社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ファイル編集履歴に基づいてデザインパターン適用事例を分析する手法の検討

浦田　大地†　　吉田　則裕†　　藤原　賢二†　　飯田　　元†

† 奈良先端科学技術大学院大学 情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
E-mail: †{daichi-u,yoshida,kenji-f}@is.naist.jp, †iida@itc.naist.jp

あらまし　デザインパターンは設計上の問題に対する解決策を示すが，適用すべき対象やコンテキストが限られる．デザインパターンを適用した効果を評価する手段として，リポジトリマイニングに基づいて長期間の分析を行う方法が考えられるが，コミット単位より細かい粒度の編集を扱いづらい．本研究では，Mylyn を用いて収集したファイル編集履歴を利用して，長期間かつ細粒度の分析が可能な手法を検討する．
キーワード　デザインパターン，ファイル編集履歴，Mylyn

A Discussion on the Analysis of Design Pattern Instances Using File Modification Records

Daichi URATA†, Norihiro YOSHIDA†, Kenji FUJIWARA†, and Hajimu IIDA†

† Graduate School of Information Science, Nara Institute of Science and Technology
Takayama-cho 8916–5, Ikoma-shi, Nara, 630–0192 Japan
E-mail: †{daichi-u,yoshida,kenji-f}@is.naist.jp, †iida@itc.naist.jp

Abstract　Design patterns are general solutions to common problems in software design. However, it has been reported that they are not effective for some given problems and contexts. For evaluating the effects of design pattern instances, long-term analysis based on repository mining is a considerable approach. However, it is limited to commit-level analysis of source code modification. In this paper, we discuss an approach for long-term analysis of design pattern instances using file modification records collected from the Mylyn logs.
Key words　design pattern, file modification record, mylyn

1. はじめに

ソフトウェア開発におけるデザインパターンとは，頻出する設計上の問題への解決策をカタログとしてまとめたものである [1]．デザインパターンを適用することで，熟練者らのノウハウを利用してソフトウェアの再利用性を向上させることができる．特に，保守作業においてソフトウェアの開発効率を向上させられる．ただし，デザインパターンはあらゆるケースにおいて手当たり次第に適用するべきものではない．個々のデザインパターンは，適用する対象となる問題や前提となるコンテキストがそれぞれ存在する．デザインパターンを適用する対象やコンテキストを誤った場合，効果を得られないだけでなく，過度に複雑な設計となり開発者の理解を妨げる要因となり得る [2]．したがって，開発者らはデザインパターンを適用した際に，適用した対象やコンテキストが誤っていないことを確認する必要がある．

そこで，デザインパターンが適用された事例において，適用による効果の有無を評価する手段が必要となる．デザインパターンを適用した効果を評価する既存の観点として，開発者の活動や欠陥率が挙げられる．ここで，開発者の活動とは，開発者によるソースファイルの選択や編集を指す．

開発者の活動に着目した既存研究 [3] では，デザインパターンを適用したケースと適用していないケースにおける開発者の保守作業を比較する手法が提案されている．このような手法は詳細な議論が可能な反面，高コストであるため大規模なソフトウェアを対象とした分析や長期間に渡る分析には不向きである．また，既存の適用事例を評価したい場合には，デザインパターンを適用していないプログラムを新たに作成する必要があるため本質的でない．

欠陥率に着目した既存研究 [4] では，開発履歴の分析により，デザインパターンが適用された箇所の欠陥率を評価する手法が提案されている．このような手法は自動化によって大規模な対

象の分析や長期間に渡る分析が可能である反面，コミットよりも細粒度の保守作業について議論できない．

本研究では，開発者の活動に着目しつつ，大規模な対象の分析および長期間に渡る分析が可能な手法を検討する．このような手法を実現するためには，編集履歴の自動収集と定量化が必要となる．本研究では，この手段として Zhang らの提案するファイル編集パターン [5] を利用した．また，ケーススタディとしてオープンソースソフトウェア（OSS）である Mylyn(注1) の開発履歴に対して手法を適用し，手法の適用可能性を検討した．

本稿の構成は次の通りである．まず 2 章では，議論の前提となる技術としてデザインパターンおよびその適用事例の評価手法，ファイル編集パターンについて説明する．3 章では，検討の対象となる分析手法について述べる．4 章では，3 章で述べた分析手法についてケーススタディを用いて検討し，考察を加える．最後に 5 章で本稿の要点をまとめて今後の課題を述べる．

2. 関 連 技 術

2.1 デザインパターン

ソフトウェア開発に用いられるデザインパターンとは，オブジェクト指向開発において頻出する設計上の問題への解決策を体系立てて整理したものである．未熟な開発者と熟練した開発者では，ソフトウェア開発における生産性に大きな差があり，一般に熟練した開発者となるには多くの時間と経験を要する．しかし，未熟な技術者であっても，デザインパターンによって熟練した開発者らによって蓄積されたノウハウを学び，利用し，共通言語として利用することができる．

最も有名なデザインパターンとして，GoF（the Gang of Four）のデザインパターン [1] が挙げられる．GoF のデザインパターンとは，Gamma らが提唱した 23 種類のデザインパターンの総称であり，オブジェクト指向のソフトウェア開発において，再利用性の高い設計とするためのノウハウがまとめられたものである．以降本稿でデザインパターンと記載した場合，特に断りがない場合は GoF のデザインパターンを指すものとする．

デザインパターンの具体例として，本研究で扱う State パターンについて説明する．State パターンは，状態によって振る舞いのパターンを変えるような実装において，1 つのクラスが巨大化することを防ぐ効果や状態の追加を容易にする効果などがある．State パターンの構成を図 1 に示す．ConcreteState クラスには，Context クラスのある 1 つの状態に関する振る舞いが実装される．Context クラスの個々の振る舞いは State クラスによってカプセル化されているので，状態を追加する場合は新たに ConcreteState クラスを追加すればよい．したがって Context クラスは変更の必要がなく，状態の増加によって巨大化することもない．ただし，ごく短い条件分岐で実現可能な処理に適用したり，状態を細分化しすぎると過度に複雑な構造となって開発者の理解を妨げる．

デザインパターンを適用した効果を評価する既存の観点として，開発者によるソースファイルの選択や編集といった開発者の活動や，ソフトウェアの欠陥率が挙げられる．

開発者の活動に着目した既存研究では，デザインパターンを適用したケースと適用していないケースにおける開発者の保守作業を比較する手法が提案されている．Prechelt らは，C++ で記述された小規模なプログラムの保守作業におけるデザインパターンの適用効果を評価している [3]．彼らは，ソフトウェアエンジニアを被験者として，デザインパターンを適用した実装とより簡素な実装において保守作業に要する時間を比較した．この実験では，デザインパターンを適用した場合に，ほとんどのケースにおいて保守作業に要する時間が短くなったことが示されている．開発者の活動に着目した分析は，保守作業に焦点を当てることができるため，デザインパターンを適用した効果を的確に議論できる．しかし，開発者の活動に着目した既存の評価手法はコストが高く，大規模なソフトウェアを対象とした評価や長期間に渡る評価を行うことが困難である．これは，あるソフトウェアを評価する際に，振る舞いの等しい 2 通りのプログラム（デザインパターンを適用したものと適用していないもの）を用意する必要があるためである．さらに，新しく振る舞いが同じプログラムを作成して比較することは，既存のデザインパターン適用事例を評価したい場合において本質的でない．

欠陥率に着目した既存研究では，デザインパターンが適用された箇所の欠陥率を，開発履歴の分析により評価する手法が提案されている．Vokáč は，いくつかのデザインパターンに関連するクラスの欠陥率を比較している [4]．彼のケーススタディでは，C++ で記述された 50 万行の産業用ソフトウェアの 3 年間に渡る開発履歴を分析している．このように，開発履歴の分析は大規模な対象の分析や長期間に渡る分析が可能である．しかし，彼らは版管理システムから取得した開発履歴を分析しているため，コミットに至るまでの開発者による試行錯誤やそれに要した時間を考慮できない．したがって，作業効率の議論が難しく，保守作業を効率化するデザインパターンの適用効果を評価する上では適切でない．

上述のように，コミットよりも細粒度な開発者の活動に着目しつつ，長期間の分析や大規模な対象の分析が可能なデザインパターン適用事例の分析手法は提案されていない．

2.2 ファイル編集パターン

Zhang らは，ソフトウェアのソースファイルがどのように編集されたかという情報から，4 つの特徴的なパターンを抽出し

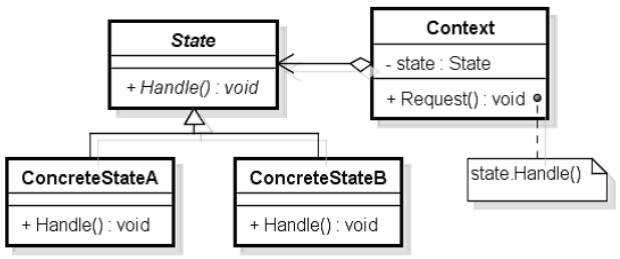

図 1　State パターンの構成

(注1)：http://www.eclipse.org/mylyn/

てファイル編集パターンとして提案した [5]．彼らが提案したファイル編集パターンの概要を以下に示す．

（1） 同時編集パターン

2 人以上の開発者が 1 つのファイルを同時に編集していたことを示すパターンである．あるファイルの編集履歴において，2 人以上の開発者による編集時間に一部でも重複が存在する場合に該当ファイルで検出される．

（2） 並行編集パターン

1 人の開発者が 2 つ以上のファイルを同時に編集していたことを示すパターンである．2 つ以上のファイルの編集履歴において，ある開発者による編集時間に一部でも重複が存在する場合に全てのファイルで検出される．

（3） 長時間編集パターン

あるファイルの編集時間が長時間に及んでいたことを示すパターンである．あるファイルに加えられた編集の合計時間が，調査対象となる全ファイルについてそれぞれ計算した合計編集時間を降順に並べたときに上位 1/4 となる時間よりも長かった場合に検出される．

（4） 中断編集パターン

あるファイルに加えられた編集が長時間中断されたことを示すパターンである．ある開発者によってあるファイルに加えられた各編集間の時間（中断時間）の合計が，調査対象となる全ファイルから同様に計算した中断時間を降順に並べたときに上位 1/4 となる時間よりも長かった場合に検出される．

ファイル編集パターンは，保守作業においてファイルが編集された履歴を分析することで検出される．ここで，ファイルが編集された履歴（編集履歴）とは，「どのファイル」が「誰に」よって「いつから」「いつまで」編集されたかという情報を指す．ファイル編集パターンは，各パターンに対応するメトリクスを計測することで，ファイルごとに検出される．これらメトリクスは，該当ファイルの編集履歴および調査対象となるすべてのファイルの編集履歴から計算される．いずれのメトリクスも，いずれも値が大きくなるほど各編集パターンの傾向が強くなる．例えば，同時編集パターンを検出するメトリクスの値が大きいほど，同時編集が繰り返されているか，同時に多くの人数が 1 つのファイル編集していることを示す．なお，ファイルによって，ファイル編集パターンが 1 つも検出されない場合と，1 つ以上のパターンが検出される場合とがある．

彼らの研究では，ファイル編集パターンを検出するメトリクス値を計算するために Mylyn のログデータを利用している．Mylyn は統合開発環境 Eclipse のプラグインであり，イベント（開発者によるファイルの編集や選択）およびその開始終了時間，対象ファイルなどの情報を記録する．一部 OSS では開発に Mylyn を利用しており，開発履歴からログを取得できる．

本研究では，ファイル編集パターンを検出するメトリクスを，ファイル編集履歴に基づく情報としてデザインパターン適用事例の分析に用いる．

3. 提 案 手 法

提案手法では，ファイルの編集履歴に基づいてデザインパターンの適用事例を分析する．提案手法により，開発者の活動に着目しつつ，大規模かつ長期間に渡るデザインパターンの適用事例の分析が可能であれば，以下の貢献が期待できる．未熟な開発者であっても，新たに適用したデザインパターンの効果を分析しながら開発することで，適用した対象およびコンテキストが適切であったかを確認できる．あるいは，既に適用されているデザインパターンが開発者に与えた効果を長期な視点で分析することで，設計を見直し，リファクタリングを実施するきっかけとなる情報を開発者に提供できる．

提案手法の概要を図 2 に示す．提案手法の入力は，デザインパターンが適用されており，かつファイルの編集履歴が取得できるソフトウェアの開発履歴である．提案手法は，デザインパターンの定義から仮説を立て，ソフトウェアリポジトリから抽出した情報によって仮説を検証する．検証の結果に基づいて，適用されたデザインパターンの効果の有無をパターンごとに判断する．

提案手法の詳細な手順を以下に述べる．

準備 提案手法の入力となるデータを収集する．入力となり得るソフトウェアの開発履歴は，デザインパターンが適用されており，ファイルの編集履歴を取得できる必要がある．ファイルの編集履歴には，「編集の対象ファイル」・「編集を行った開発者」・「編集の開始時間」および「編集の終了時間」に関する情報が含まれている必要がある．提案手法ではこれらの情報を含む編集履歴として，Mylyn のログ想定している．Mylyn のログと異なる形式のファイルの編集履歴を用いる場合には，上述の条件を満たしていることが必要である．

手順 1 ファイルごとに導出されるファイル編集パターンのメトリクスとの関連を考慮して，デザインパターンを構成する各要素とそれに該当するクラス含むファイルを対応付ける．例えば，State パターンの適用事例を評価する場合，パターンの各要素（Context・State・ConcreteState）と該当するクラスを含むファイルを対応付ける．デザインパターンを適用した開発者自身が分析を行う場合は自明であるが，既存のソフトウェアに対して分析を行う場合はツールを用いるか目視で確認することでデザインパターンを検出する必要がある．

手順 2 分析対象となるソフトウェアの編集履歴から，ファイル編集パターンのメトリクスを計測する．各メトリクスの計測式は Zhang らの定義 [5] を参照されたい．

手順 3 デザインパターンごとに，適用効果に対する仮説を定義に基づいて立てる．本研究で扱う仮説とは，デザインパター

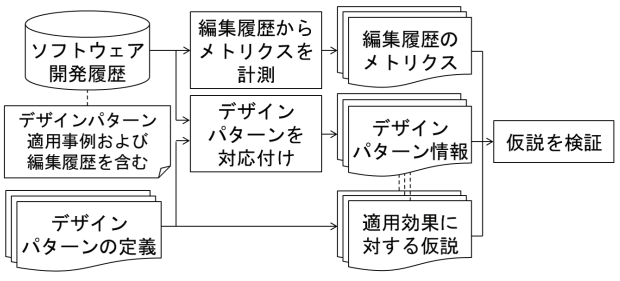

図 2 提案手法の概要

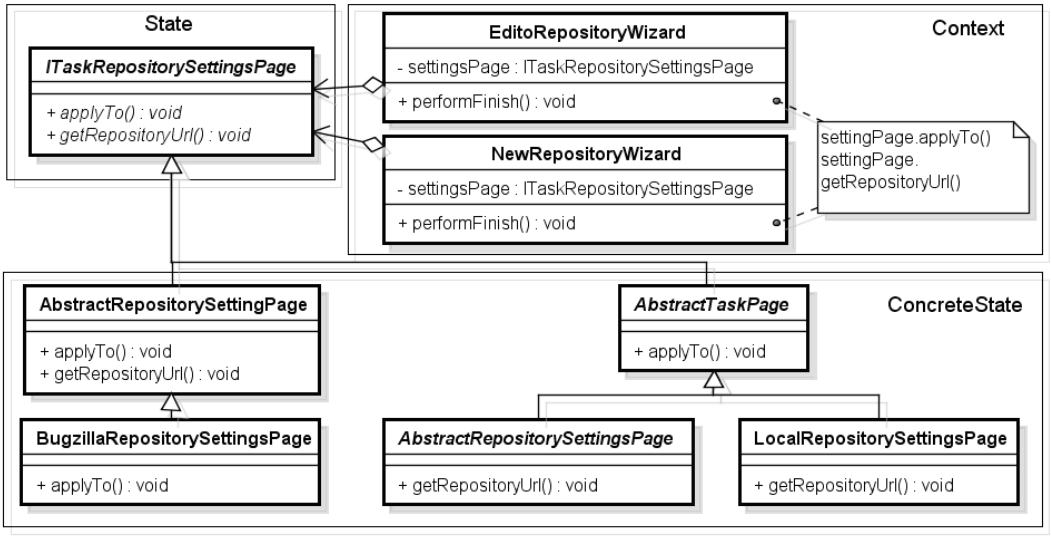

図 3　検出された State パターン適用事例の一部

ンの各要素におけるファイル編集パターンの傾向を予測したものである．デザインパターンの特定の要素におけるファイル編集パターンのメトリクス値が，同じパターンのその他の要素やソフトウェア全体におけるメトリクス値と比較して大きくなるか，小さくなるかを予測する．

手順 4　デザインパターンの要素に対応付けたファイルにおけるファイル編集パターンのメトリクス値を比較し，仮説を検証する．

4.　ケーススタディ

前章で述べた提案手法を OSS に適用するケーススタディを行った．ケーススタディの目的は，提案手法の適用可能性を検討することである．

4.1　データセットと利用ツール

Mylyn の開発履歴を対象にケーススタディを行った．

ケーススタディでは，**手順 2** を省略してファイル編集パターンに関するメトリクス値として Zhang らの実験データ [5](注2) を利用した．Zhang らの実験データには，Mylyn のファイルにおけるファイル編集パターンのメトリクス値が含まれている．Zhang らは 2009 年 1 月 1 日から 2010 年 12 月 31 日までの Mylyn ログを利用してファイル編集パターンのメトリクスを計測している．

Mylyn に適用されているデザインパターンの検出には，Tsantalis らのツール [6](注3) を用いた．Tsantalis らのツールは入力に Java の.class ファイルが必要であるため，デザインパターンの検出はリリースバージョン 3.0.3 を対象に行った．リリースバージョン 3.0.3 は 2008 年 10 月 15 日にリリースされている．これは，Zhang らの実験において Mylyn ログが収集された期間以前の最終リリースである．

(注2)：https://bitbucket.org/serap/fileeditingpatternstudy/
(注3)：http://users.encs.concordia.ca/~nikolaos/pattern_detection.html

4.2　ケーススタディ詳細

ケーススタディでは，State パターンの適用事例を用いて提案手法を検討した．

手順 1 では，前節で述べたツールで検出されたデザインパターンの適用箇所を目視で確認し，実際に適用されていることを確認した事例のみを適用事例とした．

手順 2 のメトリクス計測については，前節で述べた通り Zhang らのデータを利用したため省略した．

手順 3 では，State パターンの ConcreteState クラスに対して次の仮説を立てた．

仮説　ConcreteState は編集頻度は高いが同時編集されにくい ConcreteState クラスは，依存関係の下位に存在するため，抽象度の高い State クラスなどに比べて変更されやすいと考えられる．ただし，Context クラスの 1 つの状態に関する振る舞いが実装されるクラスであるため責務が限定されており，複数の開発者による同時編集はされにくいと考えられる．

手順 4 では，ファイル編集パターンの各メトリクスについて ConcreteState に関連するファイル群と State パターンのその他の要素（State および Context）とで最大値，最小値，中央値を比較した．また，参考値として Mylyn 全体とも比較した．ここで，Mylyn 全体とは，Zhang らの実験データにファイル編集パターンのメトリクス値が含まれている Mylyn のファイル全てを指す．リリースバージョン 3.0.3 における Mylyn の一部ファイルは，ファイル編集パターンの情報が存在しないため，これに含まない．

4.3　結果と考察

手順 1 において，State パターンの各要素に対応付けたファイル数の一覧を表 1 に，定期用事例のクラス図の一部を図 3 示す．ツールで検出した後に目視で確認できた State パターンの適用事例は 3 つであった．3 つの事例のうち 2 つで State クラスが共通していたため，State には 2 つのファイルを関連付けた．前節で述べた通り，Mylyn 全体の関連ファイル数はリリースバージョン 3.0.3 における Mylyn の全ファイル数と異なる．

表 1　State パターンの構成要素に関連するファイル数

	関連ファイル数
State	2
ConcreteState	17
Context	3
Mylyn 全体（参考）	1177

手順 4 において，各メトリクスの比較に用いた箱ひげ図を図 4 から図 7 に示す．各図において，横軸はファイル編集パターンのメトリクス値を表し，値が大きいほどパターンの傾向が強い．縦軸の上から順に，ConcreteState に関連するファイル群（ConcreteState），State または Context に関連するファイル群（State-Context）および Mylyn 全体を意味する．

図 6 より，ConcreteState は State-Context や Mylyn 全体と比べて中央値が高かった．そして，図 7 より，ConcreteState は中央値で見ると State-Context と同程度の間隔を空けて編集されていた．したがって，ConcreteState は複数回にわたり比較的長時間編集されているか，State-Context よりも多く編集されていると考えられる．また，図 4 では ConcreteState の中央値が State の最小値と同程度であり，最大値は外れ値を除き同程度であった．Mylyn 全体と比べると，ConcreteState の値域は高い値に分布しているが，中央値は同程度である．すなわち，ConcreteState は State-Context に比べて複数の開発者から同時に編集されにくく，Mylyn 全体と比べても同程度であると分かった．

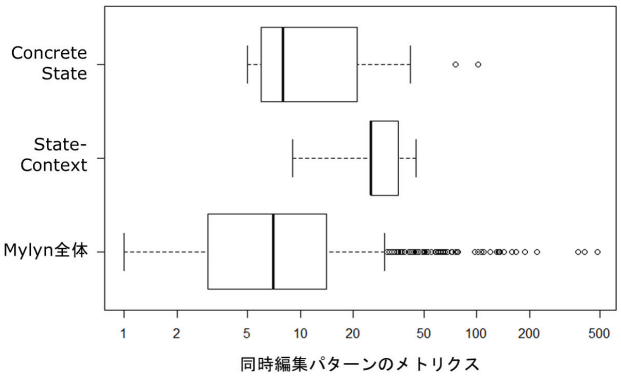

図 4　同時編集パターンのメトリクスの比較

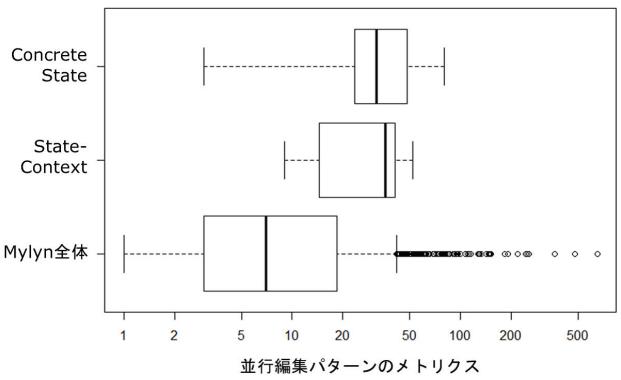

図 5　並行編集パターンのメトリクスの比較

以上より**手順 3** で立てた仮説は成立しており，分析の対象となった適用事例において State パターンは正しく適用されているといえる．

本ケーススタディにおいて行った手順は，**手順 1** と**手順 2** を除き自動化することができる．**手順 2** のメトリクス計測については，Mylyn ログの取得およびファイル編集パターンのメトリクス計測であるため，自動化は比較的容易である．**手順 4** の仮説の検証については，より多くの分析データを収集し，それを基に閾値を設けるなどして自動化が可能である．また，**手順 2** については，一度立てた仮説を使いまわすことができるため自動化するうえで問題にならない．ただし，**手順 1** について，デザインパターンの各構成要素を完全に特定することは非常に困難である．

5. おわりに

本研究では，Mylyn のログに基づくファイル編集パターンのメトリクスを用いてデザインパターンの適用事例を分析する手法について検討した．ケーススタディでは，Mylyn の開発履歴を対象に提案手法を適用し，State パターンの ConcreteState に着目した分析を行った．分析の結果，ConcreteState に対する仮説を立証できたため提案手法には適用可能性があるものと考える．

今後の課題として，提案手法を用いた分析事例を増やすことが挙げられる．現時点で提案手法を用いて分析を行った対象は

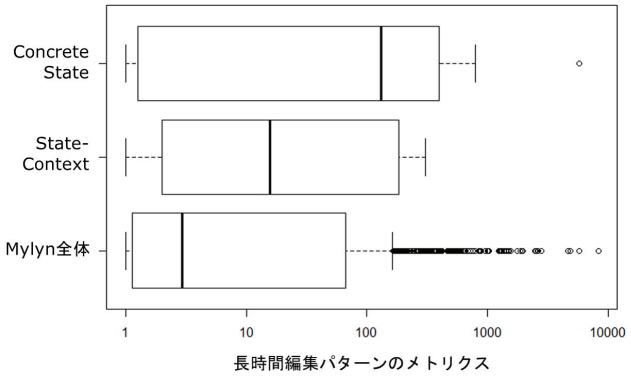

図 6　長時間編集パターンのメトリクスの比較

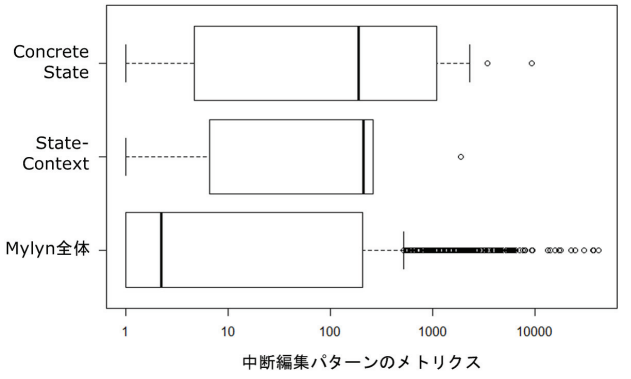

図 7　中断編集パターンのメトリクスの比較

Mylyn の一部サブプロジェクトにとどまっており，より多くの分析事例が求められる．ただし，Mylyn のログを取得できるソフトウェアの開発履歴が少ないため，Mylyn に代わるファイル編集履歴の取得方法も調査する必要がある．限られた分析対象を用いて分析事例を増やすためにも，State パターン以外のデザインパターン適用事例を対象に提案手法を適用してゆく．

また，ファイル編集パターンのメトリクスの統計値に現れる特徴が，デザインパターンに起因すると明言できないことも問題として挙げられる．これについては，一般的なソフトウェアメトリクスとの相関を調査するなどして明らかにする必要がある．

文　　献

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995.

[2] R.C. Martin, Agile Software Development: Principles, Patterns, and Practices, Prentice Hall PTR, 2003.

[3] L. Prechelt, B. Unger, W.F. Tichy, P. Brössler, and L.G. Votta, “A controlled experiment in maintenance comparing design patterns to simpler solutions,” IEEE Trans. Softw. Eng., vol.27, no.12, pp.1134–1144, 2001.

[4] M. Vokáč, “Defect frequency and design patterns: An empirical study of industrial code,” IEEE Trans. Softw. Eng., vol.30, no.12, pp.904–917, 2004.

[5] F. Zhang, F. Khomh, Y. Zou, and A.E. Hassan, “An empirical study of the effect of file editing patterns on software quality,” Inproceedings of the 2012 19th Working Conference on Reverse Engineering, pp.456–465, WCRE ’12, IEEE Computer Society, 2012.

[6] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S.T. Halkidis, “Design pattern detection using similarity scoring,” IEEE Trans. Softw. Eng., vol.32, no.11, pp.896–909, 2006.