISR-04-8. 2004.
[17] Nathan McEachen and Roger T. Alexander. Distributing Classes with Woven Concerns - An Exploration of Potential Fault Scenarios. AOSD 2005.
[18] Mira Mezini and Klaus Ostermann. Variability Management with Feature-Oriented Programming and Aspects. FSE 2004.
[19] Mira Mezini and Klaus Ostermann. Conquering Aspects with Caesar. AOSD 2003.
[20] Klaus Ostermann, Mira Mezini, and Christoph Bockisch. Expressive Pointcuts for Increased Modularity. ECOOP 2005.
[21] Hridesh Rajan and Kevin Sullivan. Eos: Instance-Level Aspects for Integrated System Design. In the proceedings of ESEC/FSE 2003.
[22] Kouhei Sakurai, Hidehiko Masuhara, Naoyasu Ubayashi, Saeko Matuura, Seiichi Komiya. Association Aspects. AOSD 2004.
[23] Susan Elliott Sim, Steve Easterbrook, and Richard C. Holt. Using Benchmarking to Advance Research: A Challenge to Software Engineering. ICSE 2003.
[24] Susan Elliott Sim. A Theory of Benchmarking with Applications to Software Reverse Engineering. Ph.D. Thesis, Department of Computer Science, University of Toronto, 2003.
[25] Kevin Sullivan, Lin Gu and Yuanfang Cai. Non-Modularity in Aspect-Oriented Languages: Integration as a Crosscutting Concern for AspectJ. AOSD 2002.
[26] Tetsuo Tamai, Naoyasu Ubayashi and Ryoichi Ichiyama. An Adaptive Object Model with Dynamic Role Binding. ICSE 2005.
[27] Éric Tanter, Jacques Noyé, Denis Caromel and Pierre Cointe. Partial Behavioral Reflection: Spatial and Temporal Selection of Reification. OOPSLA 2003.
[28] Matthias Veit, Stephan Herrmann. Model-View-Controller and Object Teams: A Perfect Match of Paradigms. AOSD 2003.
[29] 下滝 亜里. ソフトウェア設計における進化パターン. 情報処理学会 第 67 回全国大会. 2005.
[30] 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.
[31] AspectJ におけるアスペクトをオンやオフにするアスペクトライブラリの開発, http://www.ncfreak.com/asato/doc/aspect-activation.html, 2004.
[32] Eclipse AspectJ, http://eclipse.org/aspectj/, 2005
[33] The AspectJ 5 Development Kit Developer's Notebook. http://www.eclipse.org/aspectj/doc/next/adk15notebook/index.html 2005.
[34] CaesarJ, http://caesarj.org/, 2005
[35] Object Teams, http://www.objectteams.org, 2005

[36] JSR 175: A Metadata Facility for the Java Programming Language. http://www.jcp.org/en/jsr/detail?id=175
[37] LogicAJ – A Uniformly Generic and Interference-Aware Aspect Language, http://roots.iai.uni-bonn.de/research/logicaj/, 2005
[38] プログラミング言語 MixJuice. http://staff.aist.go.jp/y-ichisugi/ja/mj/, 2005.