Vol.2015-SE-187 No.10
2015/3/12

ソフトウェア進化におけるワークアイテムとコミット数に関する調査

三浦 圭裕[†1,a)]　福島 崇文[†1,b)]　亀井 靖高[†1,c)]　鵜林 尚靖[†1,d)]

概要：本稿では，ソフトウェア開発における同一の目的を持った作業群（ワークアイテム）について，ワークアイテム内のコミット数と作業目的の関係について調査する．マイニングソフトウェアリポジトリ分野における分析レベルとして，ファイル変更の単位（コミット）での分析は盛んに行われているが，ワークアイテムレベルでの分析はあまり行われていない．本稿では，ワークアイテムの理解を深めるためにワークアイテムの大きさに着目し，2 つのリサーチクエスチョンを設け，調査した．Apache，Mozilla から収集した 8 つのサブプロジェクトに対して，調査を行った結果，1) 大きさが 2 以上のワークアイテムは，全体のワークアイテムに対して，56.8%存在した，2-1) 大きさが上位のワークアイテムは大きさが下位のワークアイテムより，変更ファイル数の中央値が 2 倍大きい，2-2) 大きさが上位のワークアイテムは，大きさが下位のワークアイテムより re-open された件数が 3.57 倍多い，2-3) ワークアイテムの大きさとワークアイテムの重要度に関係はない, 2-4) 大きさが上位のワークアイテムは，大きさが下位のワークアイテムより連続コミットを含むワークアイテム数が 1.5 倍多い．といった知見が得られた．

キーワード：ワークアイテム，コミット，ソフトウェア進化，マイニングソフトウェアリポジトリ

1. はじめに

ソフトウェアは利用者を満足させるために，出荷後も故障に対するソフトウェアの修正作業や新たな機能の追加が行われている．これらの修正作業や機能追加（ソフトウェア進化 [8]）の履歴は，版管理システム（Git や Subversion）や課題管理システム（Bugzilla や Jira）などのリポジトリに保存されている．ソフトウェア進化の理解を目的として，リポジトリに管理されているデータに対する分析が行われてきた．例えば，文献 [1] では，版管理システム内のソースコードを解析し，オープンソースソフトウェアにおけるライセンスの種類やライセンス変化の頻度の傾向を分析している．

これまでのソフトウェア進化研究は版管理システムによって提供されるデータをそのままの形式で利用できるという分析の容易さから，分析の粒度としてコミットを採用するものが多かった [3][6][7]．しかしながら，コミットの仕方は開発者によって異なる傾向があり，その傾向の違いが分析結果に影響を与えることが考えられる [4]．McIntosh ら [4] は，コミットを同じ開発の目的を持つ作業群（ワークアイテム）としてまとめ，ビルドファイルとソースコードファイルの進化に関する分析を行っている．ワークアイテム単位で分析することで，コミット単位では見られなかった，ビルドファイルがソースコードと共に修正される傾向を発見できた．

ワークアイテム単位での分析を行うことで，コミット単位で発見できないソフトウェア進化の知見の発見が期待できる．本稿では従来研究で行われてきたソフトウェア進化に関する研究を今後ワークアイテム単位で行うための前段階として，ワークアイテムの理解を深めるための調査を行う．ワークアイテムの大きさ（ワークアイテム内のコミット数）に注目し，Apache と Mozilla のサブプロジェクトから計測した計 8 つのプロジェクトを対象に以下の 2 つのリサーチクエスチョン（RQ）に答える．

RQ1: ワークアイテムの大きさは，どのような分布であるか　ワークアイテムに関する全体的な理解を深めることを目的に，ワークアイテムの大きさがどのような分布であるのかを確かめる．2 コミット以上を含むワークアイテムがどの程度存在するのか，及び，他と比べて大きいワークアイテムはどの程度の大きさになるのかを調査する．

RQ2: 大きいワークアイテムは，どういった特徴である

†1 現在，九州大学
Presently with Kyushu Uniersity
a) miura@posl.ait.kyushu-u.ac.jp
b) f.taka@posl.ait.kyushu-u.ac.jp
c) kamei@ait.kyushu-u.ac.jp
d) ubayashi@ait.kyushu-u.ac.jp

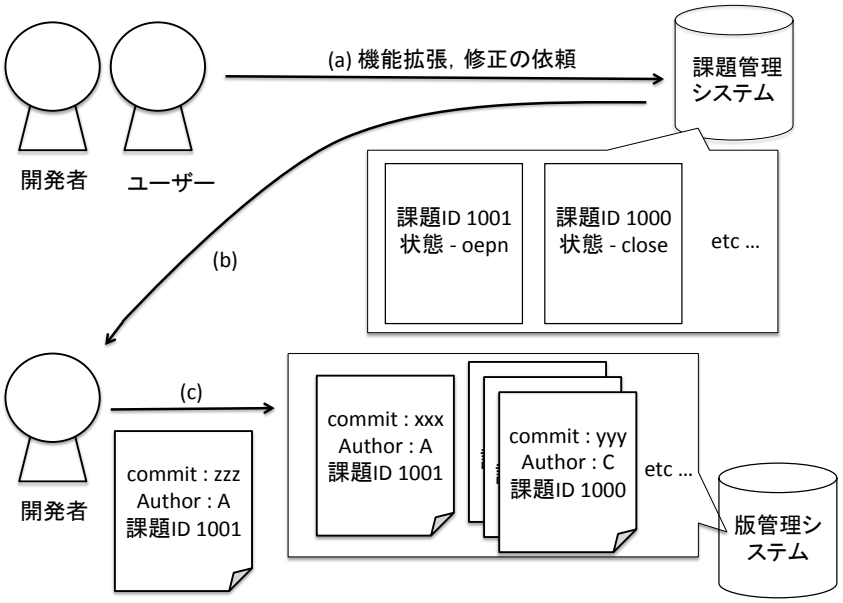


図 1 オープンソースソフトウェア開発の流れ

か 従来研究において大きいコミット（同時変更ファイル数の大きいコミット）の特徴について調査が行われてきた [2]．本 RQ では，大きいワークアイテムがどういった特徴であるかを，1) 変更ファイル数，2) re-open の有無，3) ワークアイテムの重要度，4) 連続コミットの有無の観点から調査する．

以降，2 章で背景と関連研究を示す．次に，3 章でデータセットと調査方法，及び，結果を述べ，4 章でまとめを述べる．

2. 背景と関連研究

2.1 オープンソースソフトウェアの進化のプロセス

本節では，本研究で想定するオープンソースソフトウェアの進化のプロセスを紹介する．ソフトウェアのリリース後，ユーザや開発者によって発見される障害や提案される追加機能は，発見者や提案者によって課題管理システム (Bugzilla や Jira）等に依頼・登録される．これら課題管理システムに登録された課題には，一意に特定するための課題 ID が割り振られる（図 1(a)）．

図 1(a) では，1000 という課題が既に登録され解決されており，新たに課題 1001 が登録されたとする．その後，当該チームのマネージャが，課題を担当する上で適任と思われる開発者（修正対象ファイルのオーナなど）に担当者として課題を割り振る（図 1(b)）．課題が割り振られた開発者は，当該課題のためのソースコード修正を行い，版管理システムへコミット（リポジトリへファイルを書き込む）する（図 1(c)）．コミットを行う際に，開発者はどのような変更を行ったのかを他の開発者が確認できるようにメッセージを付与でき，多くの場合，課題 ID もメッセージ中に含める．開発者により何度かファイルが修正され，課題が完了すると課題管理システム中の状態が完了を示す *close* に変更される．

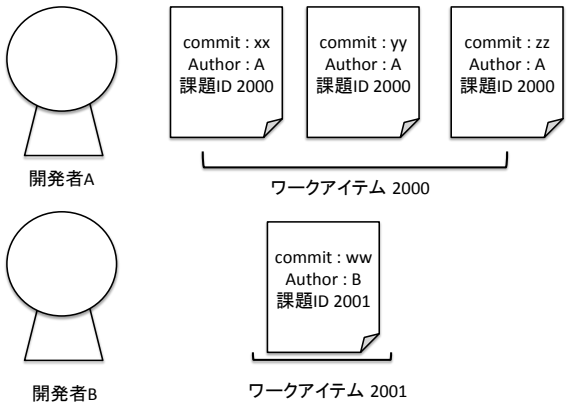


図 2 コミットとワークアイテムの関係

2.2 ワークアイテムとコミットの関係

開発者は課題を解決するために，ファイルを変更してリポジトリへコミットする．コミットはリポジトリにおける最小の作業単位を表す．しかしながら，このコミットの仕方は開発者によって異なることがある [4]．

例えば，ある課題に対して開発者 A は，解決するための作業をいくつかの行程に分割して，各工程ごとにコミットを行う（例えば，ソースコードファイルのみをコミットし，その後テストファイルを作成しコミットする）．その場合，1 つの課題に対して複数のコミットが行われる（図 2 上部）

表 1　データセット一覧

プロジェクト名	期間	コミット数	課題 ID を持つコミット数	課題 ID を持つコミット数の割合	ワークアイテム数
Apache/Ambari	2011/08/30 - 2015/01/09	10,627	10,344	0.97	7,791
Apache/Camel	2007/03/19 - 2015/01/13	24,918	15,242	0.61	6,164
Apache/Derby	2004/08/11 - 2015/01/14	10,698	8,901	0.83	3,581
Apache/Hbase	2007/04/03 - 2015/01/13	20,108	16,696	0.83	7,924
Apache/Hive	2008/09/02 - 2015/01/15	6,893	6,467	0.94	5,157
Mozilla/Addon-sdk	2009/09/14 - 2015/01/13	7,049	3,666	0.52	2,049
Mozilla/Bedrock	2010/12/15 - 2015/01/13	6,345	4,198	0.66	1,732
Mozilla/Gaia	2009/09/16 - 2015/01/09	51,469	37,825	0.73	16,046

一方，開発者 B は作業を分割せずに，作業が完了した際に一括してコミットを行う（図 2 下部）というような状況が考えられる．その場合，当該課題に対して行われるコミットは 1 回である．

これまでのソフトウェア進化研究は版管理システムによって提供されるデータをそのままの形式で利用できるという分析の容易さから，分析の粒度としてコミットを採用するものが多かった [3][6][7]．しかしながら，前段落の例のようなコミットの仕方の差異は，ソフトウェア進化の研究に一定の影響を与えることが報告されている [4]．その一方でコミットを同じ開発の目的を持つ作業群（ワークアイテム）としてまとめ，ワークアイテム単位で分析することで，コミットの仕方の差異による影響を小さくすることが可能である．例えば，図 2 の場合，ワークアイテム単位で分析すると，両者の場合も 1 つの作業として扱うことが可能である．

2.3　関連研究

本研究に関連して，ソフトウェア進化に関してコミット単位での分析が行われている [2][6]．Zaidman らは，テストと関連するプロダクトコード（ソースコード）がどのように進化するのかに注目し，2 つのオープンソースソフトウェアプロジェクトに対してテストコードと関連するプロダクトコードが同時に開発されているのかを調査した [6]．調査の方法として，修正されているテストコードとプロダクトコードを時系列順にソートし，進化の過程を可視化した．結果として，テストコードと関連するソースコードは，同時に修正されていることを示唆した．

マイニングソフトウェアリポジトリ分野において，巨大なコミット（多数の変更ファイルを持つコミット）はノイズの原因となる可能性があることから分析対象外とされる場合が多くある．Abram ら [2] は，巨大なコミットがノイズであるのか否かを明らかにすることを目的として，巨大なコミットを手作業によって作業内容ごとに分類した．9 つのオープンソースプロジェクトに対して行ったケーススタディの結果，巨大なコミットの作業内容はブランチのマージ，ソフトウェアに対する機能追加，ドキュメントの修正が多い傾向を示した．

これらの研究ではコミットを分析の単位としている一方，本研究ではワークアイテム単位を分析の単位としている点が異なる．

McIntosh ら [4] は，ソフトウェア開発におけるビルドファイルに対するオーバーヘッドがどの程度であるのかを定量的に評価することを目的として，プロダクトファイル，テストファイル，ビルドファイルの変更回数を分析した．予備実験として行われたコミット単位の分析ではプロダクトファイルとビルドファイルの同時修正の関係は見られなかった．McIntosh らは，さらにワークアイテム単位の分析を行うことで，ビルドファイルがソースコードと共に修正される傾向を発見した．

本研究ではワークアイテム単位の分析をさらにこれまでのソフトウェア進化に提供するため，ワークアイテムの理解を進めることを目的としている．

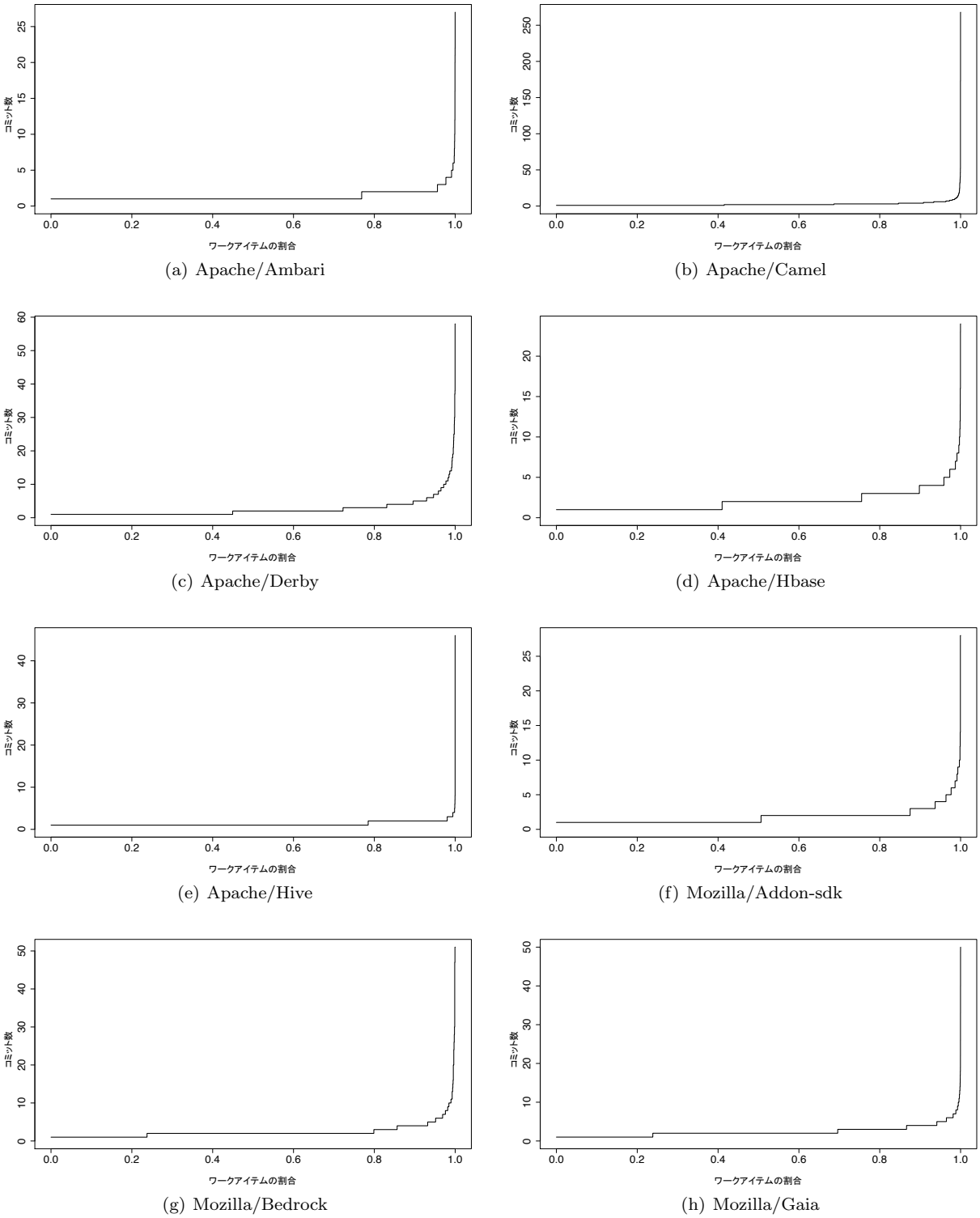
3.　調査

3.1　データセット

我々は，Git[*1] リポジトリの Apache と Mozilla のサブプロジェクトの内，5,000 コミット以上のサブプロジェクトに着目し，その内 23 つのサブプロジェクトを収集した．本稿では，ワークアイテムとコミット数に関する調査を行うため，課題 ID を有するコミットが多く存在するサブプロジェクトを選ぶ必要がある．そのため，それぞれのサブプロジェクトに対して，課題 ID を有するコミットが全体コミットの 5 割以上存在するサブプロジェクトを調査対象とする（表 1）．

3.2　ワークアイテムの特定方法

本研究では，同じ開発の目的を持つ作業単位であるワークアイテムの特定方法として，コミット中に含まれるメッセージに着目する．開発者は課題管理システムに登録された課題（欠陥の修正や機能追加）に対するソースコードの修正をコミットする際，一般的に課題管理システムの課題 ID をメッセージ中に記述する [5]．



(a) Apache/Ambari
(b) Apache/Camel
(c) Apache/Derby
(d) Apache/Hbase
(e) Apache/Hive
(f) Mozilla/Addon-sdk
(g) Mozilla/Bedrock
(h) Mozilla/Gaia

図 3 ワークアイテムの大きさの分布

本研究ではメッセージ中に含まれる課題 ID が同一のコミットをワークアイテムとして特定した．

3.3 RQ1: ワークアイテムの大きさは，どのような分布であるか

調査方法．各サブプロジェクトのコミット履歴からワークアイテムにどの程度コミットされているかを調査した．各サブプロジェクトのコミット履歴から課題 ID を抽出し，課題 ID ごとのコミット数を調べた．その結果を用いて，ワークアイテムの大きさの分布を求めた．

調査結果．各サブプロジェクトのワークアイテムの大きさの分布を図 3(a)-(h) に示す．図 3(a)-(h) は，横軸をワークアイテムの大きさでソートした全ワークアイテムに対するワークアイテムの割合，縦軸をワークアイテム内

表 2 全ワークアイテムに対する大きさが 2 以上のものの割合

サブプロジェクト名	割合
Apache/Ambari	23.1%
Apache/Camel	58.5%
Apache/Derby	55.1%
Apache/Hbase	59.0%
Apache/Hive	21.5%
Mozilla/Addon-sdk	49.4%
Mozilla/Bedrock	76.2%
Mozilla/Gaia	76.1%
中央値	56.8%

コミット数を表した分布である．調査の結果，大きさが 2 以上のワークアイテムの全体に対する割合は，最小で 21.5%（Apache/Hive），最大で 76.2%（Mozilla/Bedrock）であり，中央値が 56.8%であった（表 2）．ワークアイテムの半数が 2 つ以上のコミットで構成されていることから，コミット単位で実施されている分析をワークアイテムで分析することで新たな知見が得られることが期待できる．

大きさが 2 以上のワークアイテムは，全体のワークアイテムに対して 56.8%存在した．

3.4 RQ2: 大きいワークアイテムは，どういった特徴であるか

ワークアイテムの大きい上位 10%と下位 10%に対して，特徴ごとに比較を行う．2 コミット以上で構成されているワークアイテムに対して，ワークアイテムの大きい上位 10%と下位 10%に分割した．それぞれのワークアイテム数は 5,044 であり，上位 10%に含まれるコミット数は 27,092，下位 10%は 10,088 であった．

3.4.1 RQ2-1 変更ファイル数

調査方法. ワークアイテムの大きい上位 10%と下位 10%に対して，各ワークアイテムの変更ファイル数を求めて比較する．

調査結果. 得られた上位と下位のワークアイテムの変更されたファイル数の中央値，平均値，最大値を表 3 に示す．上位のワークアイテムは，下位のワークアイテムより変更されるファイル数が大きいことがわかった．

ここで，変更ファイル数が大きいワークアイテムの特徴をより詳細に理解するために，上位のワークアイテムの変更ファイル数が最も大きいワークアイテムに注目した．そのワークアイテムは，Apache/Camel の課題 ID 3676[*2] であった．この課題では，全ソースコードのコメント中に含まれる$Revision$タグを取り除く事を目的として，11 回のコミットで 3,414 ファイルが変更されていた．同一目的の作業が複数回に分けてコミットされていることから，こういったコミット群に対してはコミット単位の分析を行うよりもワークアイテム単位の分析を行うことが有用であると考える．

[*2] https://issues.apache.org/jira/browse/CAMEL-3676

表 3 変更ファイル数

	各ワークアイテムの変更されたファイル数		
	中央値	平均値	最大値
上位	4	18.1	3414
下位	2	6.99	693

上位のワークアイテムは下位のワークアイテムより，変更ファイル数の中央値が 2 倍大きい．

3.4.2 RQ2-2 連続コミット

調査方法. 各サブプロジェクトのコミット履歴から各ワークアイテムに連続コミットがどの程度あるか否かを調査した．本稿では，連続コミットを 10 分と 60 分の閾値を設け，閾値以内に同じ開発者で，かつ，同じ課題 ID を含むコミットとした．

調査結果. 結果を表 4 に示す．上位のワークアイテムは，下位のワークアイテムよりも連続コミットを含むワークアイテム数が両方の閾値において，1.5 倍多かった．連続コミットは，コミットのし忘れや軽微な変更の可能性があるので，連続コミットをワークアイテム単位で一つの作業としてまとめることは有用であると考える．

例えば，zaidman らはプロダクトファイルと関連するテストファイルが同時に開発されているかどうかを調査した [6]．しかしながら，zaidman らの調査では，同一のコミットでプロダクトファイルと関連するテストファイルが変更されているかを観測しているため，任意のプロダクトファイルをコミットした後，直ちに関連するテストファイルをコミットする連続コミットを考慮する事ができない．そのため，ワークアイテム単位の分析で zaidman らの調査を追調査することにより，コミット単位の分析で観測されなかったプロダクトファイルと関連するテストファイルの同時開発が観測可能と考える．

上位のワークアイテムは，下位のワークアイテムより連続コミットを含むワークアイテム数が 1.5 倍多い．

3.4.3 RQ2-3 リオープンの有無

調査方法. 2 コミット以上で構成されているワークアイテムに対して，構成されるコミット数の多い上位と下位のワークアイテムのリオープンされている件数を調査した．まず，Apache，Mozilla の課題管理システムから，上位と下位のワークアイテムの履歴を取得した．次に，取得した履歴から上位と下位の各ワークアイテムがリオープンされたか否かを確認した．

調査結果. 結果を表 5 に示す．上位のワークアイテムは，下位のワークアイテムよりリオープンされる件数において 3.57 倍上回っている．上位のワークアイテムの特徴として

表 4 連続コミット件数

	ワークアイテム数	10 分以内の連続コミットを含むワークアイテム数	1 時間以内の連続コミットを含むワークアイテム数
上位	5,044	1,635	2,120
下位	5,044	1,081	1,404

表 5 リオープンされた件数の割合

	ワークアイテム数	リオープンされた件数	リオープンされた割合 (%)
上位	5,044	1,411	27.80
下位	5,044	395	7.77

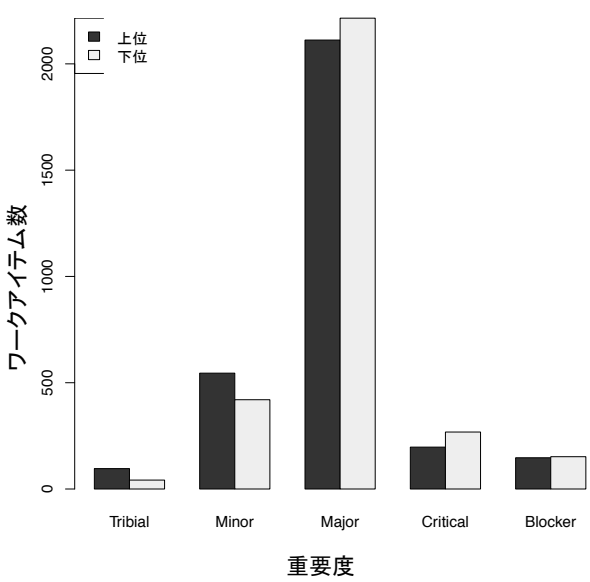


図 4 Apache ワークアイテムの重要度

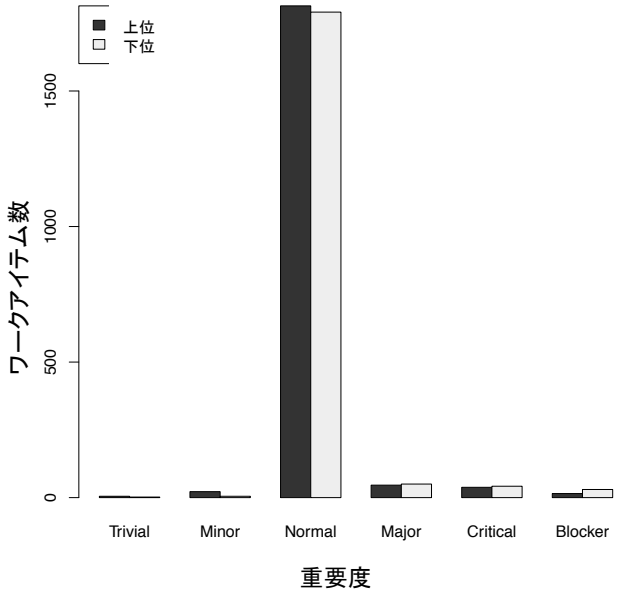


図 5 Mozilla ワークアイテムの重要度

は，1) 下位のワークアイテムよりも課題の内容が難しく修正後に再度取り組まれている可能性，2) 1 回目の close の判定時にテストが十分に行われおらずに，再度欠陥が見つけられた可能性，といったことが考えられる．

> 上位のワークアイテムは，下位のワークアイテムよりリオープン件数が 3.57 倍多い．

3.4.4 RQ2-4 ワークアイテムの重要度

調査方法． RQ2-2 で収集したページ情報から，Apache，Mozilla のプロジェクトごとに上位と下位のワークアイテムの重要度を調査した．

調査結果． 各プロジェクトごとの結果を図 4，図 5 を示す．図 4，図 5 において，上位と下位の間に大きな差は見られなかった．そのため，ワークアイテムの大きさとワークアイテムの重要度は，関係がないことがわかった．

> ワークアイテムの大きさとワークアイテムの重要度に関係がない．

4. まとめ

本稿では，Apache，Mozilla のサブプロジェクトの変更履歴データからワークアイテムの理解を深めるため，2 つの RQ を設け，調査した．8 つのサブプロジェクトのコミット履歴と課題管理システムのページ情報を用いて，2 つの RQ を行った結果，以下の知見を得た．

- RQ1 では，大きさが 2 以上のワークアイテムは，全体のワークアイテムに対して，56.8%存在した．
- RQ2-1 では，上位のワークアイテムは下位のワークアイテムより，変更ファイル数の中央値が 2 倍大きい．
- RQ2-2 では，上位のワークアイテムは，下位のワークアイテムより re-open された件数が 3.57 倍多い．
- RQ2-3 では，ワークアイテムの大きさとワークアイテムの重要度に関係はない．
- RQ2-4 では，上位のワークアイテムは，下位のワークアイテムより連続コミットを含むワークアイテム数が 1.5 倍多い．

今後の課題として，ワークアイテムの作業内容の種類とワークアイテムの大きさの関係についての調査やソフト

ウェア進化のコミットレベル分析に対するワークアイテムレベル分析の追調査が挙げられる．

謝辞 本研究の一部は，日本学術振興会 科学研究費補助金（若手 A：課題番号 24680003，挑戦的萌芽：課題番号 25540026）による助成を受けた．

参考文献

[1] Massimiliano Di Penta, Daniel M. German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the evolution of software licensing. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pp. 145–154, New York, NY, USA, 2010. ACM.

[2] Abram Hindle, Daniel M German, and Ric Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pp. 99–108. ACM, 2008.

[3] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pp. 485–495, Washington, DC, USA, 2009. IEEE Computer Society.

[4] Shane McIntosh, Bram Adams, Thanh HD Nguyen, Yasutaka Kamei, and Ahmed E Hassan. An empirical study of build maintenance effort. In *Proceedings of the 33rd international conference on software engineering*, pp. 141–150. ACM, 2011.

[5] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proc. Int'l Working Conf. on Mining Software Repositories (MSR'05)*, pp. 1–5, 2005.

[6] Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie Van Deursen. Mining software repositories to study co-evolution of production & test code. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pp. 220–229. IEEE, 2008.

[7] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pp. 563–572, Washington, DC, USA, 2004. IEEE Computer Society.

[8] 大森隆行, 丸山勝久, 林晋平, 沢田篤史. ソフトウェア進化研究の分類と動向. コンピュータ ソフトウェア, Vol. 29, No. 3, pp. 3–28, 2012.