OSS プロジェクトにおける Tangled コミットの実証分析

三浦 圭裕， 亀井　靖高，鵜林　尚靖，佐藤　亮介

リポジトリマイニング分野において，ソフトウェア開発履歴が蓄積されたリポジトリを対象にした分析が行われている．リポジトリには，開発者により変更された作業内容（バグ修正・機能追加・リファクタリングなど）が，コミットとして管理されている．コミットは理想的には 1 つの作業を扱っているまとまった単位であり，従来のリポジトリマイニング研究はその前提にしたがって研究を進めている．しかしながら，この前提が必ずしも正しくないことは，従来研究によって調査がされており，複数の作業内容を行なっているコミット（Tangled コミット）の存在が指摘されている．本稿では，正規表現によって検出される Tangled コミットの候補のうち，どの程度が実際には Tangled コミットではないのか，について調査する．Apache ソフトウェアに登録されている 14 プロジェクトから正規表現によって候補となった 2,405 件の Tangled コミットを対象に目視により分類した結果，55.47%が実際には Tangled コミットではないことがわかった．

1　はじめに

ソフトウェア進化分野の研究において，ソフトウェア開発に関する知見を得ることを目的として，ソフトウェア開発履歴が蓄積されたリポジトリを対象に分析が行われている．例えば，Di Penta ら [2] はオープンソースソフトウェアのライセンス変更の調査を行うために，版管理システムに管理されているソースコードの分析を行った．

ソフトウェア進化に関する研究では，いくつかの分析粒度が用いられる．例えば，版管理システムにおける最小単位であるコミット（修正されたファイルの集合）単位 [8][10][13] や，コミット群をリポジトリに統合するためのリクエスト集合であるプルリクエスト単位 [5][4]，ステークホルダーに対するソフトウェアの公開集合であるリリース単位 [1][3][11] が存在する．コミットは版管理システムから比較的容易に抽出可能であるため [6]，多くの研究者によってコミット単位での分析が行われている [9][14][15]．

我々の先行研究 [12] では，ソフトウェア進化分野において Work item（同一作業を目的としたコミット群）に着目し，コミットが必ずしも分析する上で適切ではない可能性を示した．ファイルの co-change に関する研究（あるファイルが変更された際，同時に変更するべきファイルを推薦する分野） [13][15] では，1 コミットを 1 作業単位としており，コミット内で変更されているファイル群に関連があるという前提がある．そのため，複数のコミットにまたがって 1 つの作業がなされると関連が発見できない．我々は Work item がどの程度影響を与えるかを理解するための調査を行い，Work item は今後の co-change 分野の研究において考慮されるべきと結論づけている．

本稿では，コミットに複数の作業内容が行われているコミット（Tangled コミット） [7] に着目する．コミットは理想的には 1 つの作業を扱っているまとまった単位であり，従来のリポジトリマイニング研究ではその前提に従って研究が進められている．研究成果

An Empirical Study of Tangled Commits in OSS Projects

Keisuke Miura, 九州大学大学院システム情報科学府, Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University.

Naoyasu Ubayashi, Yasutaka Kamei, Ryosuke Sato, 九州大学大学院システム情報科学研究院, Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University.

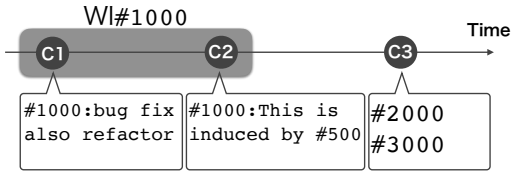

図 1　Work item と Tangled コミットの関係図

の妥当性を損なう可能性がある Tangled コミットを判別して，取除く必要がある．本稿では，正規表現によって検出される Tangled コミットの候補のうち，Apache ソフトウェアに登録されている 14 プロジェクトの 2,405 コミットを対象に Tangled コミットであるかを手作業での分類を行なった．

以降，2 章で本研究の位置づけを述べる．3 章では，実験に用いるデータと分類結果・考察を述べる．最後に 4 章でまとめと今後の課題を述べる．

2　背景と本研究の位置付け

この章では，Work item と Tangled コミットの関連性と本稿での定義を図 1 として述べる．

ソフトウェアの変更履歴を管理している版管理システム，バグや機能追加などの作業進捗を管理している課題管理システムがある．課題管理システムでは，それぞれのバグ修正や機能追加の作業目的ごとに課題番号（例えば，#1000 #2000）が振られ管理している．開発者はバグ修正を行い，その修正を適用するため版管理システムにコミットを行う．コミットする際，開発者はどのような変更を行ったかを記録するためにメッセージに開発者が行った課題番号を多くの場合記す．

2.1　Tanglde コミットと Work item

Tangled コミットは，複数の作業を一度に行ったコミットである．Herzig らの分類方法に従うと，Tangled コミットはコミットメッセージ中に記された課題番号だけではなく，自然言語で記された作業も含む．具体的に図 1 では，変更 *C1* は課題番号#1000 とメッセージ中の refactor の 2 つの作業を行っているため，Tangled コミットである．*C2* は#1000 のみの作業であるため非 Tangled コミットであり，*C3* は#2000 と #3000 の 2 つの作業を行っているため，Tangled コミットである．

Work item は同じ作業を行っているコミット集合であり，コミットメッセージに記されている課題番号を基にコミットをまとめられている．図 1 では，*C1* と *C2* は同じ作業#1000 に対する変更であるため，一つの変更 *WI#1000* とする．しかし，図 *1* の例では，*Tanglde* コミットにより以下の問題が生じる．(*1*) *C1* は#1000 の他に *refactor* を行った *Tangled* コミットであるため，*WI#1000* は#1000 と *refactor* の *2* つの作業をした集合となっている．(*2*) *C2* は自然言語処理を行っていないため，コミットメッセージ中の課題番号#500 を誤検出し，#500 と#1000 の *Tangled* コミットとしてしまう．

本稿では，*Tangled* コミットの初期的な判別のため問題（*2*）に着目して，自然言語処理を行わずに複数の課題番号記されたコミットを *Tangled* コミットと定義し，手作業で分類を行った．

3　ケーススタディ

3.1　データセット

我々は *Apache* ソフトウェアに登録されている *14* プロジェクトの全コミットである *130,608* コミットを対象に，複数の課題番号がコミットメッセージに記されたコミットを正規表現を用いた抽出した（*2,405* コミット）．*2,405* コミットに対して，我々は *Tangled* コミットであるかを目視によって分類した．各プロジェクトデータの概要を表 *1* に示す．

3.2　アプローチ

Tangled コミットの候補（*2,405* コミット）それぞれに対して，各課題番号の作業内容を課題管理システムを通して確認する．課題管理システムに記載された作業内容が明らかに異なっている場合，当該コミットを *Tangled* コミットとして分類する．

3.3　分類結果

プロジェクトごとの分類結果を表 *1* に示す．*2,405* コミット中，*1,334* コミット（*55.47%*）が実際に *Tan-*

表 1　データセット一覧

プロジェクト	期間	総コミット数	分析対象コミット数	Tangled でないコミット数（%）
Accumulo	2011/10/04 - 2015/08/29	5,317	231	121 (52.39)
Ambari	2011/08/30 - 2015/08/29	12,713	33	20 (60.61)
Camel	2007/03/19 - 2015/08/30	20,812	187	19 (10.16)
Cassandra	2009/03/02 - 2015/08/28	11,886	42	29 (69.05)
Cayenne	2007/01/21 - 2015/08/12	4,419	32	15 (46.87)
Derby	2004/08/11 - 2015/08/26	8,049	424	282 (66.51)
Hbase	2007/04/03 - 2015/08/29	10,879	399	312 (78.20)
Hive	2008/09/02 - 2015/08/29	6,718	191	191 (100)
Jackrabbit	2004/09/13 - 2015/08/22	8,080	209	114 (54.55)
Karaf	2007/11/26 - 2015/08/28	4,980	54	13 (24.07)
OpenJPA	2006/03/02 - 2015/07/21	4,752	59	21 (35.59)
Qpid	2006/09/14 - 2015/08/28	14,047	278	92 (33.09)
Sling	2007/09/09 - 2015/08/28	13,703	224	78 (34.82)
Thrift	2008/03/16 - 2015/08/30	4,253	42	27 (64.29)
ALL	-	130,608	2,405	1334 (55.47)

gled コミットでないことがわかった．

目視での分類において，正規表現では *Tangled* コミットとして候補と検出されたが，実際には *Tangled* コミットでなかったものの代表的な例を *3* つ示す．まず，コミットメッセージにおいて，ある 1 つの課題を説明するために別の課題番号を用いている例である．例えば，*Accmulo* のコミット †1 では，*“ACCUMULO-2926 Fix the multiple slf4j bindings that ACCUMULO-2808 introduced”* と述べている．*ACCUMULO-2808* †2 の変更で発生した *ACCMULO-2926* †3 の修正のみを行なっていると判断し，*Tangled* コミットではないとした．

次に，課題管理システム上の課題内容がコミットメッセージに含まれる場合である．*Hbase* のコミット †4 では，*“HBASE-1671 HBASE-1609 broke scanners riding across splits”* と述べられている．課題管理システム上において *HBASE-1671* †5 の課題内容は，*“HBASE-1609 broke scanners riding across splits”* と示されている．そのため，*HBASE-1671* †6 では，*HBASE-1609* †7 によって発生した単一の作業を処理していると判断した．

最後に，ファイル名に課題番号を含む例である．*DERBY* のコミット †8 では，*“DERBY-3624: Missing deadlock in storetests/st_derby715.java [*中略*]”* と述べている．正規表現で `st_derby715.java` が課題番号として抽出されたが，ファイル名であるため *DERBY-3624* †9 のみの修正と判断した．

4　まとめ

本稿では，正規表現によって検出される *Tangled* コミットの候補のうち，*Apache* プロジェクトに登録されている *14* プロジェクトの *2,405* コミットに対して

†1 https://github.com/apache/accumulo/commit/d9ee274dcf9b6ce18e2cf7d6fa8304cf516df154
†2 https://issues.apache.org/jira/browse/ACCUMULO-2808
†3 https://issues.apache.org/jira/browse/ACCMULO-2926
†4 https://github.com/apache/hbase/commit/a68b5ee409b927074e482bd60808018b589a1b53
†5 https://issues.apache.org/jira/browse/HBASE-1671
†6 https://issues.apache.org/jira/browse/HBASE-1671
†7 https://issues.apache.org/jira/browse/HBASE-1609
†8 https://github.com/apache/derby/commit/4f29da3d8f359314f70347182034f322caa2e45d
†9 https://issues.apache.org/jira/browse/DERBY-3624

手作業での分類を行なった．

今後の研究として，今回対象とした正規表現によって検出される *Tangled* コミットの候補だけでなく，自然言語処理を用いた全コミットを対象とした *Tangled* コミットの検出を考えている．

参 考 文 献

[1] *Bettenburg, N., Shang, W., Ibrahim, W. M., Adams, B., Zou, Y., and Hassan, A. E.: An Empirical Study on Inconsistent Changes to Code Clones at the Release Level, Sci. Comput. Program., Vol. 77, No. 6(2012), pp. 760–776.*

[2] *Di Penta, M., German, D. M., Guéhéneuc, Y.-G., and Antoniol, G.: An Exploratory Study of the Evolution of Software Licensing, Proc. Int'l Conf. on Software Engineering (ICSE'10), 2010, pp. 145–154.*

[3] *Godfrey, M. W. and Tu, Q.: Evolution in Open Source Software: A Case Study, Proc. Int'l Conf. on Software Maintenance (ICSM'00), 2000, pp. 131–142.*

[4] *Gousios, G., Pinzger, M., and Deursen, A. v.: An Exploratory Study of the Pull-based Software Development Model, Proc. Int'l Conf. on Software Engineering (ICSE'14), 2014, pp. 345–355.*

[5] *Gousios, G. and Zaidman, A.: A dataset for pull-based development research, Proc. Working Conf. on Mining Software Repositories (MSR'14), 2014, pp. 368–371.*

[6] *Hata, H., Mizuno, O., and Kikuno, T.: Bug Prediction Based on Fine-grained Module Histories, Proc. Int'l Conf. on Software Engineering (ICSE'12), 2012, pp. 200–210.*

[7] *Herzig, K., Just, S., and Zeller, A.: The impact of tangled code changes on defect prediction models, Empirical Software Engineering, Vol. 21, No. 2(2016), pp. 303–336.*

[8] *Hindle, A., German, D. M., and Holt, R.: What do large commits tell us?: a taxonomical study of large commits, Proc. Working Conf. on Mining Software Repositories (MSR'08), 2008, pp. 99–108.*

[9] *Juergens, E., Deissenboeck, F., Hummel, B., and Wagner, S.: Do Code Clones Matter?, Proc. Int'l Conf. on Software Engineering (ICSE'09), 2009, pp. 485–495.*

[10] *Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., and Ubayashi, N.: A large-scale empirical study of Just-in-Time quality assurance, IEEE Trans. Software Engineering, Vol. 39, No. 6(2013), pp. 757–773.*

[11] *McIntosh, S., Adams, B., and Hassan, A. E.: The Evolution of Java Build Systems, Empirical Software Engineering, Vol. 17, No. 4-5(2012), pp. 578–608.*

[12] *Miura, K., Mcintosh, S., Kamei, Y., Hassan, A. E., and Ubayashi, N.: The Impact of Task Granularity on Co-evolution Analyses, Proc. Int'l Conf. on Empirical Software Engineering and Measurement (EMSE'16), 2016, pp. 47:1–47:10.*

[13] *Ying, A. T. T., Murphy, G. C., Ng, R., and Chu-Carroll, M. C.: Predicting Source Code Changes by Mining Change History, IEEE Trans. Software Engineering, Vol. 30, No. 9(2004), pp. 574–586.*

[14] *Zaidman, A., Van Rompaey, B., Demeyer, S., and Van Deursen, A.: Mining software repositories to study co-evolution of production & test code, Proc. Int'l Conf. on Software Testing, Verification, and Validation (ICST'08), 2008, pp. 220–229.*

[15] *Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A.: Mining Version Histories to Guide Software Changes, Proc. Int'l Conf. on Software Engineering (ICSE'04), 2004, pp. 563–572.*