修士学位論文

題目

繰り返し出力が変更されるログ文における管理状況の調査

指導教員
肥後 芳樹　教授

報告者
柏原　大地

令和 5 年 2 月 1 日

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 ソフトウェア工学講座

令和 4 年度 修士学位論文

繰り返し出力が変更されるログ文における管理状況の調査

柏原　大地

内容梗概

ログはソフトウェア実行中の様々な動作や情報を記録するため広く使われており，ソフトウェア開発において重要な役割を果たしている．ログを解析することで，プログラムにおける実行時情報や通信の頻度や量，プログラムの内部的振舞いを知ることができる．

ログ出力の有無はソースコードでのログ文のレベル（ログレベル）のみならず，設定ファイルにおける出力の閾値設定にも影響される．そしてログ文ではそのレベル，書き方や挿入箇所，設定ファイルでは出力先および閾値設定を決める必要がある．そのため，その複雑さからログが適切に出力されないケースも数多くみられる．ログが適切に出力されない状態では，システムの状態を正確に把握できなくなったり，バグやエラーのような期待しない挙動を行った際の原因究明が困難になることが予想され，開発者・ユーザ双方にとって損失である．Qiu は，ログレベルと設定ファイルの閾値の両方を考慮した解析を行い，同一のログ出力文に対して繰り返し出力の有無が変更されるケースが存在することを明らかにしたこのことから，ログ出力を決定する開発者の側でも，どのようにログを設定すればよいか，適切に実装することが難しいことがうかがえる．

そこで本研究では，繰り返し出力が変更されたログ文に着目し，その管理状況を調査する．具体的には，あるソフトウェアの開発履歴において同一のログ出力文に対して複数回ログ出力の有無が変更された場合に，再変更までの期間や変更をおこなった開発者，そしてコミットログでの言及状況を調査することで，開発者が繰り返し出力が変更されるログ文をどのようにとらえているのかを明らかにする．調査の結果，開発者は明示的に変更を示すことが少なく，コミットメッセージでの情報共有が少ないことが分かった．

主な用語

ロギング
ログレベル
ソフトウェア保守

目 次

1　まえがき

ログはソフトウェア実行中の様々な動作や情報を記録するため広く使われており，ソフトウェア開発において重要な役割を果たしている．例えば，マイクロソフトのあるシステムにおいては，1 日に数十テラバイトのログを生成している [1]．ロギング，すなわちログを取ることは，ソフトウェアの保守や運用において，実行時情報を取得するのに非常に重要である．ログを解析することで，プログラムにおける実行時情報や通信の頻度や量，プログラムの内部的振舞いを知ることができる．また，システム障害などの問題が発生した際には，起こった出来事の経過を把握したり，原因の調査や解決法の検討などの助けになる．障害分析のみならず，ログはパフォーマンス調査，セキュリティなど多岐にわたって活用されており，現代のソフトウェアに取って必要不可欠な存在である．

ログ出力の有無はソースコードでのログ文のレベル（ログレベル）のみならず，設定ファイルにおける出力の閾値設定にも影響される．そしてログ文ではそのレベル，書き方や挿入箇所，設定ファイルでは出力先および閾値設定を決める必要がある．そのため，その複雑さからログが適切に出力されないケースも数多くみられる．ログが適切に出力されない状態では，システムの状態を正確に把握できなくなったり，バグやエラーのような期待しない挙動を行った際の原因究明が困難になることが予想され，開発者・ユーザ双方にとって損失である．Qiu らは，ログレベルと設定ファイルの閾値の両方を考慮した解析を行い，同一のログ出力文に対して繰り返し出力の有無が変更されるケースが存在することを明らかにした [2]．このことから，ログ出力を決定する開発者の側でも，どのようにログを設定すればよいか，適切に実装することが難しいことがうかがえる．

そこで本研究では，繰り返し出力が変更されたログ文に着目し，その管理状況を調査する．具体的には，あるソフトウェアの開発履歴において同一のログ出力文に対して複数回ログ出力の有無が変更された場合に，再変更までの期間や変更をおこなった開発者，そしてコミットログでの言及状況を調査することで，開発者が繰り返し出力が変更されるログ文をどのようにとらえているのかを明らかにする．この調査結果により，繰り返し変更が行われるログに関する特徴を抽出し，ロギング設定を変更する際に注意すべき点や将来の再修正につながる不吉な匂いを明らかにするための知見が得られる．

以下，2 章では本研究の背景および関連研究について述べる．3 章では実施する調査方法について記述し，4 章ではその結果について記述する．5 章で妥当性への脅威について記述し，最後に 6 章でまとめについて述べる．

2　背景

2.1　ロギング

ロギングは，プログラムの実行時情報の収集や記録を目的としてよく行われる．開発者はソースコード内にロギング文を挿入することで，挿入した地点における変数の状態や，エラーメッセージなど，多岐にわたる情報を出力可能である．Li らが行った開発者への調査では，ロギングはトラブルシューティング，追跡，理解，記録など多岐にわたって用いられていることが明らかになっている [3]．また，Yuan らの調査によると，ソースコード内に平均して 30 行につき 1 行のロギングコードが含まれている [4]．このように，ソフトウェア開発においてロギングは一般的に行われ，開発者はロギングに得られた情報を用いることで開発を効率的に進めることが可能となっている．

一方で，ロギングの実施においては，その出力ログに関するコストやロギング文の保守コストが大きな問題となっている．

出力ログに関するコストについては，例えば，マイクロソフトのあるシステムにおいては，一日に数十テラバイトのログが出力されており，その処理には膨大な時間を要するため実環境において受け入れがたいコストとなっている [1]．また，Li らの開発者らに対する調査では，過大なロギングは大量のログの管理・処理コストの増大や実行時のパフォーマンスなどに影響を与え，ソフトウェア保守コストの増大につながることが明らかになっている [3]．

また，保守コストの観点においては，ロギング文の変更も頻繁に行われていることが調査に複数の調査によって明らかになっている．Yuan らの調査ではロギング文はソースコード全体と比較して約 1.8 倍の割合が変更が行われていることが分かっている [4]．Kabina らが調査した 4 つのオープンソースソフトウェアでは，ロギングステートメントの 20%から 45%において変更が行われていることが判明している [5]．

このように，ロギングはソフトウェア開発において頻繁に行われる一方で，出力に関するコストやロギング文の保守コストは無視できないものとなっている．一方で，これらの保守によって得られるロギングによる恩恵も大きいため，ロギングの実施コストや保守コストを削減することは重要な課題の一つである．

2.2　ロギング設定

実行時に出力されるログの量を制御する方法として，多くのプロジェクトでロギングライブラリが用いられている．ロギングライブラリを用いることで開発者は，それぞれのロギングステートメントに対してその重要度に応じたログレベルを設定し，設定ファイルにてどのログレベルまで出力するかの閾値を指定することで，目的に応じたログを収集することが出来ている．例えば，ロギングライブラリの Apache log4j2 [6] では，重要度が高い順に $fatal$,

$error$，$warn$，$info$，$debug$，$trace$ の 6 つのログレベルが設定されている．開発者はソフトウェアのロギングステートメントに対してそれぞれこの 6 つのレベルを指定し，通常実行時は $warn$ レベル以上のログを出力し，テスト実行時は $debug$ レベル以上のログを出力といったように閾値設定をすることで，目的に応じたログを収集することが可能となる．この閾値設定については，ロギングの設定ファイル内で設定可能である．

2.3 ロギング設定に関する既存研究

適切なログレベルの設定は困難で，一般的に多くのコストがかかるとされている．Hassani らはログに関する問題の調査を行った [7]．その結果，ログレベルに関する問題は “不適切なログメッセージ”，“ロギング文が書かれていない” に次いで 3 番目に多い問題で，全体の 17.0%を占めることが明らかになっている．また，ログに関する問題は表面化して報告されるまでに，中央値で 320 日程度と長期間を要することも明らかにされている．一方でそのうち 80%はその報告後 10 日以内に解決される．例えば，Hadoop ではログレベルが $info$ で出力されていたロギング文が，ログの情報量の少なさから $debug$ へとログレベルが下げられている．この例は発見までに 103 日かかったが，問題の報告後は 5 日で修正されている．また，問題の解決については，78%が元のコードのコミッタ以外によるものであり，様々な開発者がロギング文の修正に携わっていることも明らかになっている．

また，ログレベルに対する修正内容も多岐にわたり，開発者は適切なレベルの設定に苦労していることが明らかになっている．Yuan らによると，72%のログレベルの修正においてエラーイベントの重要性に対する開発者の判断の変更が反映されていることが分かっている [4]．また，重要なログか，エラーに関するログかのレベルの変更について，開発者はログ出力のトレードオフを考慮しながら，適切なログレベルの設定を行っていることが分かる．

さらにログ設定においてプロジェクト毎に共通の設定がないことも明らかになっている．Li らの調査によると，プロジェクトによってロギングライブラリを用いたロギングの特性は異なる [8]．特にデバッグの目的に用いるログレベルがプロジェクトによって異なり，同じログメッセージでもプロジェクトによって異なるログレベルが設定されている場合があった．

このように，ソフトウェア開発においてログレベルの適切な設定は非常に困難となっている．一方で，ソフトウェアの開発においてログからもたらされる情報は有用であるため，開発者は有用なログの内容とその出力のトレードオフを考慮しながら開発を行っている．

2.4 ロギング設定支援

2.4.1 ソースコードのログレベル設定支援

ロギング設定の支援を目的として，これまでに機械学習や深層学習を用いたソースコードログレベルの決定の支援が行われている．Li らはソフトウェアメトリクスに基づいた回帰モデルを作成することで，ソースコードにおける適切なログレベル決定の支援を行っている [8]．Anu らは開発者のロギングの意図に応じてログレベルの推薦を行うモデルを提案した [9]．具体的には，ロギングの静的テキストや変数，例外の型，コードコメントなどの文脈情報に着目することで，開発者の意図を反映している．Li らはログレベルの順序を考慮したニューラルネットワークを用いたログレベルの提案を行う手法を提案した [10]．手法においてはログ文が配置されたソースコード上の構造情報とそのメッセージの内容をもとに学習を行っている．さらに隣り合うログレベルの順序情報を考慮することで，予測が失敗した際にも正解のログレベルに近い推薦が可能となっている．Liu らは複数のレベルのコードブロック情報の利用によるログレベルの提案を行っている [11]．コードブロック内においては AST 情報を構造情報として用い，コードブロック間については制御依存グラフを用いることで依存関係を抽出している．これらを組み合わせてモデルを作成することで，ログレベルの提案を実施している．

2.4.2 設定ファイルに関するロギング設定の研究

Zhi らは 10 のオープンソースソフトウェアと 10 の企業のソフトウェアに対して，ロギング設定に関する分析を行なった [12]．分析の中で，ロギング設定の閾値に関する変更の大部分は異なる閾値への調整である．これは，開発者が適切な閾値に設定することへ苦心していることを表している．

2.5 ログ出力に関する研究

Ding らが提案した Adaptive logging [13] は，最終的にログをフィルタすることでロギングのコストを軽減している．Mizouchi らは実行時に設定ファイルの出力閾値を切り替えることで，プログラムが異常な挙動を示した際のログを詳細に取得するツール PADLA を提案した [14]．いずれの研究もロギングにおける多大な出力がソフトウェアのパフォーマンスにおける影響に着目している．このようにロギング設定が定まった後に，ログを出力すべきかしないべきかといった観点からロギングの出力量に着目した研究もこれまでに行われている．

Qiu らは，ログレベルの分析において，ログレベルと設定ファイルの閾値の両方を考慮した解析を行った [2]．そこでは，ログレベルや設定ファイルの変更が必ずしも最終的な出力には影響しないことや，設定ファイルの変更によってソースコードが変更されていないにもか

かわらずログ出力の有無が変更される例を発見したことが報告されている．本研究は Qiu らの分析手法にもとづき，繰り返しログ出力の有無が変更されるような変更履歴を調査する．

表 1: 調査プロジェクトの概要

名称	コミット数	スター数	開発者数
activemq	11,316	2,125	126

3 調査方法

本調査ではロガーの閾値情報とソースコードにあるログステートメントを紐づけ，その出力の有無の変化を時間とともに追う．出力に変更が見られた場合，それに対応するコミットを抽出する．さらに，出力の変化回数を記録し，複数回変更が見られた際，追加で変更があった時点のコミットを抽出し，再変更までの時間やコミットした開発者などを分析することでより詳しい調査を行う．これらの調査により，ログステートメントまたは閾値を変えたことにより，出力が繰り返し変更となるような状況が開発履歴上でどのように管理されているかを調査する．

3.1 調査対象

調査対象として，Qiu らの研究対象と同じ Apache ActiveMQ を対象とした．これはプロジェクトが成熟している，およびロギングユーティリティが log4j2 という基準で選ばれている．成熟したプロジェクトはコミット数が多いため活発な開発が行われており，繰り返しロギング設定が変更されている可能性が高い．また ActiveMQ は実際にそのような事例が多く存在することが Qiu の研究によって明らかになっている．また，log4j2 は広く用いられており，将来的に別のプロジェクトを調査する際に調査手法を適用しやすい利点がある．プロジェクトの概要は表 1 に示す通りである．本研究では，2023 年 1 月時点での最新コミット（タグが打たれたバージョン）までの情報を使用した．

3.2 分析のための RQ

開発履歴の中で，明示的に繰り返し変更された出力はどのように扱われているかを明らかにするために，調査の分析は，以下の RQ に順に答えることで進める．

- RQ1: 繰り返し出力が変更されるログ文の，ある変更から次の変更までの期間はどのようになっているか
- RQ2: 繰り返し出力を変更したのは同じ開発者によるものか
- RQ3: 開発履歴の中で，繰り返し変更された出力は明示的に記録されているか

それぞれの RQ について，その調査目的を説明する．

3.2.1 RQ1: 繰り返し出力が変更されるログ文の，ある変更から次の変更までの期間はどのようになっているか

本 RQ ではプロジェクトにおいてロギング設定による出力の変更がプロジェクトに直接的な影響を与えるかに着目して，変更期間の分布に着目した調査を行う．具体的には，再変更までの期間が短ければ，設定変更によるログ出力量の増加や逆に保守作業に必要なログが出力されないといった変更の保守作業への直接的な影響の発生により，ロギング設定が繰り返し変更されたと考える．一方で，再変更までの期間が長ければ，上記のような理由ではなく，それまで変更されていなかった閾値に再び変更を加えるべき事由が生じたと推測できる．例えば，ログ文を含むモジュールの重要性の変化といった，ログ文自体の影響を受けた変化が考えられる．これらを分類するために．閾値が変更されたコミットのコミット日時から，ある変更から次の変更までの期間を調査する．

3.2.2 RQ2: 繰り返し出力を変更したのは同じ開発者によるものか

本 RQ ではログ文の保守作業のうち出力の変更という特に大きな影響を与える保守作業において，複数の開発者が携わっているかを調査する．具体的には同一の開発者が繰り返し変更を行っているのか，別々の開発者が異なる意図をもって同一のログ文に変更を行っているのかを明らかにすることがこの RQ の目的である閾値が変更されたコミットのコミット作成者を取得し，あるログ文についてすべての変更が同一の開発者によるものであるか否かを判定する．

3.2.3 RQ3: 開発履歴の中で，繰り返し変更された出力は明示的に記録されているか

本 RQ では繰り返し変更された出力がコミットメッセージとして明示的に記録されているかを分析することで，出力の変更を伴うロギング設定の変更が管理されているかを分析する．ログ出力の閾値の変更が明示的に記録されていない場合，のちの開発者がその事実に気づくことは難しくなる．一方，もしコミットメッセージにその理由が明示されていれば，再変更の理由を類推することが可能になる．そこで，繰り返し出力が変更された際のコミットメッセージを以下の 3 つに分類する．

- (a) 言及がない
- (b) 変更した事実にのみ言及している
- (c) 再変更したことについても言及している

3.3 調査手順

まず，ActiveMQ から繰り返し出力が変更されたコミットの特定を行う．特定においては，ソースコードのログレベルと設定ファイルにおける出力閾値レベルに着目する．コミット履歴において，どちらかのレベルが変更された際にその出力が変更されたかを調査することで出力の変更は特定可能である．具体的には，出力閾値レベルに対してソースコードのログレベルが同じかそれ以上である状態から出力レベルよりも低くなった場合，あるいはその逆の変更が出力の有無の変更に影響する．

本調査では先行研究である Qiu のスクリプトを用いることでコミットの特定処理を実施した．Qiu のスクリプトでは Pydriller [15] が用いられており，コミットからのソースコードの変更情報の抽出を行っている．Qiu のスクリプトを実行し，繰り返しログ出力の有無が変更されたログ文とその変更系列を抽出した．その結果，合計 19 個の系列が調査の対象となった．

本節では，このデータを対象として以下の RQ に示した内容に対する調査手順について述べる．

3.3.1 RQ1 における調査手順

繰り返し変更される出力の期間について，閾値が変更されたコミットのコミット日時から，ある変更から次の変更までの期間を調査する．変更情報を記録する際，日時情報を付け加えて，一緒に保存すると，変更情報の履歴を出力するときに，各系列の変更時間を取得することができる．これらを日時情報関数に入力することで，日時の差が得られる．また，別々の変更系列に重複するコミット，すなわちコミットのハッシュ値が一致するコミットが存在する．これは閾値の調整により，同じモジュールにあるログの出力が全部影響されることに起因する．閾値ファイルの修正による影響を調査するため，重複するコミットを取り除いたデータも同様な分析を行った．

3.3.2 RQ2 における調査手順

繰り返し変更される出力の開発者の分析のために，閾値が変更されたコミットのコミット作成者を取得し，あるログ文についてすべての変更が同一の開発者によるものであるか否かを判定する．RQ1 と同様に，変更情報を記録する際に作者情報を付け加えることで，変更系列の各コミットの作者の情報を得られる．各変更系列に対して，まったく同じ開発者が繰り返し変更を行ったのか，そうでない場合，合計何人が修正に加わったのかについて統計を行う．

3.3.3 RQ3 における調査手順

前処理として抽出した繰り返し出力が変更されたコミットに対して，コミットメッセージを以下の 3 つに分類する．

- (a) 言及がない
- (b) 変更した事実にのみ言及している
- (c) 再変更したことについても言及している

以下それぞれの項目について，どのような基準に基づいて分類したかの簡単な概要を述べる．

(a) 言及がない

issue でレベルについての言及がなく，ログ，閾値，ログレベル+メッセージ（例：warn メッセージ）も全く触れていない．

(b) 変更した事実にのみ言及している

issue でレベルについての言及があり，もしくはログ，閾値，ログレベル+メッセージ（例：warn メッセージ）について触れているもののうち，巻き戻し操作やリセットについて触れていない

(c) 再変更したことについても言及している

巻き戻し操作やリセットについて言及があるもの

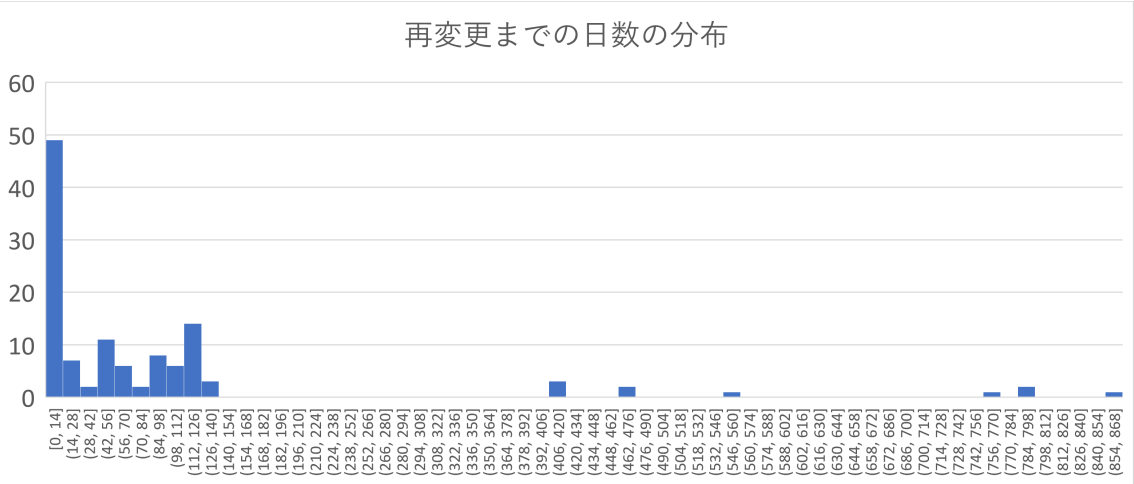

図 1: 再変更までの日数の分布

4 調査結果

4.1 RQ1: 繰り返し出力が変更されるログ文の，ある変更から次の変更までの期間はどのようになっているか

RQ1 の分析結果は図 1 および図 2 が示されている通りである．図 1 の縦軸は変更があったコミット数，横軸は経過時間を示している．データ数は 188，平均値は 93.5 日である．グラフは左側，すなわち経過時間が少ない側に寄っていることがわかる．実際，二週間以内の変更は 40%，126 日以内の変更は 90%以上を占めている．これは一度出力が変更されたログ文が，短期間で出力が再変更される可能性が高いことを示している．また，実際のログ文は 19 個よりずっと多いので，出力が変更されないログ文に関しては，ずっと変更されない可能性が高いということも意味している．そして図 2 の縦軸は重複を除いたコミット数で，横軸は同じく経過時間を示している．この場合のデータ数は 45 で，平均は 132 日である，データが半分以上減っていることから一回のコミットで複数の修正を行う傾向があることがわかる．また，重複するコミットは系列にわたって存在していることや，それらのコミットが閾値の修正にかかわっていることから，閾値変更による出力変化はログレベルのより影響が広いことを伺える．

4.2 RQ2: 繰り返し出力を変更したのは同じ開発者によるものか

RQ2 の結果は表 2 にまとめた．同じ開発者によるものは 3 個しかなく，すべての変更系列の長さは 2 であった．一方で，変更系列の長さは 1 であっても，異なる開発者によるものが 3 個存在している．また，半分以上の変更系列は 5 人の開発者が関わっている．これはロ

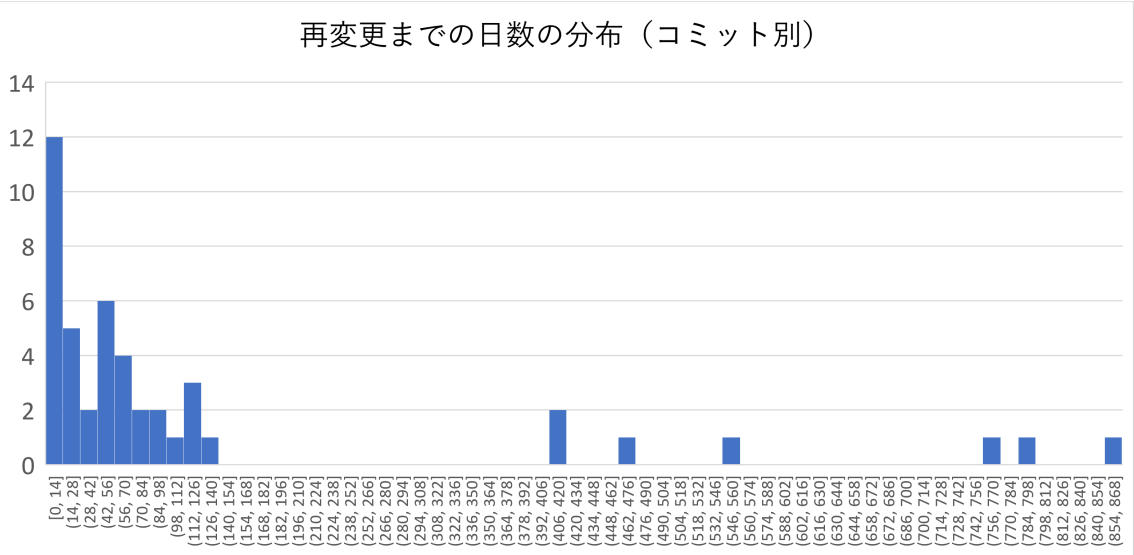

図 2: 再変更までの日数の分布（コミット別）

表 2: コミットメッセージの概要

変更系列にわたっての開発者数	変更系列の数
1	3
2	4
3	1
4	0
5	10
6	1

グ関連の修正は変更系列の長さに関係なく，個人ではなく、チームで行っていることを示している．

4.3 RQ3: 開発履歴の中で，繰り返し変更された出力は明示的に記録されているか

RQ3 の結果は表 3 にまとめた．全体を通して出力に関連することに言及があるものはわずか 33%であった．そして言及があるもののうち，出力の繰り返し変更について触れることのあるコミットはさらに 3 割しかなかった．そして，それらのコミットメッセージはそれぞれ「エラー分析で一時閾値を下げた」,「ログに関するポリシー変更」,「テストケースの差し戻し」,「ログレベルのリセット」となっていて，開発者はログ出力のどの閾値に変更を明示的に記録していない，あったとしてもそうするモチベーションを述べていないことがわかる．

また，実際に出力が繰り返して変更があったにもかかわらず，多くの場合開発者がそれに

表 3: コミットメッセージの概要

ログについての言及	コミット数
言及なし	30
変更の説明のみ：言及あり	11
コミットの差し戻しもしくは再変更：言及あり	4

ついて言及がないのは変更で増えた/減ったメッセージについて関心がないということを伺える．そして言及があったものは，増えた/減ったことに自覚があるものの，それ以上を書く必要はないと思い，詳細のメッセージを残さなかったと推測できる．

5　妥当性の脅威

本章では，本研究の妥当性の脅威について述べる．

外的妥当性として，調査対象を Apache ActiveMQ に限定したことによって，調査結果の一般性が十分に主張できない点にある．ただし，Apache ActiveMQ はコミット数が 1 万を超え，開発者数も 126 人と十分に成熟したプロジェクトであると考えられるため，ある程度の一般性は主張できる．

構成概念妥当性として，本研究の分析対象がコミットメッセージだけである点も妥当性の脅威として挙げられる．実際の開発においては，Issue やプルリクエストといったコミットに含まれる以外の情報を用いて，問題の記録やドキュメンテーションが行われている可能性がある．今後の課題として，プロジェクトにおけるロギング設定についての議論が行われているプラットフォームを特定し，追加調査を行う必要性がある．

6　まとめ

本調査では．ログステートメントまたは閾値を変えたことにより，出力が繰り返し変更となるような状況が開発履歴上でどのように管理されているかを調査した．そのためにまずロガーの閾値情報とソースコードにあるログステートメントを紐づけ，その出力の有無の変化を時間とともに追い．変更があった際はそれに対応するコミットを抽出した．さらに，出力の変化回数を記録し，複数回変更が見られた際，追加で変更があった時点のコミットを抽出した．そしてコミット時間，コードの編集者，コミットメッセージの三つの軸から，「繰り返し出力が変更されるログ文の，ある変更から次の変更までの期間はどのようになっているか」,「繰り返し出力を変更したのは同じ開発者によるものか」,「開発履歴の中で，繰り返し変更された出力は明示的に記録されているか」と 3 つの RQ を立てて，調査した．調査の結果，一つ目の RQ からは一度出力が変更されたログ文が短期間で出力が再変更される可能性が高いこと，二つ目の RQ からはログ関連の修正は変更系列の長さに関係なく，個人ではなく、チームで行っていること，三つ目の RQ からは多くの場合開発者は変更で増えた/減ったメッセージについて関心がないことが分かった．

謝辞

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 肥後芳樹 教授には，研究活動において貴重な御指導及び御助言を賜りました．肥後 教授に心より深く感謝いたします．

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 松下誠 准教授には，研究活動において貴重な御指導及び御助言を賜りました．松下 准教授に心より深く感謝いたします．

本研究の全過程にわたって，研究に関する相談に乗っていただき，特に，研究の方向性の検討，分析における問題点の提示など数多くの御指導および御助言を賜りました．奈良先端科学技術大学院大学 先端科学技術研究科 情報科学領域　嶋利一真 助教にならび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 神田哲也 助教に心より感謝いたします．

最後に，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肥後研究室の皆様には，多くの御指導及び御助言を頂きました．本論文を完成させることができたことは，肥後研究室の皆様の御協力のおかげであると感じております．肥後研究室の皆さまに，心より感謝いたします．

参考文献

[1] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, October 2018.

[2] Liang Qiu. Impact of logging configuration changes in source code and project configuration file. Bachelor Thesis, Department of Information and Computer Sciences, School of Engineering Science, Osaka University, February 2022.

[3] Heng Li, Weiyi Shang, Bram Adams, Mohammed Sayagh, and Ahmed E. Hassan. A Qualitative Study of the Benefits and Costs of Logging From Developers' Perspectives. *IEEE Transactions on Software Engineering*, Vol. 47, No. 12, pp. 2858–2873, February 2021.

[4] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. Characterizing logging practices in open-source software. In *Proceedings of the 34th International Conference on Software Engineering*, pp. 102–112, June 2012.

[5] Suhas Kabinna, Weiyi Shang, Cor-Paul Bezemer, and Ahmed E. Hassan. Examining the Stability of Logging Statements. In *Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, Vol. 1, pp. 326–337, March 2016.

[6] Log4j - Apache Log4j 2. `https://logging.apache.org/log4j/2.x/index.html`.

[7] Mehran Hassani, Weiyi Shang, Emad Shihab, and Nikolaos Tsantalis. Studying and detecting log-related issues. *Empirical Software Engineering*, Vol. 23, No. 6, pp. 3248–3280, 2018.

[8] Heng Li, Weiyi Shang, and Ahmed E. Hassan. Which log level should developers choose for a new logging statement? *Empirical Software Engineering*, Vol. 22, No. 4, pp. 1684–1716, August 2017.

[9] Han Anu, Jie Chen, Wenchang Shi, Jianwei Hou, Bin Liang, and Bo Qin. An Approach to Recommendation of Verbosity Log Levels Based on Logging Intention. In

Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution, pp. 125–134, September 2019.

[10] Zhenhao Li, Heng Li, Tse-Hsun Chen, and Weiyi Shang. DeepLV: Suggesting Log Levels Using Ordinal Based Neural Networks. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering*, pp. 1461–1472, May 2021.

[11] Jiahao Liu, Jun Zeng, Xiang Wang, Kaihang Ji, and Zhenkai Liang. TeLL: Log level suggestions via modeling multi-level code block information. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 27–38, July 2022.

[12] Chen Zhi, Jianwei Yin, Shuiguang Deng, Maoxin Ye, Min Fu, and Tao Xie. An Exploratory Study of Logging Configuration Practice in Java. In *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution*, pp. 459–469, September 2019.

[13] Rui Ding, Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Qingwei Lin, Qiang Fu, Dongmei Zhang, and Tao Xie. Log2: A Cost-Aware Logging Mechanism for Performance Diagnosis. In *Proceedings of the 2015 USENIX Annual Technical Conference*, July 2015.

[14] Tsuyoshi Mizouchi, Kazumasa Shimari, Takashi Ishio, and Katsuro Inoue. PADLA: A Dynamic Log Level Adapter Using Online Phase Detection. In *Proceedings of the 2019 IEEE/ACM 27th International Conference on Program Comprehension*, pp. 135–138, May 2019.

[15] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 908–911, October 2018.