プロセスメトリクスを用いたメソッド抽出事例の特徴調査

田中 大樹[1,a)]　吉田 則裕[2,b)]　藤原 賢二[3,c)]　崔 恩瀞[1,d)]　飯田 元[1,e)]

概要：リファクタリングとは，外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理することである．メソッド抽出は，既存のメソッドの一部を新規メソッドとして抽出するリファクタリングパターンの一つであり，実施される回数が多いことも知られている．メソッド抽出リファクタリングを支援するためには，開発者がどのようなメソッドを抽出の対象としているかを調査する必要がある．過去，メソッド抽出対象となるメソッドの長さ，パラメータ数など，コード片のスナップショットから得られるメトリクスを用いた，メソッド抽出対象の調査が実施されており，それを利用した支援手法も提案されている．異なる側面からコード片の特徴を計測する方法として，メソッドの変更回数，変更に関わった開発者数などの開発履歴から得られるプロセスメトリクスがある．開発プロセスはソースコードの品質に影響を与えるものであり，品質の良し悪しはコード片をリファクタリングするか否かの判断材料になるため，プロセスメトリクスはリファクタリング対象の特徴を適切に表現できる可能性がある．本研究では，プロセスメトリクスを用いてメソッド抽出対象の特徴を適切に表現できるかを調査した．調査の結果，いくつかのプロセスメトリクスは，メソッド抽出対象となるメソッドと対象ではないメソッドの間で有意差が確認できた．

キーワード：メソッド抽出リファクタリング，プロセスメトリクス，実証的ソフトウェア工学

1. はじめに

リファクタリングとは，外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理することである [1]．数多くのリファクタリングパターンが定義されており，どのようなコード片に対してどのリファクタリングを実施すべきかが示されている [2]．

リファクタリングはソフトウェア開発において必要な作業であるが，開発者が大規模なソースコードからリファクタリング候補を見つけることは難しい．そのため，開発者のリファクタリング作業を支援するための技術やツールが多く提案されている [3] [4] [5] [6]．

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[2] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8601, Japan
[3] 豊田工業高等専門学校
National Institute of Technology, Toyota College, Toyota, Aichi 471–8525, Japan
[a)] tanaka.daiki.sx4@is.naist.jp
[b)] yoshida@ertl.jp
[c)] fujiwara@toyota-ct.ac.jp
[d)] choi@is.naist.jp
[e)] iida@itc.naist.jp

これらの技術を効果的なものにするためには，開発者のリファクタリング作業の実態を正確に把握する必要があり，それを目的とした調査が行われている．後藤らは，メソッド抽出という頻繁に実施されるリファクタリングパターンに着目し，メソッド抽出の対象となったメソッドの特徴を調査した [7]．具体的には，抽出対象となったメソッドのサイズと凝集度など，コード片に関するメトリクスを計測し，抽出が行われなかったメソッドとの比較調査をしている．

ソースコードの特徴を定量的に表現する方法として，ソースコードのスナップショットから計測する以外に，開発の履歴情報を用いて計測するプロセスメトリクスがある [8]．開発プロセスはソースコードの品質に影響を与えるものであり，リファクタリングは一般的に，保守性や可読性が低い，品質の悪いコード片を対象に実施されるため，その影響はコード片がリファクタリングの対象となるか否かにも及んでいると考えられる．

そこで，本研究ではプロセスメトリクスを用いて，メソッド抽出の対象となったメソッドの特徴を調査した．ソースコードの品質に影響を与えていると考えられる，コード片の変更回数や，その変更に関わった開発者数等のメトリクスを定義し，それらを用いて抽出の対象となったメソッドと対象とならなかったメソッドを比較した．

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

図 1　メソッド抽出リファクタリング

調査の結果，抽出の対象となったメソッドは，それが属するクラスの変更回数とその変更に関わったオーサーの数が，対象とならなかったメソッドと比べ多いことがわかった．

以降，2 節では本研究の背景となる，メソッド抽出リファクタリングとリファクタリングの支援手法を紹介する．続く 3 節では本研究の調査方法を説明し，4 節で調査結果とその考察を述べる．5 節で本研究のまとめと今後の課題を述べる．

2. 背景

2.1 メソッド抽出リファクタリング

メソッド抽出リファクタリングとは，既存のメソッドの一部を新規メソッドとして抽出するリファクタリング操作である [1]．長すぎるメソッドや，複数の機能が実装されたメソッドを適切に分割することで，ソースコードの可読性や保守性を向上させる．

図 1 にメソッド抽出リファクタリングの例を示す．図 1 中の上側，printOwing メソッドのコメント print details 以下の 2 文が抽出され，新たに printDetails メソッドとして定義されている．抽出元の printOwing メソッドには，抽出された printDetails メソッドの呼び出し文が追加されている．

メソッド抽出リファクタリングは実施される回数が多い [9]．また，メソッドはソースコード中に大量に存在するため，それらの中からリファクタリング候補となるものを開発者が適切に選択することは難しい．これらの点から，支援の必要のあるリファクタリングであると考えられる．

2.2 リファクタリングの支援

リファクタリングの支援手法について，これまでに提案されているものをいくつか紹介する．

Tsantalis らは，JDeodorant というリファクタリング支援ツールを開発している [3]．このツールは，ソースコードから不吉な臭い [1] と呼ばれる，将来的に不具合の原因となりうるコード片を特定し，そのコード片に適用すべきリファクタリングを開発者に提案する．メソッドが長く複雑である Long Method や，1 つのクラスが大きすぎる God Class などの複数の不吉な臭いに対応している．

Danilo らは，メソッド抽出リファクタリングの対象メソッドを特定し，それらをメソッド抽出を適用すべき度合いでランキングして開発者に提示する手法を提案した [4]．この手法は，メソッドから抽出可能なコード片全てをリファクタリング候補とし，抽出するコード片と抽出元のコード片の依存関係の弱いものを，抽出すべきコード片と捉え，ランキングの上位に順位付けする．

後藤らは，機械学習を用いてメソッド抽出リファクタリングの対象メソッドを開発者へ推薦する手法を提案した [5]．この手法は，メソッドの長さ，凝集度などのコード片のメトリクスを計測し，それに基づいて抽出すべきメソッドを推薦する．今里らは，機械学習の入力にメソッドの構文情報を用いて，抽出対象メソッドを推薦する手法を提案した [6]．メソッド中の for 文，if 文などの各プログラム要素の出現回数を 1 つの次元とした構文情報ベクトルを用いて，抽出対象メソッドを推薦する．

これらの手法は，いずれもソースコードのスナップショットから得られるコード片の特徴を用いて何らかの支援を行う．スナップショットからコード片の特徴を計測する以外に，開発履歴を用いてコード片の特徴を計測するプロセスメトリクスがある．開発プロセスはソースコードの品質に影響を与えることが知られており，品質の良し悪しはコード片をリファクタリングするか否かの判断材料になる．そのため，プロセスメトリクスを用いたコード片の特徴付けは，リファクタリング対象箇所の特定に有効であり，スナップショットから得られるメトリクスと併用することで，従来の支援手法の改善が期待できる．しかし，未だプロセスメトリクスを用いたリファクタリング支援手法は提案されておらず，手法確立のための調査も実施されていない．

3. 調査方法

調査方法の概要を図 2 に示す．調査は次の手順で進める．

step 1　特定のリビジョンを 1 つ選択し，そのリビジョンの全メソッドのリストを得る．

step 2　各メソッドに対して，選択したリビジョン以降でメソッド抽出されたかどうかを検査し，抽出ありメソッドと抽出なしメソッドの 2 群に分ける．

step 3　抽出なしメソッド群から，抽出ありメソッドと同じ数だけランダムにメソッドを選択し，これらと抽出ありメソッド群を合わせて調査対象データセットとする．

step 4　開発履歴より，対象データセットの各メソッドの

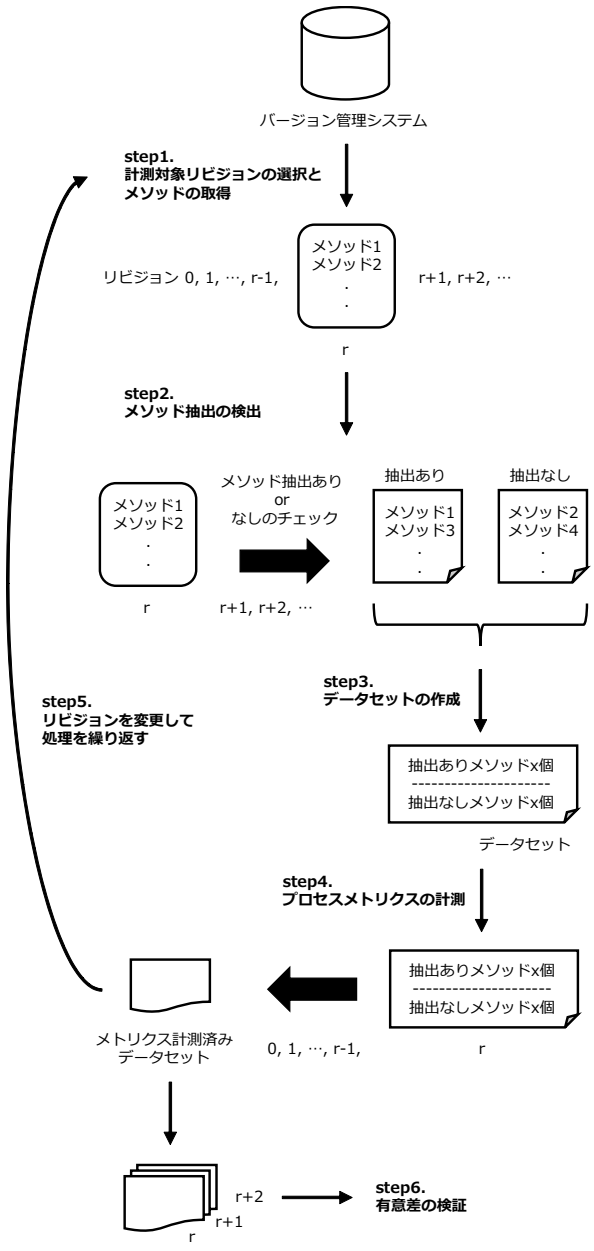

図 2　調査方法の概要

プロセスメトリクスを計測する．

step 5　リビジョンを変更して step 1 – 4 を繰り返し，各リビジョンで取得したメソッドのプロセスメトリクスを最終的な調査対象とする．

step 6　抽出ありメソッドと抽出なしメソッドの 2 群間でメトリクスに有意な差があるか検証する．

開発履歴の長さがプロセスメトリクスの値を左右するため，特定の 1 つのリビジョン内でのメソッドを比較する．

メソッド抽出リファクタリングの検出には，藤原らの提案した Kenja[10] を用いる．Kenja は，畑らの提案した細粒度なバージョン管理システムである historage[11] を利用して，メソッド抽出を検出する．historage は，各クラスに実装されたメソッドごとに開発履歴を記録できる．これを用いて，リビジョン間での変更情報からメソッド抽出が実施されたと推定されるメソッドの組み合わせを特定し，抽出元のメソッドから削除されたコード片と，抽出先のメソッドの類似度が閾値以上のものを，メソッド抽出リファクタリングとして検出する．その検出精度は，適合率が 0.96，再現率が 0.86 と高いものである．また，大規模な開発履歴に対しても，高速にリファクタリング検出できるという利点がある．本実験では，比較調査のために多くのメソッド抽出事例が必要となるので，このツールを利用した．

また，選択したリビジョン以降でメソッド抽出されたメソッドを抽出ありメソッドとする．これは，抽出の必要のあるメソッドであっても，即座にリファクタリングが実施されるわけではないので，選択したリビジョン以降でメソッド抽出された場合，そのリビジョンですでにメソッド抽出の対象となる特徴を持っていると考えたためである．

3.1　メトリクス計測

計測するメトリクスを表 1 に示す．計算式において，r は計測対象リビジョン，m_r，C_{m_r} はそれぞれリビジョン r に存在するメソッドと，そのメソッドが属するクラスを表す．$isChanged$ は，引数に指定された m_r，C_{m_r} に変更があったときは 1，なかったときは 0 となる．また，$changedRevisions$ は，引数に指定された m_r，C_{m_r} がリビジョン r 以前に変更されたリビジョンの集合を，$author_i$ はリビジョン i のオーサーの数を表す．メトリクスは不具合予測において用いられていたものを参考に選定した [8]．

クラスに関するメトリクスは，バージョン管理システム git を利用して計測する．今回の実験では，java で記述されたソフトウェアを対象としたため，各クラスは単一のファイルに実装されている．3 節の冒頭で説明した方法で選択された特定のリビジョンから過去の開発履歴を辿って，java クラスファイルの変更回数とその変更に関わったオーサーの数を計測した．

メソッドに関するメトリクスは，先ほど紹介した historage[11] を利用し，クラスに関するメトリクスの計測方法と同様にして，過去の開発履歴を辿り，各メソッドの変更回数とその変更に関わったオーサーの数を計測した．

3.2　対象ソフトウェア

オープンソースプロジェクトである jEdit[12] を対象に調査した．記述言語は java，リビジョン数は全部で 7,707 である．開発の進行に応じていくつかのリビジョンを選択し，メトリクス計測対象とした．

今回，調査したリビジョンの情報を表 2 に示す．リビジョン進行度は，最新リビジョンを進行度 100%とした場合の，開始リビジョンからの進行度合いである．様々な開発時期のメソッド抽出事例を調査するため，進行度を 10 – 50%まで変動させたときの各リビジョンを選択した．メ

表 1　計測メトリクス一覧

カテゴリ	メトリクス名	計算式
クラスに関するもの	メソッドが属するクラスの変更回数	$\sum_{i=1}^{r} isChanged(C_{m_{r-i}})$
	メソッドが属するクラスの変更に関わったオーサー数	$\lvert\bigcup_{i \in changedRevisions(C_{m_r})} author_i\rvert$
メソッドに関するもの	メソッドの変更回数	$\sum_{i=1}^{r} isChanged(m_{r-i})$
	メソッドの変更に関わったオーサー数	$\lvert\bigcup_{i \in changedRevisions(m_r)} author_i\rvert$

表 2　調査対象リビジョン

リビジョン進行度	メソッド数	抽出有	抽出無
10%	4,107	5	4,102
20%	5,080	7	5,073
30%	3,760	7	3,753
40%	4,974	14	4,960
50%	5,649	10	5,639

ソッド数はそのリビジョンに含まれる全メソッドの数を，抽出有，抽出無はそれぞれ，全メソッド内のメソッド抽出対象メソッドと対象ではないメソッドの数を表す．

単一のリビジョンでは充分な数のメソッド抽出対象メソッドを得ることができなかったので，複数のリビジョンで得た抽出対象メソッドを合わせた計 43 個を今回の調査対象とした．

4. 調査結果と考察

調査対象メソッドの各プロセスメトリクスの分布を表した箱ひげ図を図 3 に示す．図中の抽出有，抽出無はそれぞれ，メソッド抽出対象メソッドと対象ではないメソッドを表す．

メソッド抽出対象メソッドと対象ではないメソッドの間で，各プロセスメトリクスに有意差があるか確認するため，マンホイットニーの U 検定を実施した．表 3 に各プロセスメトリクスの中央値，平均値，検定の結果得られた p 値を示す．

表 3 より，メソッドが属するクラスに関するメトリクスは，有意水準 0.05 で有意差があるという結果になった．メソッドに関するメトリクスは，いずれも有意差が確認できなかった．これらの結果より，今回対象としたソフトウェアにおいては，メソッド抽出の対象となるメソッドが属するクラスは，対象ではないメソッドが属するクラスと比べ，変更回数が多く，また，関わった開発者も多いということが判明した．

次に，図 3 より，各プロセスメトリクスの計測結果について考察する．クラスに関するメトリクスはいずれも有意差が確認できた．特にクラスの変更回数は，抽出の対象となるメソッドと対象ではないメソッドの間で顕著な差が確認できる．クラスの変更に関わったオーサー数は，クラスの変更回数ほど顕著な差ではなかった．抽出の対象ではないメソッドの属するクラスの変更に関わったオーサー数が，ほとんど 1 人であったため，有意差が確認できたと考えられる．

メソッドに関するメトリクスはいずれも有意差が確認できなかった．メソッドの変更回数は，中央値，平均値ともに抽出の対象となるメソッドの方が大きい値を示した．有意差が確認できなかったのは，抽出の対象ではないメソッド内において，かなり大きい値を示したいくつかの事例による影響が大きいと考えられる．今後，調査対象を増やして結果を再確認したい．メソッドの変更に関わったオーサー数は，メソッド抽出の対象となるメソッドと対象ではないメソッドのいずれも，ほとんど 1 人であった．クラスの変更に関わったオーサー数が，その変更回数に比べかなり少ないことを考えると，複数人が変更に関わるメソッドはほとんどないと考えられる．そのため，このメトリクスはメソッドの特徴の表現方法として不適切であると考える．

最後に，これらの調査結果の妥当性について述べる．まず，調査対象について述べる．本実験では，様々な開発時期のメソッドを調査するため，開発の進行度に応じた複数のリビジョンを選択した．それらからメソッド抽出の対象となるメソッドを 43 個取得した．抽出の対象ではないメソッドはランダムに 43 個選択し，計 86 個のメソッドを対象に実験を行った．そのため，今回選択されなかったリビジョンやメソッドを対象にすると異なる結果が得られる可能性がある．また，対象ソフトウェアは jEdit のみである．今後，調査対象を増やし，より詳細な実験をする必要があると考えられる．

次に，リファクタリング検出ツールであるが，本実験では，Kenja を利用してメソッド抽出リファクタリングを検出した．Kenja は高い適合率，再現率でメソッド抽出リファクタリングを検出するが，誤りや漏れは存在する．そのため，実際にメソッド抽出が行われたメソッドを全て正確に検出し，それらを用いて実験した場合には，今回と異なる結果が得られる可能性がある．

5. まとめ

本研究では，プロセスメトリクスを用いて，メソッド抽出リファクタリングが行われたメソッドの特徴を調査した．オープンソースソフトウェアである jEdit から，メソッド抽出の対象となったメソッドと対象とならなかったメソッドを取得し，それらメソッドのプロセスメトリクスを計測して，比較した．実験の結果，抽出の対象となったメソッドは，対象とならなかったメソッドと比べ，そのメソッド

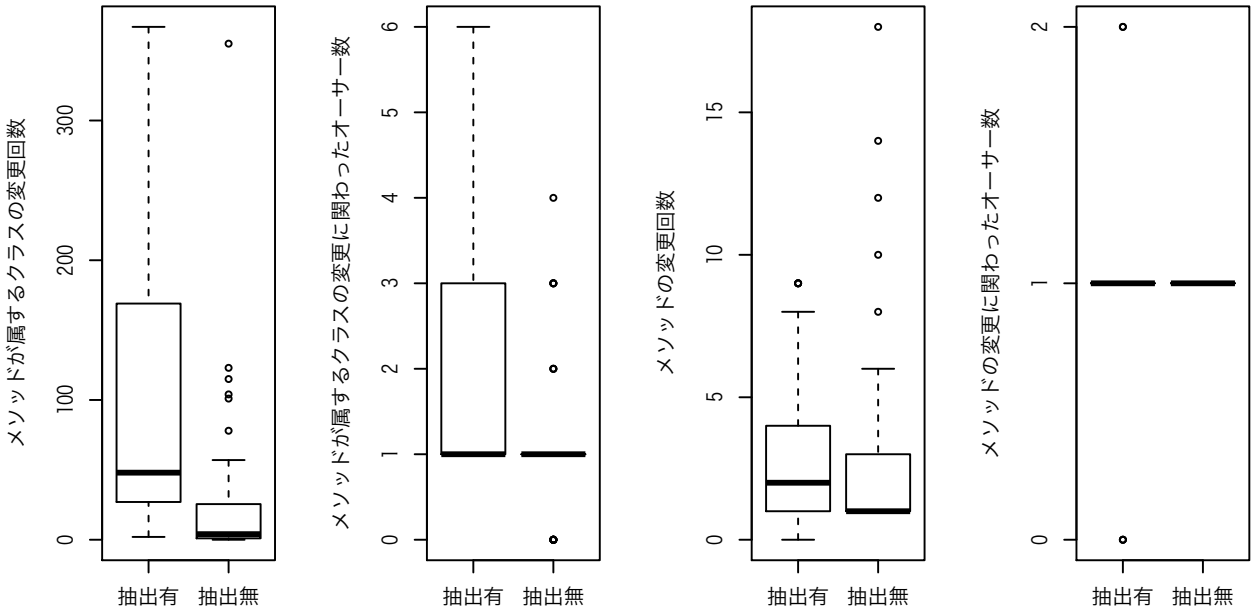

図 **3**　メソッドの各プロセスメトリクスの分布

表 **3**　プロセスメトリクスの比較結果

メトリクス	抽出の有無	中央値	平均値	p 値
メソッドが属するクラスの変更回数	有	48	122	$p < 0.05$
	無	4	28.090	
メソッドが属するクラスの変更に関わったオーサー数	有	1	1.884	$p < 0.05$
	無	1	1.116	
メソッドの変更回数	有	2	3.093	$0.05 < p$
	無	1	2.930	
メソッドの変更に関わったオーサー数	有	1	0.977	$0.05 < p$
	無	1	1	

が属するクラスの変更回数とその変更に関わったオーサーの数が多いことがわかった．メソッドの変更回数とその変更に関わったオーサーの数は，各メソッド群の間に有意差は確認できなかった．

今後の課題として，対象ソフトウェア，計測対象メソッド，プロセスメトリクスの数を増やした調査の実施が挙げられる．プロセスメトリクスとして，クラス・メソッドの存在期間，バグ修正回数などを考えている．また，これらの調査より，有効性が確認されたプロセスメトリクスを用いて，メソッド抽出の対象となるメソッドを推薦する手法の改善に取り組みたいと考えている．既存研究は，コード片のスナップショットから得られるメトリクスを利用している．手法の改善にあたって，プロセスメトリクスと既存のメトリクスのどちらを利用したほうが良いのか，もしくは両方とも利用する場合，どのようなバランスで両者のメトリクスを用いるのかなどの課題が考えられる．

謝辞　本研究は JSPS 科研費 JP26730036, JP16K16034, JP15H06344 の助成を受けたものです．

参考文献

[1] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).
[2] Fowler, M.: Refactoring, `http://refactoring.com/`.
[3] : JDeodorant, `http://www.jdeodorant.com/`.
[4] Silva, D., Terra, R. and Valente, M. T.: Recommending Automated Extract Method Refactorings, *Proc. of ICPC '14*, pp. 146–156 (2014).
[5] 後藤　祥，吉田則裕，藤原賢二，崔　恩瀞，井上克郎：機械学習を用いたメソッド抽出リファクタリングの推薦方法，情報処理学会論文誌，Vol. 56, No. 2, pp. 627–636 (2015).
[6] 今里文香：構文情報を用いた機械学習に基づくリファクタリング箇所の推薦–メソッド抽出リファクタリングへの適用–，修士論文，大阪大学大学院情報科学研究科 コンピュータサイエンス専攻 (平成 26 年).

[7] 後藤　祥，吉田則裕，藤原賢二，崔　恩瀞，井上克郎：メソッド抽出リファクタリングが行われるメソッドの特徴調査，コンピュータソフトウェア，Vol. 31, No. 3, pp. 318–324 (2014).
[8] 畑　秀明，水野　修，菊野　亨：不具合予測に関するメトリクスについての研究論文の系統的レビュー，コンピュータソフトウェア，Vol. 29, No. 1, pp. 106–117 (2012).
[9] Murphy-Hill, E., Parnin, C. and Black, A. P.: How We Refactor, and How We Know It, *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 5–18 (2012).
[10] 藤原賢二，吉田則裕，飯田　元：構文情報リポジトリを用いた細粒度リファクタリング検出手法，情報処理学会論文誌，Vol. 56, No. 12, pp. 2346–2357 (2015).
[11] Hata, H., Mizuno, O. and Kikuno, T.: Historage: Fine-grained Version Control System for Java, *Proc. of IWPSE-EVOL '11*, pp. 96–100 (2011).
[12] : JEdit, `http://www.jedit.org`.