ソフトウェア再利用時における
ライセンス違反検出技術に関する研究

2011年7月

真鍋　雄貴

ソフトウェア再利用時における
ライセンス違反検出技術に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2011年7月

真鍋　雄貴

論文一覧

主要論文

[1-1] 真鍋雄貴, ダニエル モラレス ゲルマン, 井上克郎: “階層的ライセンス知識を用いたライセンス特定ツールの開発”, 情報処理学会論文誌, Vol. 53, No. 8, 2011 年 8 月 [採録決定]（学術論文）

[1-2] Daniel M. German, Yuki Manabe, and Katsuro Inoue: “A Sentence-Matching Method for Automatic License Identification of Source Code Files”. In Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, pp. 437–446, 2010 年 9 月（国際会議録）

[1-3] Yuki Manabe, Yasuhiro Hayase, Katsuro Inoue: “Evolutional Analysis of Licenses in FOSS”, Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), pp. 83–87, 2010 年 9 月（国際会議録）

[1-4] Akito Monden, Satoshi Okahara, Yuki Manabe, Ken-ichi Matsumoto: “Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations”, IEEE Software, Special Issue on Software Protection, vol. 28, no. 2, pp. 42–47, 2011 年 3 月（学術論文）

関連論文

[2-1] Yu Kashima, Yasuhiro Hayase, Norihiro Yoshida, Yuki Manabe, Katsuro Inoue: “A Preliminary Study on Impact of Software Licenses on Copy-and-Paste Reuse”,The 2nd International Workshop on Empirical Software Engineering in Practice, 2010 年 12 月（国際会議録）

[2-2] 真鍋雄貴, Daniel M. German, 井上克郎: “ライセンス知識に基づくライセンス特定ツールの設計”, ウィンターワークショップ 2010・イン・倉敷 論文集, 情報処理学会シンポジウムシリーズ, vol.2010, No.3, pp. 13–14, 2010 年 1 月（国内会議録）

[2-3] 真鍋雄貴, 市井誠, 早瀬康裕, 松下誠, 井上克郎: “コメント中の頻出文字列を用いたソフトウェアライセンスの特定支援”, ソフトウェア工学の基礎 XIV, 日本ソフトウェア科学会 FOSE2007, pp. 105–114, 2007 年 11 月 (国内会議録)

内容梗概

ソフトウェア開発コスト削減のために，ソフトウェアの再利用が行われている．ソフトウェアの再利用とは，既存のソフトウェアに含まれるソフトウェア部品を同一，もしくは異なるソフトウェアの開発に利用することである．

近年では，オープンソースソフトウェアを用いたソフトウェア再利用が容易となっている．オープンソースソフトウェアが盛んに開発されており，ソフトウェア部品の大きな供給源となっている．また，ソフトウェア部品を蓄積し，それらに対する検索手段を提供するソフトウェア部品検索システムにより，開発者が必要とするソフトウェア部品を容易に発見し，そのソフトウェア部品のソースファイルを取得することができる．

一方，オープンソースソフトウェアの再利用が行いやすくなるに従い，ソフトウェアライセンス違反が問題となっている．ソフトウェアライセンスとは，著作物を利用するための許諾と，その許諾を得るための義務である．本稿で想定するソフトウェアライセンス違反とは，あるソースファイルをあるソフトウェアに再利用した際，ソースファイルのライセンスとソフトウェアのライセンスが矛盾することである．

ソフトウェアライセンス違反を予防するためには，再利用したいソースファイルのライセンスが特定されていること，開発中のソフトウェアに想定していないソフトウェアの再利用が行われていないか確認することが重要である．前者に対応する既存研究としてソフトウェアライセンス特定手法，後者に対応する既存研究としてコードクローン検出手法，剽窃検出手法がある．

本研究では，これらのソフトウェアライセンス違反予防策を踏まえ，ソフトウェアライセンス違反を検出することを目指す．ソフトウェアライセンス違反検出を行うに当たり，以下の問題点がある．

問題点 1 通常，ソースファイルのライセンスはソースファイル中のコメントで記述されるが（これをライセンス記述と呼ぶ），ライセンス記述の表記ゆれにより既知のライセンス記述との単純な照合だけではライセンスが特定できない．

問題点 2 あるソフトウェアにおいてライセンスは混在しているのか，またそれらは変更される可能性があるのかということが明らかでないため，ライセンスの特定はどの程度の頻度で行う必要があるかわからない．

問題点 3 あるソースファイル間で再利用が行われていると判断するための明確な基準が存在しない．

本研究では，これらの問題点を解決することを目的とし，以下の課題に取り組んだ．

問題点 1 に対し，オープンソースソフトウェアから抽出した知識に基づいたライセンス特定手法を提案する．使用する知識は，複数の大規模オープンソースソフトウェアから抽

出し，ライセンス記述の現状を反映している．評価として，既存のライセンス特定手法と回答率，正答率，所要時間で比較を行い，提案手法が既存の手法より優れていることを示した．

問題点２に対し，オープンソースソフトウェアにおけるライセンスの分布を調査した結果を示す．調査結果から，調べた対象のオープンソースソフトウェアには様々なライセンスのファイルが含まれているだけでなく，その比率がソフトウェアの進化に沿って大規模に変化することが分かった．また，オペレーティングシステムと非オペレーティングシステムでは進化の傾向が異なっていることが分かった．

問題点３に対し，コードクローンメトリクスに基づくソースコード再利用判定閾値の決定手法を提案する．本論文では，3 つのコードクローンメトリクス（コードクローン検出数，最大コードクローン長，部分類似度）を用い，これらのコードクローンメトリクスの値がどの程度であれば２ソフトウェア間でソースコードの再利用が行われていると判断できるか，もしくは，ないと判断できるか，その基準となる閾値をそれぞれ算出する．50 件のオープンソースソフトウェアを用いた実験を行った結果，使用した各コードクローンメトリクスに対して，再利用が行われたとみなせる下限値と，再利用が行われていないとみなせる上限値を算出できた．これらの閾値はソフトウェア間での再利用の有無を判断する値であるが，ソースファイルを対象とした場合も同様の手段で再利用の有無を判断するための閾値を得ることができると考えている．

謝辞

本研究の全般に関し，常日頃より適切なご指導を賜わりました，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 井上 克郎 教授に，心から深く感謝申し上げます.

本論文を執筆するにあたり，適切なご助言とご指導を頂きました，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 増澤利光 教授，同 楠本真二 教授に心から感謝いたします.

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在籍中に，適切なご助言とご指導を頂きました，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 萩原兼一 教授，同 八木康史 教授 に感謝いたします.

本研究を行うに当たり，直接具体的なご指導を頂きました，奈良先端科学技術大学院大学情報科学研究科コンピュータ科学領域 松本 健一教授，ヴィクトリア大学コンピュータサイエンス学部 Daniel M. German 准教授，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，奈良先端科学技術大学院大学情報科学研究科コンピュータ科学領域 門田 暁人准教授，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 石尾 隆 助教，筑波大学 システム情報工学研究科 早瀬 康裕 助教に心より御礼申し上げます.

大阪大学大学院情報科学研究科在学中，様々なご指導，ご協力を頂き，また，様々な相談に乗っていただいた，市井 誠氏に心より御礼申し上げます.

奈良先端科学技術大学院大学情報科学研究科にて，研究を通して，ご指導，ご協力頂きました，山内 寛己氏，岡原 聖氏に対し，心より御礼申し上げます.

最後に，井上研究室の皆様のご助言, ご協力に御礼申し上げます.

目 次

図 目 次

表 目 次

第1章　はじめに

ソフトウェア開発コストの削減を目的とし，ソフトウェアの再利用が行われている．ソフトウェアの再利用とは，既存のソフトウェア部品を同じシステム，もしくは新しいシステムで利用することである [19]．ソフトウェア部品とは，ソフトウェアの構成単位であり，クラス，関数，ソースコードなどが含まれる．

特に，近年では，オープンソースソフトウェアを用いたソフトウェア再利用が行いやすくなっている．オープンソースソフトウェアが活発に開発されており，ソフトウェア部品の大きな供給源だと考えることができる．オープンソースソフトウェアは SourceForge.net[11] や Google Code[8] などのサイトにて開発が進められている．また，ソフトウェア部品を検索するためのシステムとして，ソフトウェア部品を蓄積し，それらに対して検索手段を提供するソフトウェア部品検索システムがあり [1, 9]，開発者が求めるソフトウェア部品を選択し，取得することが容易になっている．

しかし，オープンソースソフトウェアの再利用が行いやすくなった一方で，ソフトウェアライセンス違反となる事例が多く発生している [13, 14, 16]．

そこで，本研究では，ソフトウェアライセンス違反を自動的に検出することを目的とし，ソフトウェアライセンス違反検出問題を設定した．また，問題を解くにあたって問題となる点について取り組んだ．

本章では，まず，研究背景として，ソフトウェア再利用とその法的問題について述べる．次に，オープンソースソフトウェアを再利用する際留意すべき事項であるソフトウェアライセンスについて述べる．その後，ソフトウェアライセンス違反をどう予防するかについて述べ，その関連研究としてライセンス特定手法，コードクローン検出手法，剽窃検出手法について述べる．次に，既存研究に基づき，ソフトウェアライセンス違反検出手法の概要を述べ，手法における問題点の整理を行い，それぞれの問題点に対してどのような研究を行ったかについて述べる．

1.1　ソフトウェアの再利用

ソフトウェアの再利用とは既存のソフトウェア部品を同じシステム，もしくは新しいシステムで利用することである [19]．ソフトウェア部品とは，クラスや関数などのソフトウェア開発過程で生成される成果物を指す．

ソフトウェア再利用のメリットとして，Lim[41] は以下の項目を挙げている．

- ソフトウェアの品質向上
- 開発の生産性向上

- 市場に出すまでの時間を短縮する
- 一貫したアプリケーションの機能
- コストやスケジュール超過のリスクを削減する
- ユーザの要求のプロトタイピングや検証ができるようになる
- 専門技術や知識の活用

再利用の対象となるソフトウェア部品は，社内で以前開発されたソフトウェアやオープンソースソフトウェア，プログラミングの本に含まれるサンプルに含まれている．また，近年では商用ソフトウェア部品 (Commercial Off the Shelf, COTS) も販売されている．

特に，本稿では，ソフトウェア部品の供給源として，オープンソースソフトウェアに着目する．オープンソースソフトウェアは多数開発されている．FLOSSMole[33] によると，SourceForge.net にて 204439 件 (2009/12 時点)，Google Code にて 12711 件 (2011/5 時点)，Freshmeat で 38027 件 (2011/5 時点) のプロジェクトが存在する．また，オープンソースソフトウェアプロジェクトは指数関数的に増加している．Russo ら [23] によると，5000 の活動中のプロジェクトを調査し，x を 1995 年 1 月から経過した月数，y をオープンソースプロジェクト全体の件数とするとき，以下のモデル（$R^2 = 0.956$) を得ている．

$$y = 7.1511e^{0.0499x}$$

このように，オープンソースソフトウェアはソフトウェア部品の非常に大きな供給源となっている．

大量にあるソフトウェア部品を様々なオープンソースソフトウェアから収集し，開発者の目的にあったソフトウェア部品を取得するのは，開発者にとって大きな労力のかかる作業である．その作業を支援するために，ソフトウェア部品検索システムがある．ソフトウェア部品検索システムとはソフトウェア部品を収集，蓄積，解析し，蓄積したソフトウェア部品に対する検索手段を提供するシステムである．既存のソフトウェア部品検索システムとして，Google Code Search[9] や Black Duck Koders.com[1] などがある．また，筆者が所属する研究グループでは SPARS-J[70] を開発している．

これらのソフトウェア部品検索システムでは，ソフトウェア部品としてソースファイルが取得できる．そのため，本稿では，オープンソースソフトウェアを利用した，ソースファイル単位のソフトウェア再利用に焦点を当てる．

1.2 ソフトウェア再利用の法的問題

ソフトウェアの再利用を行う際，ソフトウェア部品を統合先のソフトウェアに合わせて改変し，再利用したソフトウェア部品を含めてソフトウェアを配布することが想定される．

一方でソフトウェアは知的財産である．ソフトウェアの所有者に対する知的財産保護の手段として，著作権，営業秘密，特許が利用可能である [41]．以下，それぞれについて説明する．

著作権 ソースコードのリストやプログラムによって生成されたテキストを含めた「記述された」表現をそのまま複製されないようにするための権利である [59]．著作権により，著作者は著作物の利用を管理できる．しかし，インターフェイスやプロトコルなどの規約や，アルゴリズムなどの解法，プログラム言語などには著作権は発生しない [62]．ソフトウェアの場合，複製，再配布，改変が該当する [68]．

営業秘密 秘密として管理されている生産方法，販売方法その他の事業活動に有用な技術上または営業上の情報であって，公然と知られていないものである．不正競争防止法により保護される [59]．

特許 新たな技術を開発したものに独占的な権利を与えるもの．日本では，特許法により「発明」とは「自然法則を利用した技術的思想の創作のうち高度のものをいう」とし，「発明」を特許として保護するとしている [62]．

本研究では，特に著作権に着目する．ソースコードとして表現されたソフトウェアは主に著作権により保護される．著作権によりソフトウェア部品を改変したり，複製したり，再配布する行為は著作者に対して，排他的に認められている権利である．そのため，他者により作成されたソフトウェアを再利用するためには，著作者とライセンス契約を結ばなければならない．

1.3 ソフトウェアライセンス

著作権で保護された著作物を利用する際には，その著作物の著作者とライセンス契約を結び，その著作物のソフトウェアライセンス（以下，ライセンス）を得る必要がある．ライセンスとは，著作物を利用するための許諾と，その許諾を得るための義務である．

オープンソースソフトウェアの場合も同様であるが，ライセンスはソフトウェアとともに文書として記述されている場合が多い．オープンソースソフトウェアの場合はそのソースファイルで指定されたライセンスを遵守することを条件に，オープンソースソフトウェアの改変や，再配布を許諾するという構造になっているのが一般的である [12]．

オープンソースソフトウェアには，様々なライセンスが用いられている．Open Source Initiative[1]により，定められているオープンソースソフトウェアの定義[2]を満たすオープンソースライセンスは現在 69 種類である．しかし，これら以外にもライセンスは存在しており，BlackDuck は自社が所有する Black Duck KnowledgeBase には 2050 種以上のライセンスに対する詳細なデータが含まれているとしている [4]．

オープンソースライセンスの具体例を以下に示す．

GNU Public License(GPL) Free Software Foundation[7] が定めたオープンソースライセンス．本ライセンス下でバイナリファイルを配布する場合，ソースファイルも同時に入手可能でなければならない．また，動的リンク，静的リンクにかかわらず利用したソフトウェアも GPL で配布されなければならない．

[1] http://www.opensource.org/
[2] http://www.opensource.org/docs/osd

GNU Lesser Public License(LGPL)　Free Software Foundation[7] が定めたオープンソースライセンス．GPL と同様に，LGPL のもとで配布されているソフトウェアを利用したソフトウェアも LGPL のもとで配布されなければならない．しかし，動的リンクによる利用の場合には，LGPL のもとで配布しなくてよい．

Eclipse Public License(EPL)　Eclipse[5] で用いられているオープンソースライセンス．EPL のもとで配布されているファイルが改変された場合，改変後のファイルも EPL の下で配布されなければならない．

BSD 3-clause License　カリフォルニア大学バークレー校により定められたオープンソースライセンス．前述の 3 つのオープンソースライセンスと異なり，派生物に対するライセンスの制約が存在しない．また，本ライセンスの条項の一つである，作成した組織の名前や貢献者の名前を派生物の宣伝に使用してはならないという条項を撤廃した BSD 2-clause License も存在する．

ソースファイルのライセンスは，ソースファイル中のコメントで指定されている場合が多い．本稿では，この記述をライセンス記述と呼ぶ．ライセンス記述は図 1.1 のようにライセンスの条文がライセンス記述となっている場合や，図 1.2 のようにライセンスへの参照が記述されている場合がある．

どのオープンソースライセンスを選択するかはプロジェクトの活動に対して影響を与える．Colazo ら [22] はコピーレフト性のあるオープンソースライセンスを用いるオープンソースソフトウェアプロジェクトはコピーレフト性のないオープンソースライセンスを用いるプロジェクトと比べ，開発者の地位やコーディングの活発さ，永続性が高く，また，開発期間が短くなると述べている．Stewart ら [52] はオープンソースライセンスの制約性と組織的資金援助がオープンソースソフトウェア開発へのユーザの興味や，開発者の活動へどう影響を与えるか調べた．その結果として，ユーザは非商業組織により資金援助され，非制約的ライセンスを用いるプロジェクトに最もひきつけられることが分かった．また．開発者の活動へのオープンソースライセンスの影響はプロジェクトが持つ組織的な資金援助の種類に依存することが分かった．

ソフトウェア部品検索システムにより．オープンソースソフトウェアの再利用が行いやすくなった一方で，ソフトウェアライセンス違反（以下，ライセンス違反）が多く発生している．ライセンス違反とは，あるファイルのソースコードが別のソフトウェアにコピーされたとき，あるファイルのライセンスと別のソフトウェアのライセンスに不整合が起こっている状態とする．

以下，ライセンス違反の例を示す．

セイコーエプソン製スキャナ及び Linux 用ドライバにおける GPL，LGPL 違反 [16]　国際化のために利用していた gettext パッケージのソースコードの一部が GPL であるにも関わらず，GPL の条件を満たさないバイナリファイルのみという形態で配布を行っていた．また，ソースコード非公開のライブラリにおいて，LGPL のもとで配布される libc ライブラリとリンクしていたが，ライブラリの使用許諾と LGPL が適合しなかった．

図 1.1: ライセンスの条文全体が載るタイプのライセンス記述の例

Copyright (c) 2005, the JUNG Project and the Regents of the University of California All rights reserved.

This software is open-source under the BSD license; see either ”license.txt” or http://jung.sourceforge.net/license.txt for a description.

図 1.2: ライセンス名が指定されるタイプのライセンス記述の例

ゲームソフトにおける GPL 違反 [14]　プレイステーション 2 用ソフト「ICO」において，GPL のもとで配布されている libarc ライブラリが使用されていたにもかかわらず，ソースコードが配布されていなかった．

ライセンスに違反した場合，そのソフトウェアの利用が認められない．そのため，ソースコードを公開する必要が生じたり，開発したソフトウェアを販売することができなくなる．このように，ライセンスは企業等に大きな損害を与えうる．

また，ライセンス違反が発生していても気づきにくい側面がある．なぜなら，ライセンス違反が生じていたとしても，ソフトウェアの動作には影響を与えないためである．そのため，ライセンス違反を防止するためには，ライセンス違反となるソースコードがソフトウェアに含まれていないか検査する必要がある．

1.4　ソフトウェアライセンス違反の予防策

ライセンス違反を予防するためには，開発中，開発後にライセンス違反が発生していないか確認する必要がある．

開発中にライセンス違反を予防するためには，再利用するソースファイルのライセンスをきちんと調べ，それが開発中のソフトウェアのライセンスと矛盾しないことを確かめたうえで再利用を行えばよい．しかし，ソフトウェア部品検索システムを通じてたくさんのソースファイルを取得できるうえ，一つの機能が一つのソースファイルのみから構成されているわけではない．そのため，この作業は何度も繰り返される可能性がある．そこで，ソースファイルのライセンスを容易に確認できる手段が必要となる．

次に，開発後にライセンス違反を予防するためには，ソフトウェアのソースファイルに別のソフトウェアのソースコードが再利用されていないか確認し，再利用されている部分が見つかった場合は，その再利用されたと思われるソースファイルと開発したソフトウェアのライセンスが矛盾していないか確かめればよい．近年，ソフトウェア開発が国内，国外へと外注される場合が増えており，特に，この確認は重要である．そのためには，まず，再利用されたと思われる部分を発見しなければならない．そのため，再利用部分を検出することが重要である．

また，どちらの場合においても，ソースファイルのライセンスとソフトウェアのライセンス間の比較は行わなくてはならない．ただし，ライセンス間で矛盾が生じているかの判断は法的問題であり，工学的立場のみから論じることはできないと考える．ライセンスの比較の結果，矛盾していることが分かった場合は，適切に解消しなくてはならない．

以下，ライセンスの確認を容易にすることの関連研究としてライセンス特定手法，外注により生成されたソースファイルの中に不正な再利用がないかチェックすることの関連研

究として，コードクローン検出手法，剽窃検出手法を説明する．ライセンス間の比較を行うことの関連研究として，ライセンス不整合を扱った研究について説明する．

1.4.1 ライセンス特定手法

既存ライセンス特定手法として，FOSSology, ALSA, OSLA, Ohcount を紹介する．

FOSSology[31] bSAM アルゴリズム[3]を用いて既知のライセンス記述とソースコード中のコメントを比較する．コメント中で類似したライセンス記述を持つライセンスを特定結果として出力する．

ASLA[55, 56, 57] 各ライセンス記述に対応した正規表現を用いて，ソースコード中のコメントにマッチした正規表現からライセンスを特定する．

OSLC ソースコード中のコメントとライセンス記述間で同一となる行を発見し，最長一致列となる部分を探索する．類似度はライセンス記述に対する最長一致列の割合である．類似度の高いライセンス記述に対応するライセンスを特定結果として出力する．

Ohcount 各ライセンス記述に対応した正規表現を用いて，ソースコード中のコメントにマッチした正規表現からライセンスを特定する．ASLA と違い，すべて正規表現はハードコーディングされている．

コードクローン検出

コードクローンとは，互いに一致または類似したコード片を指す．あるファイルのソースコードがコピーアンドペーストにより他のファイルに流用されている場合，流用により生じた部分が２ファイル間にまたがるコードクローンとして検出することができる場合がある．

既存のコードクローン検出手法では，プログラムを特定の形式に変換し，そのうえで同一部分を見つけることによって，コードクローンを検出している．用いられる形式としては，文字列 [25]，トークン [35]，抽象構文木 [18, 27, 34, 38, 61]，プログラム依存グラフ [32, 37, 36]，メトリクス [44] などがある．

剽窃検出手法

剽窃検出を目的とした手法として，JPlag[48]，MOSS，GPLAG[42]，Steven らの手法 [20] がある．JPlag[48] はプログラムをトークン列として比較し，２プログラム間で一致している部分が２プログラム全体に対して占める割合に基づいて剽窃を検出する．MOSS は [50] のアルゴリズムに基づき，剽窃を検出する．GPLAG[42] はプログラム依存グラフにおい

[3] http://fossology.org/symbolic_alignment_matrix

て，同形部分を検出することで，剽窃を検出する．Steven ら [20] はテキスト類似度と local alignment を用いて剽窃検出する方法を提案している．

また，剽窃検出手法として，ソフトウェアバースマークを用いたものもある．ソフトウェアバースマークとは，p, q を与えられたプログラム，$f(p)$ を p からある方法 f により抽出された特徴の集合とするとき，以下の性質を満たす $f(p)$ のことである [53].

- $f(p)$ はプログラム p のみから外部情報を用いることなく得られる．
- q が p のコピーならば，$f(p) = f(q)$ である．

また，ソフトウェアバースマークを評価する観点として，以下の性質を満たすことが望まれる．

保存性

p から任意の等価変換により得られた p' に対して $f(p) = f(p')$ を満たす．

弁別性

同じ仕様を持つ p と q に対し，それらが全く独立に実装された場合，$f(p) \neq f(q)$ となる．

既存研究では，ソフトウェアやソースコードに含まれる様々な特徴をソフトウェアバースマークとして用いている．玉田ら [66] は java クラスファイルの初期値代入部分やメソッド呼び出し部分を用いている．Myles ら [46] はオペコードの k-gram を用いる．TAMADA ら [53] はフィールドの定数値，メソッド呼び出しの列，頂点をクラス，継承関係を辺とするグラフ上においてあるクラスから java.lang.Object まで到達する際に通過するクラスの列，使用しているクラスを用いる．Lim ら [40] はブロックを頂点，辺を制御フロー関係とする有向グラフである制御フローグラフにおける各ブロックから到達できるブロック列に含まれるバイトコードのふるまいを用いる．

また，バイナリファイルから特徴を抽出し，バイナリファイル間の類似性を扱うバースマークもある．Choi ら [21] はバイナリファイルを逆アセンブルして得られたコールグラフと，各ブロックに含まれる API を用いる．Park ら [47] は実行した際，オペランドスタックの深さが 0 で開始し，再び 0 に戻るようなバイトコード列の集合を用いる．Myles ら [45] はプログラム実行時の実行経路のトレースを用いる．

1.4.2 ライセンス不整合に関する関連研究

ライセンス不整合を取り扱った研究として，Agrawal ら [17] と German ら [28] がある．Agrawal ら [17] はライセンスの条項を $\langle actor, operation, action, object \rangle$ のタプルで表現し，ソフトウェアトレーサビリティツールの ArchStudio4 上で義務の伝搬，義務の衝突，システム全体で利用可能な権利を計算する手法を提案した．German らはソフトウェアパッケージのライセンスとソフトウェアパッケージを構成するソースファイルのライセンスが適合するかどうか調べる手法を提案している [28].

ライセンスが不整合を起こした時どう対処するかについて取り扱った研究として，German ら [29] の研究がある．German らはライセンスが不整合を起こした際，どのようにしてライセンスの不整合を解消するかのパターンをライセンスを与える側とライセンスを受ける側それぞれの観点からまとめた．

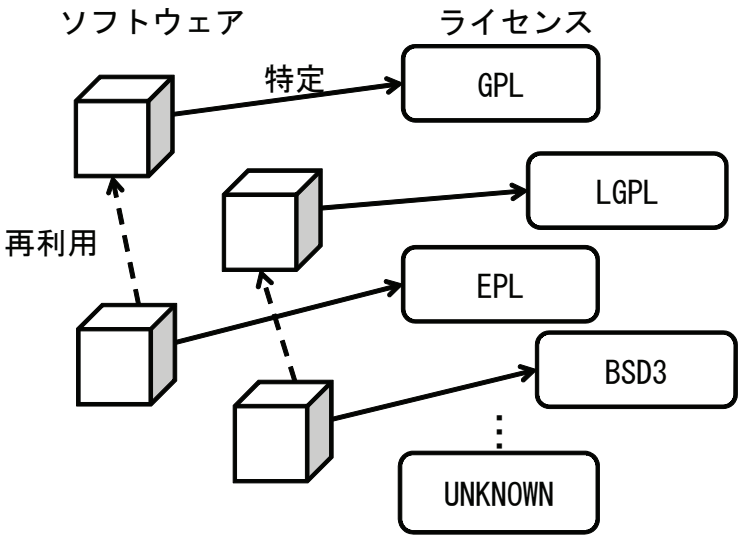

図 1.3: ライセンス違反検出問題の概要図

1.5 ソフトウェアライセンス違反検出問題

ライセンス違反の予防は開発中，開発後にできることとした．しかし，自動的にライセンス違反を検出できれば，開発者のライセンス違反を予防する労力を軽減でき，ライセンス違反予防に効果があると考える．そこで本稿では，ライセンス違反を自動的に検出することを目指す．

本稿では，ライセンス違反検出問題を設定する．ソフトウェアとライセンスの関係を図 1.3 に示す．ここでのソフトウェアとは，ソースファイルの集合，または，単一のソースファイルとする．この図では，ソフトウェアからはそのソフトウェアのライセンスが決定でき，また，ソフトウェア間では再利用が行われている場合があることを示している．ライセンス違反検出問題とは，ソフトウェアのライセンスと，そのソフトウェアが再利用しているソフトウェアのライセンス集合とを比較し，矛盾の有無を判定する問題である．

しかし，実際にはソフトウェアの集合，ライセンスの集合は既知であるが，ソフトウェアのライセンスが何であると特定されるかは既知でない場合があり，どのソフトウェア間で再利用が行われているかは既知ではない．これらを調べるため，本稿では，1.4 節で紹介したライセンス違反予防に関する関連研究に基づき，ライセンス違反検出手問題を解く．各ソフトウェアのライセンスを特定するため，ライセンス特定手法を用いる．また，ソフトウェア間で行われている再利用を検出するため，本研究ではコードクローン検出手法を用いる．また，なお，1.4 節でも述べたとおり，ライセンスの比較は法的問題であり，人手による判断が必要となる．そのため，ライセンスの比較部分は自動化しない．

ライセンス違反検出問題を解くにあたって，以下の点が問題となる．

問題点 1 として，ライセンス特定手法において，通常，ソースファイルのライセンスはソースファイル中のコメントで記述されるが，ライセンス記述の表記ゆれにより既知のライセンス記述との単純な照合だけではライセンスが特定できないことを挙げる．ソー

スコード中のライセンス記述には，綴り間違いや単語の同義語への変更などの表記揺れ，ソースファイルにより変化する著者名や組織名が含まれるため，簡単な照合だけで特定することは困難である．既存手法ではこれらを実証的に調査し，その結果に基づいている手法は存在していない．

問題点 2 として，ライセンス特定手法において，ソースファイルのライセンスがどの程度変更されるか明らかでないため，ライセンスの特定はどの程度の頻度で行う必要があるかわからないことを挙げる．大規模な FOSS には様々なライセンスのファイルが含まれている可能性があり，また，それらのライセンスは著作者の意向により変更される可能性がある．

問題点 3 として，コードクローン検出手法，既存手法では，あるソースファイル間で再利用が行われていると判断するための明確な基準が存在しないことを挙げる．現状，既存手法では，再利用が行われているかどうかを判断する基準は一つの設定に過ぎず，実証的な裏付けがなされたものではない．

これらの問題を解決するため，本研究では以下の研究課題に取り組んだ．

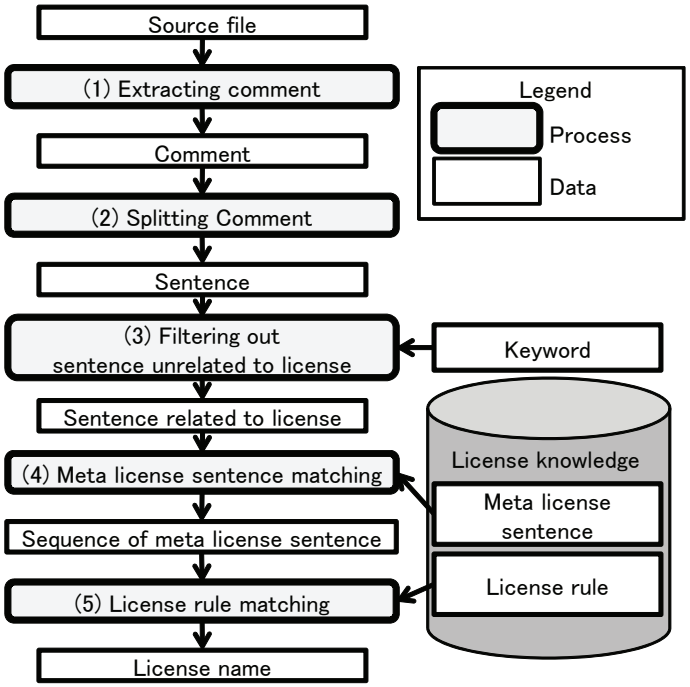

図 1.4: ライセンス特定ツール Ninka の構成

階層的ライセンス知識を用いたライセンス特定ツールの開発 [第 2 章]　問題点 1 に対し，本稿では，オープンソースソフトウェアから抽出した知識に基づいたライセンス特定手法を提案する．最初に複数の大規模オープンソースソフトウェアに含まれる約 30000 個のソースファイルを調査し，実証的にライセンス特定の課題と要件を定めた．次に，これらの課題と要件，既存研究の状況からライセンス特定ツールに求められる要件を定めた．こ

れらの要件を満たすように，ライセンス特定ツール Ninka を構築した．Ninka の構成を図 1.4 に示す．Ninka は複数の大規模オープンソースソフトウェアから抽出した知識を用い，また，この知識は階層化されている．評価実験として，既存のライセンス特定手法と回答率，正答率，所要時間で比較を行い，提案手法が既存の手法より優れていることを示した．

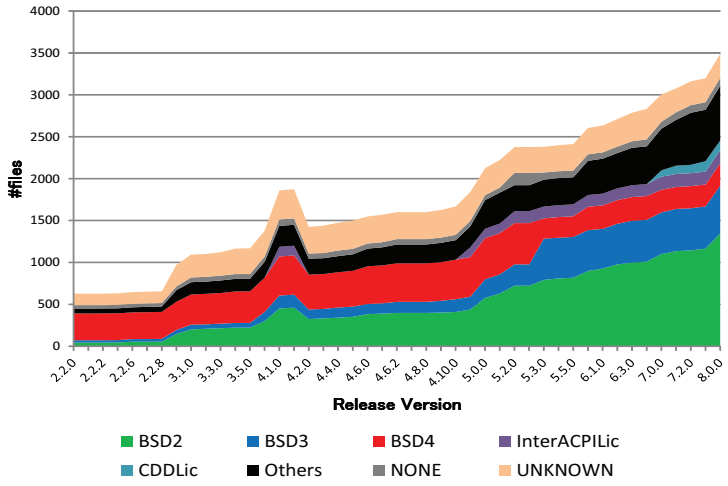

図 1.5: FreeBSD のライセンス比率

オープンソースソフトウェアにおけるライセンス分布の調査 [第 3 章]　問題点 2 に対し，オープンソースソフトウェアにおけるライセンスの分布を調査した結果を示す．本調査では長期間開発が行われている複数のオープンソースソフトウェアを解析対象とし，ライセンスの分布とライセンスの分布がどのように変化するか，その傾向を調べた．図 1.5 に FreeBSD のライセンス分布が時系列に沿ってどのように変化したかを示す．調査結果から，調べた対象のオープンソースソフトウェアには様々なライセンスのファイルが含まれているだけでなく，その比率がソフトウェアの進化に沿って大規模に変化することが分かった．また，オペレーティングシステムと非オペレーティングシステムでは進化の傾向が異なっていることが分かった．

コードクローンメトリクスに基づくソースコード再利用判定閾値の決定手法 [第 4 章]　問題点 3 に対し，コードクローンメトリクスを用いたソースコード再利用検出手法を提案する．本論文では，3 つのコードクローンメトリクス（コードクローン検出数，最大コードクローン長，部分類似度）を用い，各コードクローンメトリクスについて，ソースコード再利用があるとみなせる閾値，ソースコード再利用がないとみなせる閾値を算出する．図 1.6 にその概要を示す．実験では，50 件の OSS を用い，実際に検出したコードクローンについて，再利用により生成されたものかを調査した．次に，先ほど調査により，ソースコード再利用が行われているソフトウェアの組をソースコード再利用が行われている場合の正解集合，それ以外の組をソースコード再利用が行われていない場合の正解集合とし，

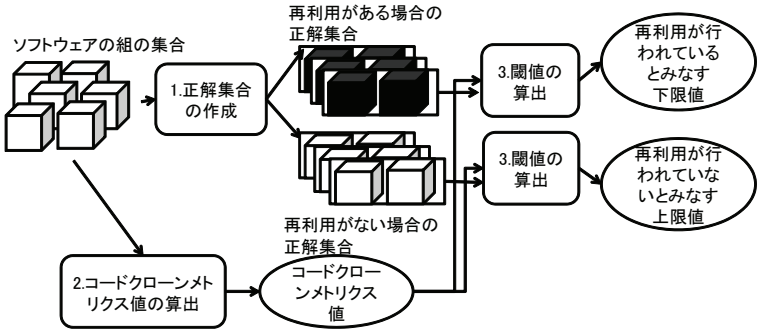

図 1.6: ソースコード再利用判定閾値の概要

ソフトウェア間で検出されるコードクローン検出数，最大コードクローン長，部分類似度と適合率，再現率の関係を調査した．そして，各メトリクスについてソースコード再利用ありの場合に適合率が 1 となる定義域から，ソースコード再利用が行われたとみなす下限値，ソースコード流用なしの場合の適合率が 1 になる定義域から，ソースコード流用が行われていないとみなせるメトリクス値の上限値を求めた．また，それらの閾値を用いることでソフトウェア再利用が行われているソフトウェアの組 121 組のうち 102 組を検出することができた．

本稿での研究成果は以下のようにまとめられる．従来手法の FOSSology と比べてライセンス特定の精度を 75%(55.0%→ 96.6%) 向上できており，従来手法を用いる場合と比べてより正確にライセンスを特定できるようになった．また，オープンソースソフトウェアではライセンスが大きく変更されることが確認できたため，各ソースファイルについて，各バージョンについてもライセンス特定を行う必要があることが明らかになった．また，検出したコードクローンに基づき，ソースファイル間で再利用が行われたと判断できる閾値を実験的に求めたことで，実証的なデータという根拠をもって再利用が行われたソースファイルの組のうち，半数程度を誤りなく判別できるようになった．そのため，どのソフトウェア間に再利用があるかを根拠に基づき判別できるようになった．

第2章　階層的ライセンス知識を用いたライセンス特定ツールの開発

2.1　はじめに

開発コスト削減の一手段として，既存のソフトウェアの一部や全部を，部品として同一システム内や他のシステムで再利用することが広く行われている [19]．再利用においては，ソフトウェア部品に修正が加えられることも多い．Selby によれば，NASA software development environment が持つ 25 のソフトウェアシステムのためのリポジトリを調査したところ，全体の 14.75%のモジュールが修正が加えられた上で再利用されていた [51]．

特に，オープンソースソフトウェア（以下，OSS）に含まれる関数やクラスなどのソフトウェア部品（以下，部品）が利用されることが多い．Li らによると，36%の企業がオープンソースソフトウェアを修正して利用している [39]．

OSS の部品を利用するためには，多くの場合，ソースファイルのコメントとして記述されている，利用に関する許諾と許諾を受けるために負う義務を示したソフトウェアライセンス（以下，ライセンス）を読み，理解し，それを遵守しなければならない．

一般に，ライセンスは自然言語で記述された複数の文で構成されており，また，前提知識が必要だったり，他のファイルへの参照を含んだりするため，開発者がライセンスを正確に読解することは容易ではない．

一方，多くの OSS 部品のライセンスは，定型の文で構成された既知のライセンスが用いられている場合が多い．この場合，ライセンスの文章全体を詳細に読解せずに，既知のどのライセンスと合致しているかが分かれば（これをライセンスの特定と呼ぶ），どのような許諾条件や義務があるか比較的容易に認識でき，その許諾条件や義務をそのまま適用することで，効率的な部品の利用を促進することができる．

しかし，ライセンスの特定も容易ではない．ソースコード中のライセンスの記述には，綴り間違いや単語の同義語への変更などの表記揺れ，ソースファイルにより変化する著者名や組織名が含まれるため，簡単な照合だけで特定することは困難である．そのため，細かい差異を認識して，ライセンスの特定を効率良く行えるシステムが望まれる．

現在，いくつかのライセンス特定ツールが提案されているが，それらはライセンスの版（バージョン）を答えない，複数回答を報告する，ライセンスを特定する際に使用する知識の管理が容易でない，などの問題がある．

本稿では，まず上記のライセンス特定の問題を詳しく理解するため，複数の OSS からソースファイルに含まれるコメントを収集し，調査を行い，ライセンス特定における課題を特定した [30]．次に，調査により明らかになったライセンス特定の課題と，OSS の現状に基づき，ライセンス特定ツールに必要な要件を定めた [69]．そして，それらの要件を満

All right reserved. Redistributions of source code must retain the above
copyright notice. Redistributions in binary form must reproduce
the above copyright notice.

図 2.1: ライセンス記述の例

1. All right reserved.
2. Redistributions of source code must retain the above copyright notice.
3. Redistributions in binary form must reproduce the above copyright notice.

図 2.2: ライセンス文の例

たせるよう手法の設計，実装を行った．

本研究では，階層的なライセンス知識を用いた特定手法を提案する．提案手法が従来手法と比較して精度，スケーラビリティ，ライセンス知識の管理の点でどのように優れているかを評価するため，評価実験を行った．結果として，提案手法は従来手法に比べ，実用的な精度，時間で特定することができた．また，大半のソフトウェアでソフトウェアに含まれるすべてのソースファイルに対して，ライセンスを特定することができることを示した．

以降，2.2 節で本稿で用いる用語を定義する．2.3 節でライセンス特定の課題について説明し，課題を解決するために必要な要件を述べ，そして，それらの要件と既存手法との対応関係を述べる．2.4 節では，関連研究として，既存のライセンス特定手法について説明する．2.5 節では提案するツールの設計について述べ，ツールの実行例を示す．2.6 節では提案するツールを精度と速度，スケーラビリティ，ライセンス知識の表現法の観点から他の既存のライセンス特定ツールとの比較を行った評価実験について述べる．2.7 節では，まとめと今後の課題について述べる．

2.2 用語

ライセンスとは，著作者が定めたソフトウェアの利用者に対する指示の集合である．良く知られているライセンスの名前と本稿で用いるそれらの略記を表 2.1 に示す．本稿では，ライセンスに含まれる指示の具体的な内容については議論しない．

ソースファイルには，プログラムコードとコメントが記述される．**コメント**はライセンスに関係のある文の系列（**ライセンス記述**）もしくはその他の文字列から構成されている．ライセンス記述中の各文のことを**ライセンス文**と呼ぶ．図 2.1 にライセンス記述の例を，図 2.2 に図 2.1 のライセンス記述に含まれるライセンス文の例を示す．

メタライセンス文とはライセンス文の集合を拡張正規表現で表記したものである．図 2.2 の文 (2) において，above が省かれた文も同様なライセンス文として取り扱う場合，これらを一般化したメタライセンス文は

```
Redistributions of source code must retain the (above )? copyright notice.
```

表 2.1: 一般的なオープンソースライセンスの名前と本章での略記

Abbrev.	**License Name**
Apache	Apache Public License
BSD4	Original BSD, also known as BSD with 4 clauses
BSD3	BSD with 3 clauses (BSD4 minus advertisement clause)
BSD2	BSD with 2 clauses (BSD4 minus advertisement and endorsement clauses)
boostV1	The Boost Software license
CPL	Common Public License
CDDL	Common Development and Distribution License
EPL	Eclipse Public License
GPL	General Public License
GPLnoVersion	GPL with no version indicated
IBM	IBM Public License
LesserGPL	Lesser General Public License (successor of the Library GPL, also known as LGPL)
LibraryGPL	Library General Public License (also known as LGPL)
MIT/X11	Original license of X11 released by the MIT
MITold	License similar to the MIT/X11, but with different wording
MPL	Mozilla Public License
SameAsPerl	File is licensed in the same terms as Perl
SeeFile	File points to another where the its license is
SunSimpleLic	Simple license used by software developed by Sun Microsystems
ZLIBref	The zlib/libpng license
publicDomain	File is in the public domain

となる，ただし，(xyz)?は xyz を省略可能であることを示す．

ライセンスルールとは，メタライセンス文の系列からライセンスへの対応を定義するためのメタライセンス文の系列とライセンスの組である．メタライセンス文の集合及び，ライセンスルールの集合をまとめて**ライセンス知識**という．

ライセンスの特定とは，ソースファイルからそのソースファイルのライセンスを決定する作業をいう．

2.3 ライセンス特定の課題と要件

2.3.1 課題

ライセンス特定問題の課題を特定するため，我々は OSS に現れるライセンス宣言の特徴を調査した．調査では，各ソースファイルのライセンス記述について，ライセンス記述に含まれる文や，文の並びについて調査した．そして，調査により得られた知見をライセンス宣言の特徴としてまとめた．調査で，Linux，Eclipse JDT，OpenBSD，Mozilla などに含まれるソースファイルからなるおよそ 30000 ファイルを調査対象とした．調査により，以下の課題を特定した．

課題１：ライセンス記述の表記揺れ　ライセンス記述は表記揺れを含むことがある．一つ目の理由は綴り間違いである．これに該当する記述として，本来分割されるべきでない単語が空白により分割されていたという記述がある．

もうひとつは，あらかじめライセンス記述に著者名や団体名を記述する部分が決められている場合があり，そこに具体的な著者名や団体名が入ることにより表記揺れが発生する．例として，BSD ライセンスでは，条文として *“THIS SOFTWARE IS PROVIDED BY* $\langle name \rangle$ *AS IS AND ANY EXPRESS . . . ”* という条文を持つが，この条項のうち，$\langle name \rangle$ の部分は著作者名を入れることになっている．そのため，表記揺れが生じる．

次に，ライセンスの版の違いにより，表記ゆれが発生する場合がある．例として，GPLv2 では *“either version 2 of the License, or (at your option) any later version.”* となる記述が，GPLv3 では，*“either version 3 of the License, or (at your option) any later version.”* となる．

また，著作者により，文法や綴りの変更が行われる場合がある．確認できたこの種の変更として，*“license”* と *“licence”*，*“it would be useful”* と *“it will be useful”*，*“developed by”* と *“written by”*，*“dealings in”* と *“dealings with”*，*“Redistribution”* と *“Redistributions”* の置換があった．また，句読点においてもコンマやセミコロンの追加，削除が行われていた．

課題２：著作者による既存のライセンスの修正　著作者により，既存のライセンスにある条項を修正して使用する場合がある．また，開発者の目的にあったライセンスを定めるため，既存のライセンスに条項を追加したり，削除したりすることがある．

例として，BSD4 からの派生がある．BSD3 は BSD4 から一つの条項（宣伝条項）を取り除いて作成され，Apache v1.1 は BSD3 に 4 つの文を追加することで作成されている．また，Apache v1.1 から OpenSSL ライセンスが派生している．

より修正が捕らえにくい例として，MIT/X11 ライセンスでは *“Premission to use, copy, modify, distribute, and sell this software . . . ”* から *“sell”* が欠落した場合が多く存在した．また，BSD の条項 *“Redistributions of source code must retain . . . ”* が *“Redistributions of source code or documentation must retain . . . ”* へ変更された場合があった．

別のテキストファイルにライセンス記述が記述されるGPLやMPLなどのライセンスでは，許諾者が補遺を用いてライセンスを修正することを認めている．これは，ライセンスの派生を避けるためである．これらの補遺は例外として知られている．

課題３：ソースファイルとは異なるファイルへの参照がある　ライセンスの指定はコメントにより行われることが多い．しかし，他のファイルにライセンスが記述され，コメントにはそのファイルへの参照が記述されている場合がある．このようなファイルへの参照は *“For licensing information, see the file . . . ”*，*“See . . . for licensing information”*，*“See . . . for license detail”* などと記述されている．また，このようなファイルは *“main”*，*“root”*，*“top”* ディレクトリにある場合が多く，その名前も *“COPYING”*，*“LICENSE”*，*“README”*，*“copyright.txt”*，*“AUTHORS”*，*“COPYRIGHT_and_LICENSE”* などさまざまである．

課題４：複数のライセンスが含まれる　一つのファイルに複数のライセンスが含まれる場合がある．これは，ファイルのライセンスを複数のライセンスから選択できる場合，もしくはソースコード中に以前用いられていたライセンスのライセンス記述が残ったまま，新しいライセンス記述が記述された場合がある．これらのライセンスはそれぞれ別のライセンス記述で表現される場合と，一つのライセンス記述で表現される場合がある．ライセンス特定問題においては，複数のライセンスが含まれている場合でも，すべてのライセンスが特定されていることが望ましいと我々は考えて，できるだけ多くのライセンスを特定するようにした．

2.3.2　ライセンス特定ツールに求められる要件

2.3.1節で述べたライセンス特定の問題と，OSSにおけるライセンスの特徴に基づき，今回のライセンス特定ツールに求められる要件を以下のように定めた．

要件１：多くの表記ゆれを考慮してライセンスを特定できる　課題１として，ライセンス記述に表記揺れが含まれるという課題をあげた．しかし，実際に，どのようにライセンス記述に表記揺れが含まれるか，類似したライセンス記述があるかどうかについては明らかになっていない．そのため，実際に開発されているOSSを多数調査し，その結果得られたライセンス記述の表記ゆれを考慮してライセンスの特定ができる必要がある．また，調査により明らかにできた表記揺れや，類似したライセンス記述を認識してライセンスを特定するため，細かい文字列の差異により生じる対応するライセンスの違いを正しく認識できる機構を備えていることが望ましい．

要件２：新しいライセンス記述への適合が容易　Open Source Initiative[1]に承認されているOSSの定義を満たすライセンスは76種あるが，それ以外のライセンスも多数あり，必要に応じてそれらを特定できるようにしたい．また，課題４で挙げた通り，ひとつのライセンスが複数の記述を持つ場合がある．これらに対応するため，新しいライセンスに容易に適合できるようにするための機構を備えていることが望ましい．この機構が満たすべき要件としては，同一のライセンスに対して，複数のライセンス記述を定義できること，短いルールとしてライセンス記述を定義できること，ツールに手を加えることなく後からライセンス記述を追加できることが挙げられる．

[1] http://www.opensource.org/

要件３：高速に処理できる　OSS の部品を利用するためには，事前に各部品のライセンスを特定しておくことが便利である，しかし OSS の規模が大きいと，部品の数も増え，全てのライセンスの特定に要する時間も大きくなる．従って，個々のライセンスの特定には，スケーラビリティの高い，高速な処理が望まれる．

2.4　既存のライセンス特定手法

ライセンス特定ツールの既存研究として，Fossology[31] と，ASLA[58] がある．

Fossology は bSAM アルゴリズムを用いて既知のライセンス記述とコメントを比較し，コメント中で類似したライセンス記述を持つライセンスを特定結果として出力する．しかし，このアルゴリズムの計算量が大きいため，スケーラビリティが低く，要件３を満たさない．

ASLA はメタライセンス文のみを用いて特定するライセンスの記述を行う．しかし，各記述の正規表現は複雑で容易に作成できないため，要件２を満たさない（詳しくは 2.6.3 で述べる）．また，これらの研究論文にはメタライセンス文を作成する上で，どのような調査に基づいているのか，また，それらの調査結果に基づいてどのようにメタライセンス文を作成したのかという点が示されていないため，要件１を満たしているかは不明である．

Fossology と同様にアルゴリズムを用いて既知のライセンス記述とコメントを比較するライセンス特定ツールとして Open Soucerce License Checker[2]がある．本手法はコメントとライセンス記述間で同一となる行を発見し，最長一致列となる部分を探索する．類似度はライセンス記述に対する最長一致列の割合である．

ALSA と同様に正規表現を用いたライセンス特定ツールとして，Ohcount[3]がある．これらのツールではライセンス記述に対応する正規表現はハードコーディングされており，追加する仕組みは用意されていないため，要件２を満たすための条件であるツールに手を加えることなく後からライセンス記述を追加できることを満たさない．

また，商用のサービスとして，Palamida[4]，Black Duck Protex[5]，Protecode[6]などがあるがどのような方法で特定を行い，どのような精度で特定できるか不明である．

ライセンスの不整合を扱った研究として，Agrawal らはライセンスの条項を $\langle actor, operation, action, object \rangle$ のタプルで表現し，ソフトウェアトレーサビリティツールの ArchStudio4 上で義務の伝搬，義務の衝突，システム全体で利用可能な権利を計算する手法を提案した [17]．また，German らはライセンスの不整合が生じたときの対処をライセンス統合パターンとして整理した [29]．これらの手法では，ライセンスの特定は扱っていない．

2.5　ツールの設計

図 2.3 に提案するツールの構成を示す．本ツールでは，ソースファイルを入力とし，特定できたライセンス名の列を出力する．本ツールは 5 段階の処理と 2 種類のライセンス

[2]http://forge.ow2.org/projects/oslcv3/
[3]http://sourceforge.net/projects/ohcount/
[4]http://www.palamida.com/
[5]http://www.blackducksoftware.com/protex
[6]http://www.protecode.com/

知識からなる．これらのライセンス知識はツールの外部ファイルとして実装することにより，要件２の「ツールに手を加えることなくライセンス知識を追加できる」という条件を満たす．

本ツールでは，ライセンス知識により正確に特定できたライセンス名のみを出力とする．既存ツールである Fossology や OSLC では，ライセンス知識に入力のソースファイルのコメントと類似したライセンス記述により指定されているライセンスも出力として返す．そのため，ライセンス知識が不足し，正確に特定できない場合でも，ライセンスを特定できる場合がある．一方で，出力されるライセンス名が多くなるため，実際には入力のソースファイルに関係のないライセンスが出力される可能性がある．これは，ライセンスを特定したソースファイル利用する観点から考えたとき，誤ったライセンスに従って利用する可能性を高めることになる．そのため，提案手法では，正確に特定できたライセンスのみ出力することにした．

2.5.1 ライセンス知識

調査ではコメントを文に分割し，どのような文が存在するかを調べた．その結果，多数の表記揺れを検出した．これらを吸収するため，調査により得られた知識をライセンス知識（License Knowledge）として整理した．ライセンス知識は，427 のメタライセンス文（Meta license sentence），112 個のライセンスに対応する 126 個のライセンスルール（License rule）からなる．

提案手法では，**メタライセンス文**を“（メタライセンス文名）:（メタライセンス文に対応するライセンス文にマッチする拡張正規表現のパターン）”と記述する．本ツールでは，要件２を満たす条件の一つである「同一のライセンスに対して，複数のライセンス記述を定義できること」を満たすため，メタライセンス文を記述する際，同一のメタライセンス文名を持つ，複数のメタライセンス文を記述することを許す．例として，提案手法では，メタライセンス文 BSDcondSource に対するメタライセンス文を以下のように記述する．以下の記述において，*“s?”* は *“s”* が省略可能であることを意味し，また，*“(above)?”* は *“above ”* が省略可能であることを意味する．

```
BSDcondSource:Redistributions? of source code must retain the (above )?
copy right notice, this list of conditions(,)? and the following disclaimer
(, without modification)?
```

また，**ライセンスルール**を“（ライセンス名）:（メタライセンス文名の系列）”と記述する．ライセンスルールはメタライセンス文の系列とライセンス名（License name）からなる．例として，BSD2 に対応するライセンスルールを以下に示す．

```
BSD2:BSDpre,BSDcondSource,BSDcondBinary,BSDasIs,BSDWarr
```

このルールでは BSD2 のライセンス記述はメタライセンス文の例として示した BSDcondSource の他，BSDpre，BSDcondBinary，BSDasIs，BSDWarr から構成されており，それら

はBSDpre，BSDcondSource，BSDcondBinary，BSDasIs，BSDWarrの順に並ぶことを定義している．

以上のように，正規表現に対する記号名を導入することによって，あるライセンス記述に対応するライセンス知識をその記述に含まれる文（メタライセンス文）と文の並び（ライセンスルール）として表現することができる．このようにライセンス知識を構成することにより，要件２を満たす条件の一つである「ライセンス記述を短いルールとして記述できる」を満たすことができる．実際に，一つの正規表現として定義した場合，上記の例においてBSDpre，BSDcondSource，BSDcondBinary，BSDasIs，BSDWarr，BSDPreの各ライセンス文を一つの正規表現にまとめなければならず，長大な正規表現としてライセンス記述が表記される．一方，提案手法では，ライセンス記述を複数のルールにより定義できている．

また，ライセンスルールを分離したことにより，ライセンス文を複数のライセンスルールで共有でき，ライセンス知識全体を小さくすることができる．例として，ライセンスルールBSD4ではBSDpre，BSDcondSource，BSDcondBinary，BSDcondAdvPart1，BSDcondAdvPart2，BSDcondEndorseRULE，BSDasIs，BSDWarrとなっており，8つのライセンス文を使用する．一方，ライセンスルールBSD3では，BSDpre，BSDcondSource，BSDcondBinary，BSDcondEndorseRULE，BSDasIs，BSDWarrと6つのライセンス文で構成されており，そのすべてがBSD4のライセンス文になっている（順序は異なる）．このような場合，ライセンス文の共有によりライセンス知識全体をコンパクトに記述できている．

このようにライセンス知識をコンパクトに記述できるため，より多くのライセンス知識を保持することが可能である．そのため，多くのライセンスとそれらの派生に対応することができるようになり，ライセンス特定結果の精度を向上させることができる．

2.5.2 処理

提案手法ではソースファイルを入力とし，ライセンス名を出力する．提案手法では(1) コメント（Comment）の抽出，(2) コメントの文（Sentence）への分割，(3) 文のフィルタリング，(4) 文とメタライセンス文との照合，(5) メタライセンス文の系列（Sequence of meta license sentences）とライセンスルールとの比較を行う．以下，各手順について説明する．

(1) 初めに，ソースファイルからコメントを抽出する．提案手法ではソースコードからフォーマットやコメントを削除するユーティリティであるMangle[7]を改変し，コメント抽出を行った．また，本ツールでは高速化のため，ソースファイル中で最初に現れたコメントの先頭から，次に現れるコードまでをコメントとして抽出する．

(2) 抽出したコメントを文へと分割する．ここでは，区切り文字“．”，“！”，“？”，“：”を検出し，区切り文字と区切り文字の間を文とする．また，文中の改行やタブは一つの空白にする．その結果，連続した空白が現れる場合は一つの空白に置き換える．

(3) 次に，文の集合から，ライセンスに関係のない文を取り除く．提案手法では，キーワード（Keyword）集合に含まれる単語を全く含まない文はライセンスに関係のない文としてみなす．**キーワード集合**とは，ライセンスに関係が深いと考えられる単語もしくは熟

[7] http://gnu.triplemind.com/directory/devel/specific/mangle.html

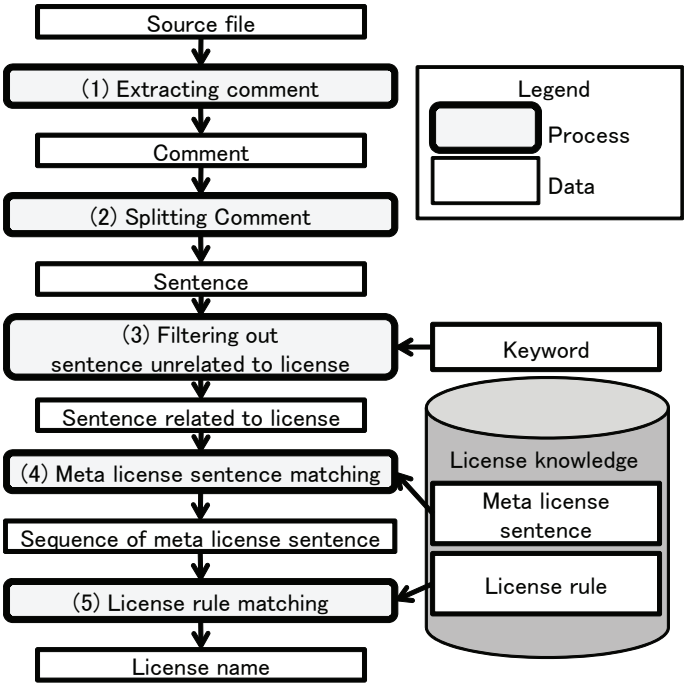

図 2.3: ライセンス特定ツールの構成

語の集合である．現在キーワード集合には，*“as is”*,*“wrote this file”*,*“acknowledgement”*, *“advertising”*,*“conditions”*,*“copies”* などの 82 語が含まれている．

(4) フィルタリング後，残った各文とメタライセンス文集合に含まれる各メタライセンス文の拡張正規表現パターンとマッチングを行い，マッチした場合は文をメタライセンス文名に置換し，メタライセンス文名の系列を得る．

(5) 最後に，メタライセンス文名の列とライセンスルール集合に含まれる各ライセンスルールのメタライセンス文名の列を比較し，一致したライセンスを出力する．

2.5.3 実行例

(1) 入力されたソースコードから以下のコメントを抽出したと仮定する．

```
Redistribution and use in source and binary forms, ...
 ...
1. Redistributions of source code must retain ...
 ... .
2. Redistributions in binary form must reproduce  ...
 ...
 ... .
```

(2) 初めに，コメントを文に分割する．上記のライセンス記述を分割するとき，*“Redistribution and use in source and binary forms, . . . ”* ，*“1.”* ，*“Redistributions of source code must retain . . . ”* ，*“2.”* ，*“Redistributions in binary form must reproduce . . . ”* の 5 文に分割される．

(3) 次のフィルタリングでは，*“1.”* と *“2.”* はキーワード集合に含まれる単語を全く含まないため，これらの文を取り除く．次に，(4) 文とメタライセンス文との照合を行う．文 *“Redistribution and use in source and binary forms, . . . ”* は，メタライセンス文 *“BSDPre:Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met”* と同一なので，この文はメタライセンス文名 BSDPre に置換される．

(5) 得られたメタライセンス文の系列とライセンスルールとの比較を行う．メタライセンス文名の系列 *“AllRights,BSDpre,BSDcondSource,BSDcondBinary, BSDasIs,BSDWarr”* となっており，これは，ライセンスルール *“BSD2:BSDpre, BSDcondSource,BSDcondBinary,BSDasIs,BSDWarr”* とマッチするので，メタライセンス文名の系列は BSD2 に置換される．すなわち，このソースファイルは BSD2 ライセンスと特定される．

2.6 評価実験

2.6.1 ライセンス特定性能

本評価実験では，対象とするソースファイル集合に対し，本ツールと既存のライセンス特定ツールを適用し，ツールの出力とあらかじめ作成された正解集合を比較した．そして，得られた結果を回答率と正答率，所要時間で評価した．

ここでは，本ツールと FOSSology v1.0.0，ohcount v3.0.0rc，oslc v3 を用いた．これらのツールには異なる特徴が存在する．本ツールと Fossology では，ツールの結果として Unknown を出力する場合があるが，他の二つは常に何らかのライセンスを出力する．OSLC はライセンスの一致をパーセントで示している．しかし，我々は他のツールと同条件にするため，OSLC が出力したライセンス名のみを特定した結果としてみなした．

実験対象は以下の手順にて作成した．初めに，Debian5.0.2 から 250 個のパッケージをランダムに選択した．次に，選択した各パッケージから 1 ファイルをランダムに選択した．これらの 250 ファイルの集合を N とよぶ．作成したサンプルは Debian5.0.2 の 80 万ファイルに依存することを考慮すると，信頼度 95% で誤差 ±6.2% である．

評価実験は以下の手順で行われた．初めに，N に含まれる各ソースファイルに対し，各ツールでライセンスを特定した．また，本ツールのライセンス知識の作成を主に行った人とは異なる人が手作業によりライセンスを特定した．本実験ではこの手作業によるライセンス特定結果を正解集合とした．

次に，各ツールから出力された結果と正解集合を比較した．そして，各ツールについて，各ファイルへのライセンス特定結果を以下の 4 つの集合に分類した．これらの集合は互いに重複しない．

C_v **ライセンス名と版が一致** ツールの出力結果が，名前と版まで含めて正解と一致している．例として，正解が GPLv2 の場合，ツールの出力結果が GPLv2 である場合はこのクラスに分類される．一つのソースファイルが複数のライセンスを持つ場合は，ツールの出力結果が正解と一致したときのみこのクラスに分類する．また，ソースファイルにライセンス記述がなく，ツールの出力結果もライセンスなしに相当する結果だった場合もこのクラスに該当する．

C_n **ライセンス名のみが一致** ツールの出力結果が正解と名前のみ一致している．例として，正解が GPLv2 の場合，ツールの出力結果が GPLv3 であればこのクラスに分類される．

I **不正解** ツールの出力結果のライセンス名と正解のライセンス名が一致していない．例として，正解が GPLv2 の場合，ツールの出力結果が MPL v1.1 であればこのクラスに分類される．また，ソースファイルが複数のライセンスを持つ場合は，ツールの出力結果と正解が一部しか一致しないときにはこのクラスに分類する．

U **不明** ツールの出力結果として，不明が出力された場合，このクラスに分類される．

最後に，分類結果から回答率，正答率を求める．回答率，正答率は以下の通りに定義する．

$$\text{回答率} = \frac{|C_v|+|C_n|+|I|}{|N|} \qquad \text{正答率} = \frac{|C_v|}{|N|-|U|}$$

$$\text{正答率（名前のみ）} = \frac{|C_v|+|C_n|}{|N|-|U|}$$

表 2.2 に実験結果を示す．本ツールは他のツールと比較して，回答率が低い．これは数多く不明を出力するからである．一方，名前のみを考慮した場合，名前と版の両方を考慮した場合において，本ツールは正答率では最も高かった．

実験結果から分かるように，本ツールはライセンス名と版の両方をかなり正確に特定することができる．ライセンス名のみであれば FOSSology や ohcount が優れており，その版に興味がないのであればこれらのツールの方が優れている．しかし，複数の版をもつライセンスは版によって条文が異なり，許諾条件も異なるため，ライセンス名だけでは目的を達成できない．

また，本ツールは他のツールに比べて回答率が低かった．これは，誤った結果を報告するよりはライセンスが不明であると報告するほうがよいと我々は考えて設計したためである．なぜなら，ライセンスが未知であるソフトウェア部品をユーザーは安易に調査せずに

表 2.2: 各ツールの評価結果．同一の尺度で最も高かった値を太字で示した．

	本ツール	FOSSo.	ohcount	OSLC
$\|C_v\|$	200	137	83	57
$\|C_n\|$	1	84	137	8
$\|I\|$	6	28	30	185
$\|U\|$	43	1	0	0
回答率	82.8%	99.6%	**100.0**%	**100.0**%
正答率	**96.6**%	55.0%	33.2%	22.8%
正答率（名前のみ）	**97.1**%	88.7%	88.0%	26.0%
所要時間	**22s**	923 s	27s	372s

使用しようとは思わないが，明確にライセンスの特定結果が示されている場合，その結果を信用して安易に使用してしまうことが考えられるためである．

また，本ツールは6つのファイルのライセンスを特定できていない．これらの原因は，6ファイル中5ファイルについてはライセンス文は存在していたが，メタライセンス文の不備によりライセンス文であると認識できなかったためである（メタライセンス文の変更，追加により容易に認識できるようになる）．また，他の1ファイルは通常のライセンスのようにライセンス文の系列としてライセンスを記述するのではなく，コメント文字列中にライセンス名が記述されていただけであったため認識できなかった．しかし，これらのファイルでは誤ったライセンスを特定していることはなかったため，本ツールの結果を信用しても使用者に不具合は生じない．

2.6.2 スケーラビリティ

提案手法のスケーラビリティを評価するため，大規模ソフトウェアに提案手法を適用し，所要時間と回答率を評価した．

評価では，Fedora[8] 11 の SourceDVD[9]に含まれるソースファイルに各ツールを適用し，所要時間と回答率を測定した．対象のソースファイルは 93760 個であり，ファイルサイズの総計は 1.17GB である．実験で使用した計算機の主要な性能は，CPU:Intel Core2Quad 2.4GHz，メモリ：2GB，OS：Ubuntu （Linux Kernel 2.6.28-17）である．

表 2.3 にその結果を示す．この結果から，本ツールは ohcount とほぼ同様に高速にライセンスを特定することができることが分かった．

また，この評価の際，Fedora11 Source DVD に含まれる各パッケージについて，回答率を調査した．調査により得られた度数分布を図 2.4 に示す．グラフから 318 個のパッケージに対して，95%以上の回答率を示した．一方で 5%未満だったパッケージが 28 個あった．また，65%以上の回答率を示したパッケージが全体の 80.1%（457 個）を占めていた．

2.6.3 ライセンス知識の表現法

ライセンス知識の表現法の違いによる影響を評価するため，この評価では，各ツールがどのようにライセンス知識を表現しているか調査したうえで，いくつかのライセンスに対し，ライセンス知識のサイズがどのように異なるのか調査した．実験では，本手法を実装したツール，FOSSology，ohcount，OSLC，ASLA を対象とした．本調査におけるライセンス知識のサイズとは，あるライセンスに対するライセンス記述，その他のメタデータ等

表 2.3: スケーラビリティ評価の結果

	本ツール	FOSSo.	ohcount	OSLC
所要時間	172min	5258min	168min	213min

[8]http://fedoraproject.org

[9]http://fedora.mirror.iweb.ca/releases/11/Fedora/source/iso/

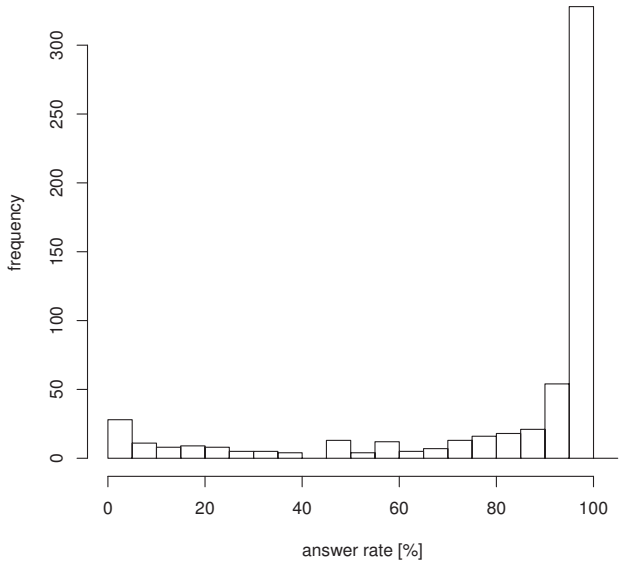

図 2.4: 各パッケージの回答率に基づくヒストグラム

も含む．ohcount はライセンス知識がメソッド呼び出しによりハードコーディングされている．このメソッドは引数としてライセンスの略称，ライセンスのページへの URL，ライセンス名，メタライセンス文を用いる．本実験では，これらのメソッド呼び出しで用いられている引数全体をライセンス知識とした．ASLA については実装が公開されていないため，不明である．

調査結果を表 2.4 に示す．表中，本ツールのライセンス知識のサイズを”（ライセンスルールの合計サイズ）＋（メタライセンス文の合計サイズ）”と表現した．ohcount は各ライセンスに対して 1 行程度の正規表現を用いており，最もライセンス知識のサイズが小さかった．ALSA も同様に正規表現を用いているが，そのサイズは不明である．参照文字列を用いる OSLC や FOSSology はライセンスへの参照を示す記述だけでなく，ライセンスの条項全文をライセンス記述として使用している場合があり，サイズが最も大きくなった．また，FOSSology は各ライセンスに対して複数のライセンス知識を持っているため，OSLC よりもサイズが大きかった．本ツールは 2 層のライセンス知識を用い，各ライセンスに対し複数のライセンス知識を持っているため，ohcount のライセンス知識よりもライセンス知識のサイズが大きかった．

本ツールではライセンス知識の表現方法として 2 層のライセンス知識を用いている．そのため，既存のメタライセンス文を組み替えることで新しいライセンスルールを容易に定義できる．また，既知のメタライセンス文に類似した未知のライセンス文に対応するために正規表現を修正する場合でも，ライセンスルールに影響を与えない．

2.7 まとめと今後の課題

本章では，階層的ライセンス知識を用いたライセンス特定ツールを提案した．大規模オープンソースソフトウェアを対象とし，ライセンス特定の課題を挙げ，課題を解決するために必要なライセンス特定ツールの要件を述べた．そして，ライセンス特定ツールの要件に基づくライセンス特定ツールの設計について述べ，実装を行った．

評価では，大規模オープンソースソフトウェアを対象に特定の精度とスケーラビリティについて本ツールと従来手法との比較を行った．その結果，本ツールは従来手法と比べ高い精度で版も含めてライセンスを特定することができ，また，高速に処理することができた．さらに，大規模オープンソースソフトウェアに含まれていたパッケージのうち，大半のパッケージにおいてパッケージに含まれるソースファイルの95%以上のソースファイルのライセンスを特定できていた．

今後の課題として，より多くの調査に基づくメタライセンス文やライセンスルール，キーワードの洗練がある．また，大規模オープンソースソフトウェアを対象としたライセンスに関する定量的な調査研究も重要な課題である．

表 2.4: 既存手法と提案手法のライセンス知識のサイズ（文字数）

	本ツール	FOSSo.	ohcount	OSLC	ASLA
ライセンス	2 層/正規表現	単層/参照文字列	単層/正規表現	単層/参照文字列	単層/正規表現
BSD3	267+4032	3223	147	1930	不明
Apachev2	147+852	未実装	143	11361	不明
X11	47+1766	1364	125	未実装	不明
MPLv1_1	15+191	24293	151	23650	不明

第3章　オープンソースソフトウェアにおけるライセンス分布の調査

3.1　はじめに

ソフトウェア開発コストを削減する手段の一つとして，オープンソースソフトウェア（Free and Open Source Software, 以下 FOSS）を改変し，新たなソフトウェアを作成する方法がある．

FOSS の再利用を行う場合，開発者は一般的に各ソースコードファイルにコメントとして記述されているソフトウェアライセンスを注意深く読み，記述されている要求や制約を理解し，これらに従わなくてはならない．多くの場合，FOSS は Open Source Initiative[1] がオープンソースの定義を満たすソフトウェアライセンス（以下，ライセンス）として承認したオープンソースライセンスの下で配布されている．現在，Open Source Initiative は 67 種類のオープンソースライセンスを承認しており，これらのライセンスは有名な FOSS でよく使用されている．

しかし，大規模な FOSS には様々なライセンスのファイルが含まれている可能性がある．それを理解せずに FOSS を改変した場合，ライセンス違反につながる恐れがある．Ruffin ら [49] は「FOSS を用いる際の深刻なリスクの一つは，ライセンスがチェックできていない多くの FOSS ファイルを開発者が利用することである」と述べている．

そのため，一つの FOSS に含まれるライセンスがどのように分布し，また，その分布がどのように変わりうるかを理解することで，上記の潜在的なライセンス違反を防ぐことができるのではないかと考える．ライセンスの分布を調査する基盤として，ソースコードファイルのライセンスを自動的に解析，特定するツールが存在しており，それらのツールを用いた，FOSS におけるライセンスの解析に関する既存研究が存在する [24, 28, 30, 31]. しかしながら，これらの論文では，ライセンスがどのように分布し，その分布がどのように変化するかについて，複数のアプリケーションを対象とした定量的な調査は行われていない．

本章では，第 2 章で提案したライセンス特定手法の実装である Ninka を用いて，複数の大規模な FOSS を対象としたライセンス分布の調査を行った [43]．我々は FreeBSD，OpenBSD，Eclipse，ArgoUML の各バージョンを解析対象とした．この調査では解析結果を基に，主にそれらのシステムの各リリースにおいて使用されているライセンスがどのように分布しているか，ソフトウェアの進化の過程においてどのように各ライセンスごとのファイル数が変化したか，また，顕著に表れたライセンスの変化がどのような影響を持つのか調査した．

本研究における主な貢献は以下の事実を発見したことである．

[1] http://www.opensource.org/ (最終アクセス日 2010/3/31)

- 調べた対象の FOSS においては，FreeBSD や OpenBSD のようなオペレーティングシステムで用いられているライセンスの数は Eclipse や ArgoUML のような特定のアプリケーションのライセンスの数よりも多い．
- ライセンスはあるリリースバージョンにおいて劇的に変化しうる．
- 非オペレーティングシステムのライセンスはオペレーティングシステムのライセンスと比べ，急激に変化が起こる場合がある．
- ライセンスが緩いライセンスに変化する場合と，厳しいライセンスに変化する場合の両方が存在する．

以下，本章の構成について述べる．3.2 節では本章で用いる用語について説明する．3.3 節ではライセンス分布を解析するための，調査項目を設定する．3.4 節では 4 つの主要な FOSS に対する解析結果を示し，3.3 節で設定した調査項目への回答を述べる．3.5 節では，解析結果の妥当性へ影響を与える事項について述べる．3.6 節では関連研究を示し，3.7 節ではまとめと今後の課題について述べる．

3.2 用語

ソフトウェアライセンス（以下，**ライセンス**）はソフトウェアの作者により定められたソフトウェアのユーザーへの指示の集合である．ライセンスはソフトウェアプロダクト全体だけでなく，ソフトウェアプロダクトを構成する個々のソースコードファイルについても決まっている．ライセンスの中には変更を受け，複数の版が存在するものもある．例として，GNU General Public License(GPL) には version1, version2, version3 が存在する．また，GPL の場合，*“GPL version 2 or later”* とライセンスの指示に記載されている場合，version2 以降の最新の GPL を適用できる．本章では，個々のライセンスの法的問題については議論しない．

表 3.1 に本章で使用するライセンスの名前と略称を示す．ライセンスが複数の版を持つ場合，特定の版を示すために接尾辞 v<number> を用いる．また，*“or later”* を接尾辞+で表現する．例として，GPL の version2 or later を GPLv2+と略する．

ソースコードファイルはコメントとプログラムコードを含んでいる．コメントはプログラムコードの説明や，ライセンスの指示，その他の目的のために使用されている．ライセンスの指示は一般的に複数の自然言語の文からなっている．

ライセンスの混在とは，一つの FOSS を構成するソースコードファイルのライセンスが２種類以上存在することを意味する．ライセンスの混在はその FOSS を利用して新たなソフトウェアを構築し，FOSS のライセンスを考慮して新たに開発するソフトウェアのライセンスを決定した場合，ソフトウェアのライセンスとソースコードファイルのライセンスが不整合を起こし，結果として，ライセンス違反につながる可能性があるという点で問題となる．

ライセンス比率は対象に含まれる全てのファイルの数に対し，あるライセンスを持つファイルが占める比率を意味する．

表 3.1: 本章で使用する一般的なオープンソースライセンスとその略称

Abbrev.	Name
Apache	Apache Public License
BSD4	Original BSD, also known as BSD with 4 clauses
BSD3	BSD4 minus advertisement clause
BSD2	BSD3 minus endorsement clause
CPL	Common Public License
CDDLic	Common Development and Distribution License
EPL	Eclipse Public License
GPL	General Public License
LesserGPL	Lesser General Public License (successor of the Library GPL, also known as LGPL)
LibraryGPL	Library General Public License (also known as LGPL)
MIT/X11	Original license of X11 released by the MIT
MITold	License similar to the MIT/X11, but with different wording

3.3 調査項目

本研究では,「ライセンスの分布はどのような特徴を持つのか」を明らかにするため調査を行う．目的を達成するため，以下に示す３つの調査項目を設定した．

RQ1: 個々の FOSS ではどの程度ライセンスが混在しているのか

RQ2: ライセンスの分布は進化の過程でどのように変化していくのか

RQ3: ライセンスの分布の変化は FOSS にどのような影響を与えるのか

3.4 調査結果

我々は上記の調査項目に回答するため，実験を行った．実験では，解析対象として主要な FOSS である，FreeBSD，OpenBSD，Eclipse，そして ArgoUML を用いた．解析対象のソフトウェアとその特徴を表 3.2 に示す．ここで，リリースバージョンについては，版管理システムに保存されているログやそれらの FOSS のウェブサイトにおけるリリース情報に基づいて調査した．FreeBSD や OpenBSD は，それぞれのカーネルのみ調査対象として用いた．Eclipse は Eclipse プラットフォームプロジェクト以下にある全モジュールを用いた．

本実験では，ソースコードファイルのライセンスを特定するツールとして第２章で提案したライセンス特定手法の実装である Ninka[30] を用いた．Ninka はソースコードファイルを入力とし，入力されたファイルに対して特定したライセンス名，つまり BSD4 や GPLv2+ と出力する．Ninka はソースコードファイルの先頭にあるコメントにある各英文を特定し，よく使用されるライセンスのライセンスの指示に含まれる各文に対応するメタライセンス文の集合と比較し，どの英文がライセンスの指示に含まれる文のうちどの文と一致するかを特定する．その後，Ninka はそれらの文の配置とよく使用されるライセンスの文配置パターンであるライセンスルールとマッチングを行い，ライセンスを特定する．現在，Ninka は 427 のメタライセンス文や 126 のライセンスルールをライセンス知識としてデータベー

表 3.2: 本研究における解析対象

	FreeBSD	OpenBSD	Eclipse	ArgoUML
バージョン	2.2-8.0	2.0-4.7	2.0-3.5.2	0.8.1-0.31.1
リリース日	1994/11 -2009/11	1996/10 -2010/5	2002/6 -2009/9	2000/10 -2010/6
種類	OS kernels	OS kernels	SDE platform	UML Design Tools
リリース数	45	28	25	79
ファイル数 (最古-最新)	627-3490	987-3314	11419-35880	686-2208
版管理システム	CVS	CVS	CVS	Subversion

スに持っている．Ninka は 110 種の異なるライセンスを 93%の精度で特定し，1 分間に 600 ファイル以上処理することができる [30]．また，Ninka は NONE(ライセンスに関係のある文を含まないファイル) や UNKNOWN(ライセンスに関する文を含むが，ライセンスルールと一致しなかったファイル) も出力する．

実験では，以下の方法で実験対象とした各 FOSS の各リリースバージョンを解析した．初めに，解析対象となるリリースバージョンのソースツリーを版管理システムから取得した．ここで，我々は C (*.c), C++(*.cpp, *.c++, *.cxx, *.cc), Java(*.java), Python(*.py), Perl(*.pl, *,pm), Emacs lisp(*.el) and Sawfish scripting language(*.jl) で記述されたファイルを解析対象として用いた．次に，取得したファイルのライセンスを Ninka を用いて特定した．

3.4.1 RQ1：個々の FOSS ではどの程度ライセンスが混在しているのか

ライセンスの混在が実際にどのように起こっているかを確認するため，表 3.2 に示した各 FOSS の最新版において各ファイルのライセンスを調査した．

調査結果を表 3.3 に示す．結果から，全ての FOSS において，ソースコードファイルのライセンスは 1 種類ではなく，ライセンスの混在が起こっていたことが分かった．オペレーティングシステムである FreeBSD や OpenBSD に含まれるファイルのライセンスは多様である．また，ソフトウェア全体のライセンスと同一のライセンスのファイルがソフトウェア全体のファイル数に対する割合はそれぞれ 38.5%，13.6%であった．そのため，これらの FOSS ではソフトウェア全体のライセンスとソースコードファイル個別のライセンスが異なっている場合が多いといえる．一方，オペレーティングシステムでない Eclipse や ArgoUML ではライセンスの種類は FreeBSD や OpenBSD と比べて少ない．ソフトウェア全体のライセンスと同一のファイルのライセンスの割合はそれぞれ 68.3%と 98.5%と FreeBSD や OpenBSD と比べて高い．

以上より，RQ1 に対する回答は”調査したすべてのアプリケーションについて，ライセンスの混在があった．また，オペレーティングシステムに比べ，非オペレーティングシステムは混在しているライセンスの数が少なく，ソフトウェア全体のライセンスと同じライセンスのファイルは多い．”となる．

表 3.3: 最新版におけるライセンス分布

	FreeBSD	OpenBSD	Eclipse	ArgoUML
Release Version	8.0	4.7	3.5.2	0.31.1
ファイル数	3490	3314	35880	2208
ライセンスの種類	81	91	11	5
ソフトウェア全体のライセンス	BSD2	MITOld1[2]	EPLv1	EPLv1
ソフトウェア全体と ライセンスが同一のファイル数	1346	451	24507	2177
（割合 [%]）	38.5	13.6	68.3	98.5

3.4.2 RQ2：ライセンスの分布は進化の過程でどのように変化していくのか

RQ2 への回答を行うため，RQ2 を以下に示す 2 つの副調査項目に分解した．

RQ2.1: オペレーティングシステムのライセンス分布はどのように変化していくか

RQ2.2: 調査対象において，オペレーティングシステムのライセンス分布は非オペレーティングシステムのライセンス分布と比べ，変化の傾向がどのように異なるのか

RQ2.1：オペレーティングシステムのライセンス分布はどのように変化していくか

RQ2.1 に回答するため，FreeBSD と OpenBSD について各バージョンにおけるライセンス比率と，各バージョン間でのライセンス比率の変化を調査した．

図 3.1 は FreeBSD の各バージョンにおけるライセンス比率，図 3.2 は OpenBSD の各バージョンにおけるライセンス比率を表す．これらのグラフにおいて，X 軸はリリースバージョンを表し，Y 軸はファイル数を表す．各層は各ライセンスに対応する．我々は各グラフにおいて，調査対象とした範囲の最古の版と最新の版でファイル数が多かった 5 つのライセンスを選択し，示した．選択しなかったライセンスのファイル数は Others としてひとまとめにした．次に，図 3.3 は FreeBSD のファイル数の変化を，図 3.4 は OpenBSD のファイル数の変化を示す．グラフの X 軸はリリースバージョンを表し，Y 軸はあるバージョンとそのひとつ前のバージョン間でのファイル数の差を表している．ただし，この解析において，NONE や UNKNOWN，Others に分類されたファイルは数えていない．

得られた結果は以下のとおりである．

- 図 3.1 より，FreeBSD の場合, 進化するに従い，BSD4 ライセンスは減っている．一方，BSD2 や BSD3 は増えている．これは BSD ライセンスの作者であるカリフォルニア大学バークレー校のポリシー変更によるものである[3]．ライセンス比率の変化 (図 3.3) からも，BSD4 の減少は定期的に起こっており，BSD2 や BSD3 はそのような減少に対応して増えていることがわかる．特にライセンスの分布の大きな変化が発生

[2]実際には OpenBSD のライセンスは ISC ライセンスである（http://www.openbsd.org/policy.html）．しかし，Ninka が MITOld1 として ISC ライセンスを認識しているため，MITOld1 をソフトウェア全体のライセンスとしてみなした．

[3]ftp://ftp.cs.berkeley.edu/pub/4bsd/README.Impt.License.Change (最終アクセス日 2010/3/31)

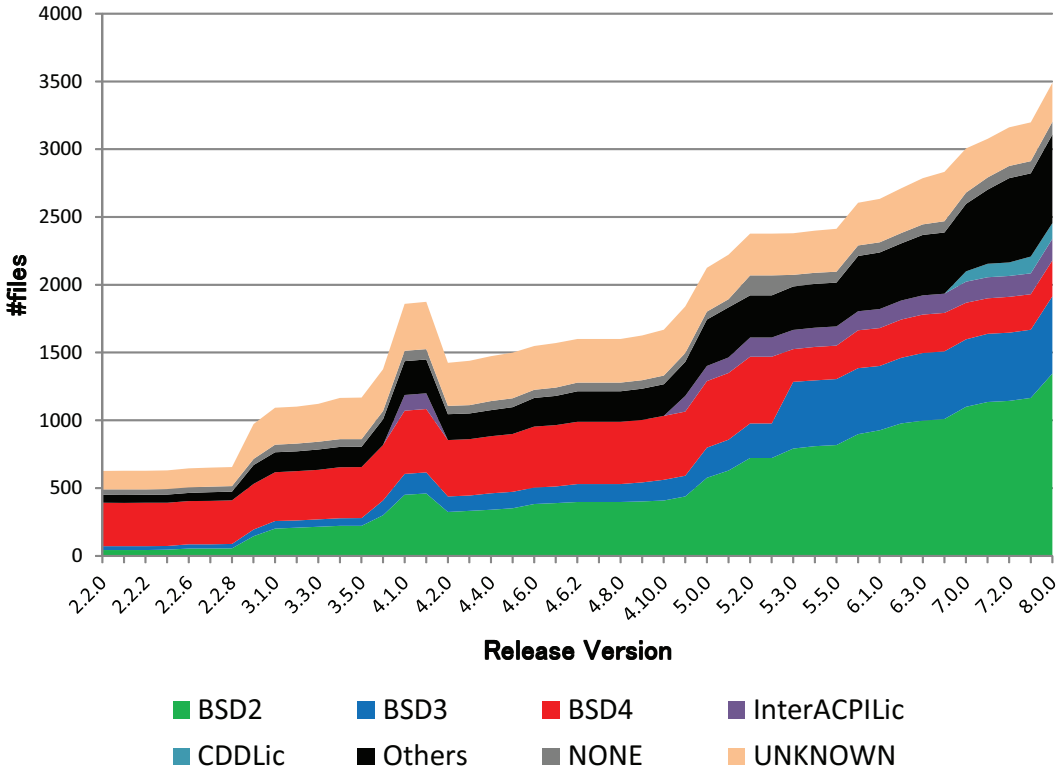

図 3.1: FreeBSD のライセンス比率

しているリリースバージョン 5.2.1 と 5.3 の変化を詳細に解析した結果を表 3.4 に示す．この表では，行が 5.2.1 でのライセンス，列が 5.3 でのライセンスを意味する．この表から，BSD4 のファイルのうち，233 ファイルが BSD3 に変更されていたことが分かる．また，23 ファイルが削除されていた．一方，新規に BSD2 のファイルが 97 ファイル，BSD3 のファイルが 24 ファイル追加されていた．以上から，BSD3 ファイルの増加の原因は，おおむね BSD4 ファイルのライセンスの変更によるものであり，BSD2 ファイルの増加は新規に作成されたことによると言える．

- OpenBSD の場合にも同様の傾向が見られる（図 3.2）．ライセンス比率の変化 (図 3.4) から，BSD4 が減った分だけ BSD2 や BSD3 のファイルが増加していることがが分かる．特にライセンスの分布の大きな変化が発生しているリリースバージョン 3.3 と 3.4 の間を詳細に解析した結果を表 3.5 に示す．この表では，行が 3.3 でのライセンス，列が 3.4 でのライセンスを意味する．この表から，BSD4 ファイルのうち，500 ファイルが BSD3 ファイルへ，107 ファイルが BSD2 ファイルへ変更されていたことが分かる．また，16 ファイルが削除されていた．

以上より，RQ2.1 に対する回答は ”２つのオペレーティングシステムにおけるライセンス進化の詳細な解析の結果により，FOSS の進化に沿って，定期的にライセンスの分布に

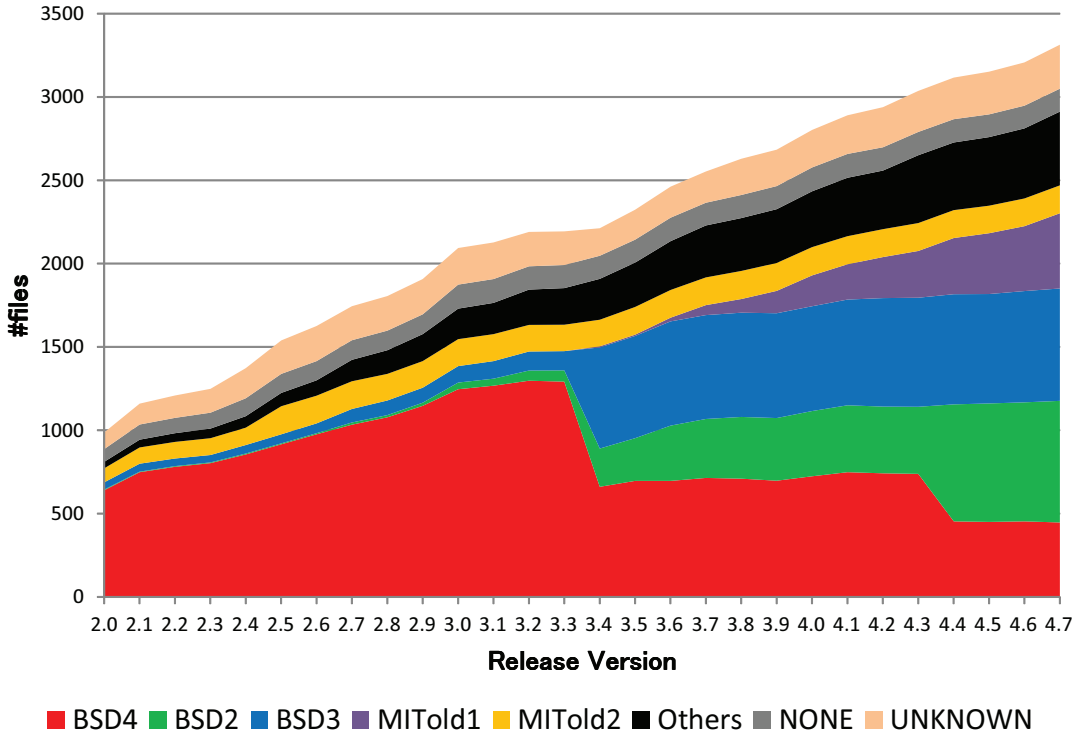

図 3.2: OpenBSD のライセンス比率

大規模な変化が起きていることが分かった．”となる．

RQ2.2 : 調査対象において，オペレーティングシステムのライセンス分布は非オペレーティングシステムのライセンス分布と比べ，変化の傾向がどのように異なるのか

RQ2.2 に対して答えるため，Eclipse や ArgoUML のライセンス比率とその変化を調べ，RQ2.1 で調査した FreeBSD や OpenBSD の場合と比較した．

図 3.5，図 3.6 はそれぞれ Eclipse, ArgoUML の各リリースのライセンス比率を示す．図 3.7，図 3.8 は Eclipse，ArgoUML の各リリース間でのファイル数の変化を示す．

興味深い発見は以下の通りである．

- Eclipse の場合 (図 3.5)，既知のライセンスの大半が CPLv0.5 から CPLv1.0 に変化し，その後，EPLv1.0 に変化している．これらの変化は図 3.5 でははっきりと表れており，特に，図 3.7 により，変化している部分とそうでない部分がはっきりと分かれている．一方，上記で言及された FreeBSD や OpenBSD の変更はややあいまいであり，より古いライセンスがより長く残っている．これは，開発管理の違いのためである．FreeBSD や OpenBSD は多くの提供されたソースコードやソースコードファイルを

表 3.4: FreeBSD 5.2.1 から 5.3 の間でのライセンスの変化

		バージョン 5.3					
	ライセンス名	存在しない	BSD2	BSD3	BSD4	InterACPILic	CDDLic
	存在しない	-	**97**	**24**	13	0	0
	BSD2	31	689	0	0	0	0
バージョン	BSD3	18	2	235	0	0	0
5.2	BSD4	**23**	0	**233**	229	0	0
	InterACPILic	0	0	0	0	141	0
	CDDLic	6	0	0	0	0	0

表 3.5: OpenBSD 3.3 から 3.4 の間でのライセンスの変化

		バージョン 3.4					
	ライセンス名	存在しない	BSD4	BSD3	BSD2	MITOld1	MITOld2
	存在しない	-	10	7	13	2	0
	BSD4	**16**	648	**500**	**107**	0	0
バージョン	BSD3	0	1	97	10	5	0
3.3	BSD2	0	0	0	68	0	0
	MITOld1	0	0	0	0	0	0
	MITOld2	0	0	0	0	0	159

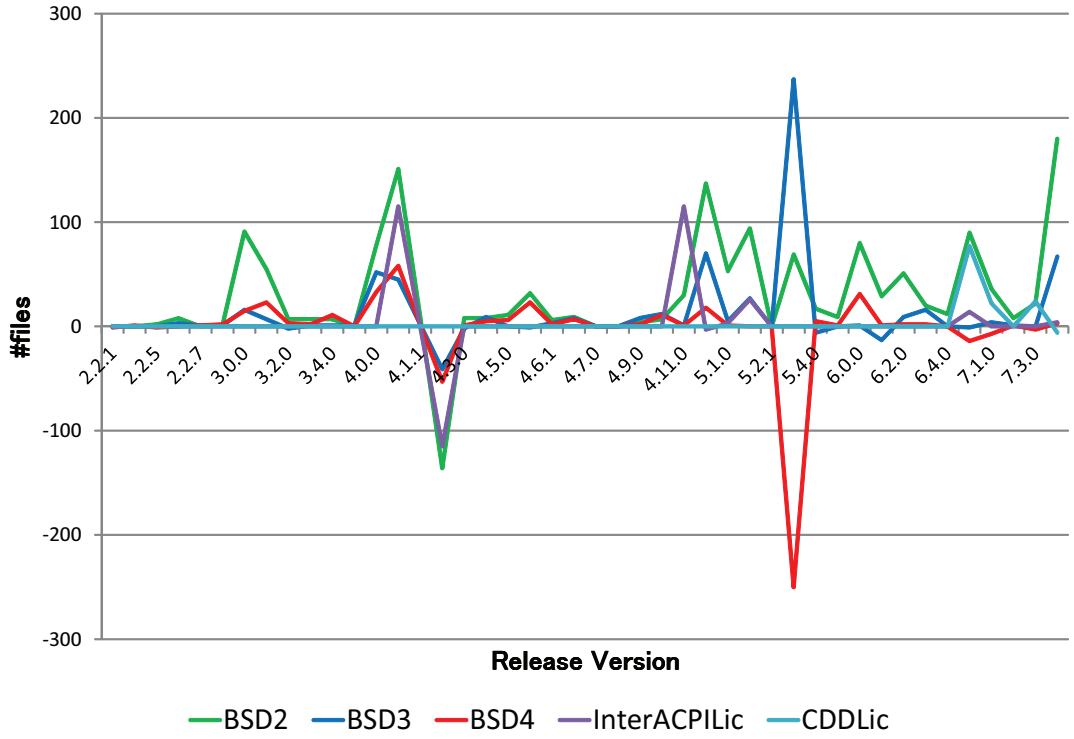

図 3.3: FreeBSD におけるファイル数の変化

用いて開発されてきた．しかし，Eclipse は Eclipse Development Process[26] に従って，より厳密な管理のもとで開発されてきた．

- AlgoUML の場合 (図 3.6), 進化の過程で多くの UNKNOWN ファイルが存在している．さらに解析を行った結果，それらはほとんど BSD ライセンスに似たライセンスである[4]が，一般的によく使用される BSD4 や BSD3，BSD2 といったものとは異なっていた．また，Eclipse の場合と同様，ライセンスの大規模な変化が図 3.6 や図 3.8 にはっきりと表れている．

以上より，RQ2.2 への回答は ”オペレーティングシステム (FreeBSD, OpenBSD) におけるライセンスは非オペレーティングシステム (Eclipse, ArgoUML) と比較して，やや多様であり，かつ緩く管理されている．非オペレーティングシステムにおいて，少数のライセンスがほぼすべてのファイルをカバーしており，それらのライセンスがシステム全体への強い管理によって他のライセンスに劇的に変更されることもある．” となる．

[4]新たにメタライセンス文やライセンスルールをデータベースに追加することにより，Ninka はそれらを特定することができる．我々は紙面の都合上，オリジナルデータのみ示している．

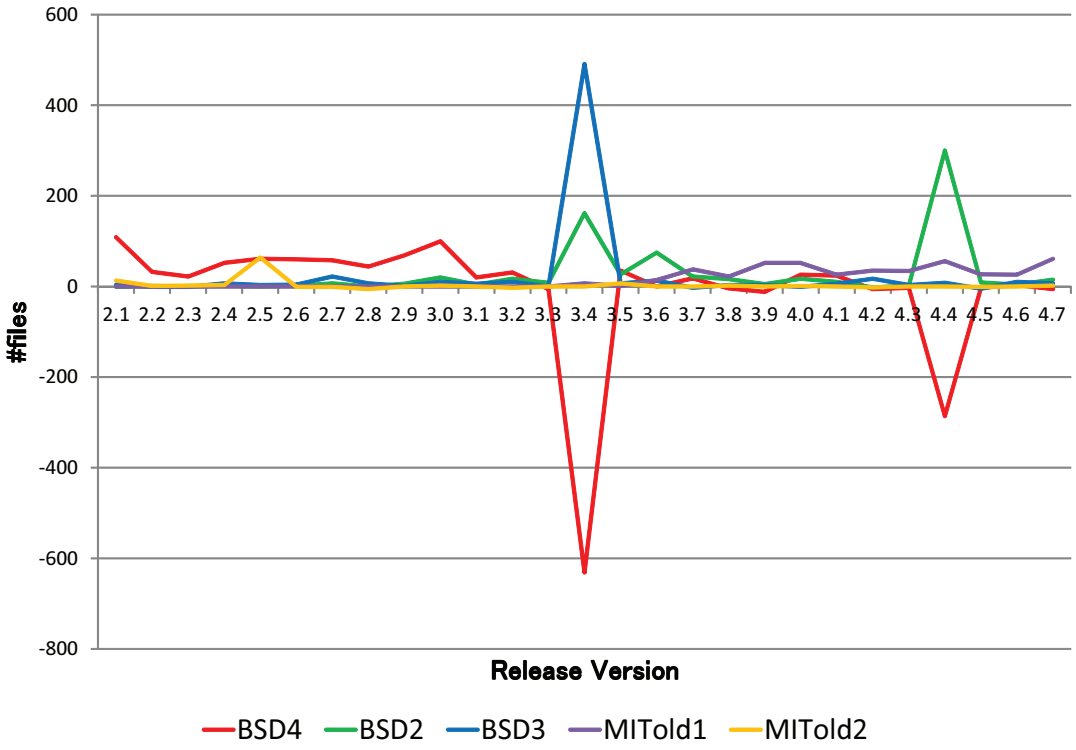

図 3.4: OpenBSD におけるファイル数の変化

RQ2 に対する回答

服調査項目 RQ2.1, 2.2 に対する回答から RQ2 に対する回答は以下の通りとなる．”調査対象のオペレーティングシステムにおけるライセンスの分布は定期的かつ大規模に変更されている．一方，非オペレーティングシステムではオペレーティングシステムと比べて急な変化を示している．”

3.4.3 RQ3：ライセンスの分布の変化は FOSS にどのような影響を与えるのか

RQ2 に対する調査において，大規模なライセンス比率の変化が確認できた．実際に起こった変化を基に，これらの変化がどのように影響があるかを調査した．

- FreeBSD(図 3.1) や OpenBSD（図 3.2）の場合では，BSD4 ライセンスのファイルが減っている一方で，BSD2 や BSD3 は進化するにしたがって増えている．この変更により，宣伝資料に謝辞を含めなくてよくなっている．それゆえ，この変更はライセンスの条項をゆるくするものである．
- Eclipse の場合 (図 3.5)，既知のライセンスの大半が CPLv0.5 から CPLv1.0 に変化し，その後，EPLv1.0 に変化している．CPLv1.0 と EPLv1.0 間の違いは特許に関連した条項の削除である．そのため，この変更はライセンスの条項をゆるくするものである．

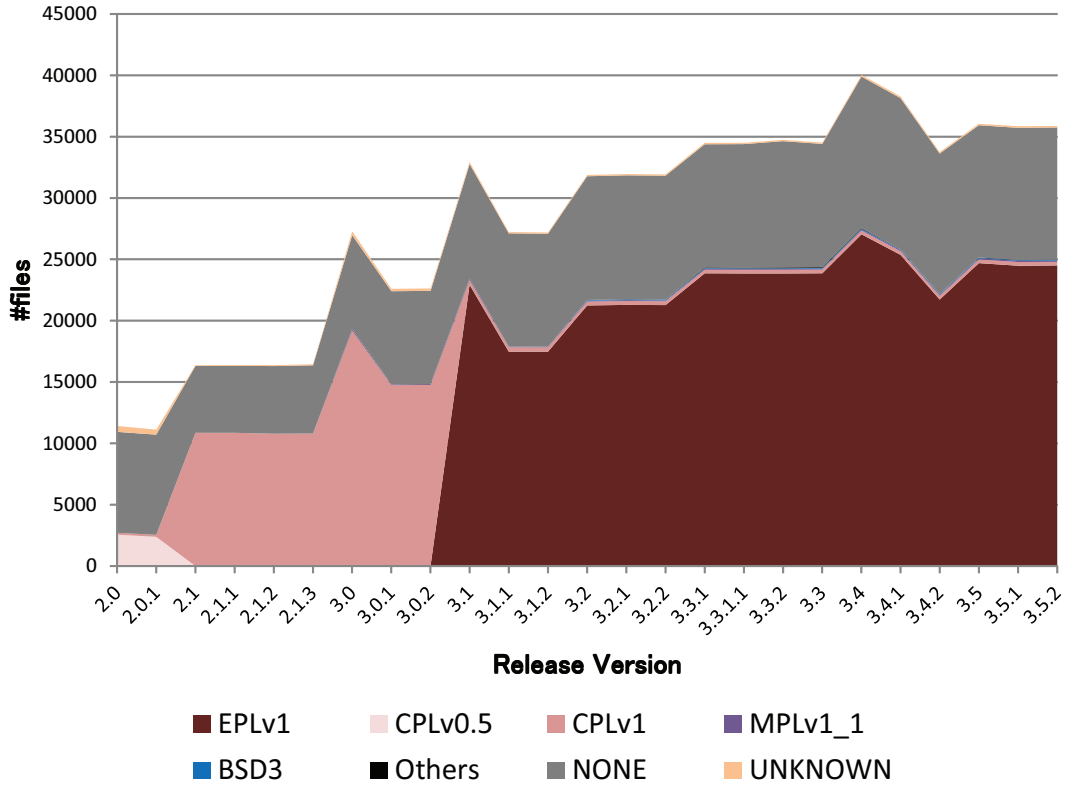

図 3.5: Eclipse のライセンス比率

- AlgoUML の場合 (図 3.6), 進化の過程で BSD ライセンスに似たライセンスのファイルが減少し，EPLv1.0 のファイルが増加していた．BSD ライセンス下にあるファイルを修正するとき，我々はほかのライセンスのもとでそれを再配布できる．しかし，EPLv1.0 の下にあるソースコードファイルを修正するとき，他のライセンスの下では配布することができない．それゆえ，この変更はライセンスの条項を厳しくするものである．

以上より，RQ3 への回答は，”大規模なライセンスの変更により，多くのソースコードファイルのライセンスが変更されるが，厳しいライセンスに変更される場合と緩いライセンスに変更される場合の両方が存在した．”となる．

3.5 妥当性

調査項目への回答の妥当性に影響を与えうる要因として，Ninka から得られた結果の正しさがある．Daniel et.al.[30] によると，Ninka はライセンスを特定できたソースファイルのうち，96.6%に対し，正しくライセンスを特定できている．我々はこれは十分に高く，そのため，Ninka のライセンス特定の誤りは 3.4 節でのライセンス比率の議論に影響を与えないと思われる．

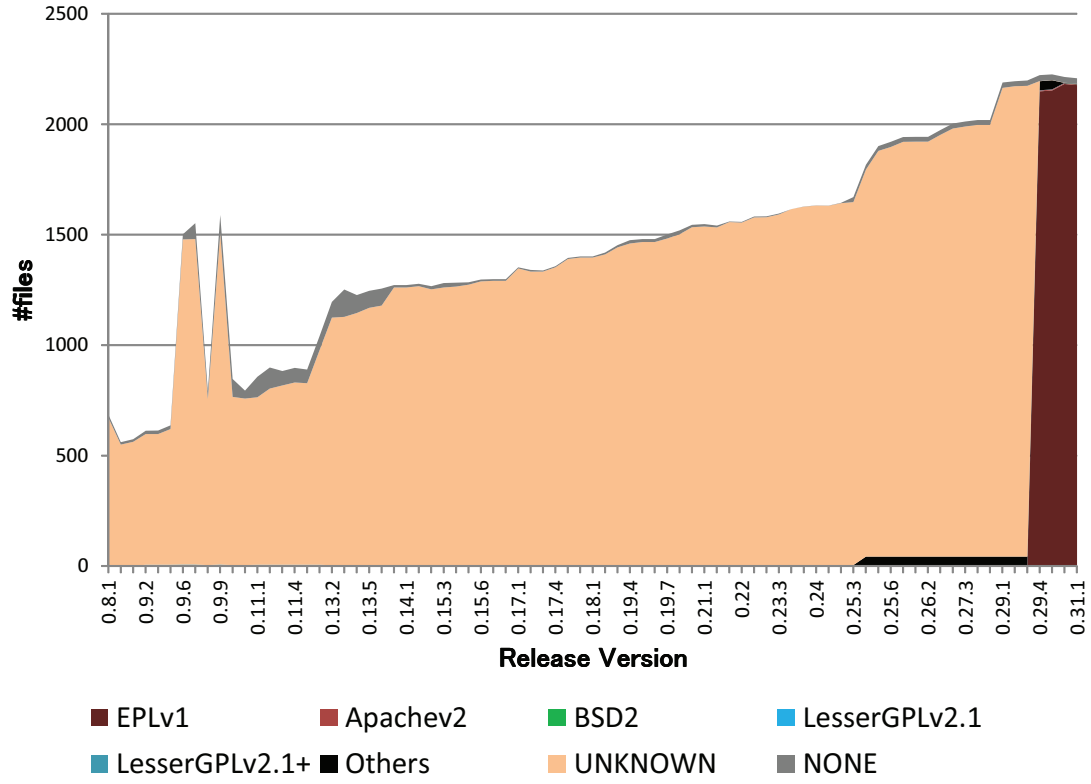

図 3.6: ArgoUML のライセンス比率

他の要因として，アプリケーションの選択がある．我々は FreeBSD や OpenBSD を解析対象のオペレーティングシステムとして用いた．これらは 386BSD という同じ源流をもつが，これらのプロジェクトは異なる開発ポリシーの下，十年ほど前に別々に開始されている（FreeBSD は 1991 年，OpenBSD は 1995 年）．そのため，我々はそれらのシステムには関連があるが，異なったライセンスポリシーを持つ異なったオペレーティングシステムであると考えている．

我々は解析対象として，４つのシステムのみを用いた．そのため，本研究結果の特性が他の FOSS に当てはまるか，一概に言えない．しかしながら，ここで対象とした FOSS プロジェクトと同様の開発形態を取る FOSS のプロジェクトも多く，その場合も同様なライセンスの特性が成り立つ場合も多いのでないかと考えられる．今後，より多くの対象を分析していく予定である．

3.6 関連研究

Di Penta ら [24] はライセンス条項における変更を追跡するための方法を提案している．この研究ではライセンスが頻繁に，かつ何度も変更されていることを示している．この研究では，解析対象の各版を解析し，それらの版の間で現れるライセンス変更の多くのパターンを特定している．一方，我々の研究では，各システムの多くのリリース間でライセ

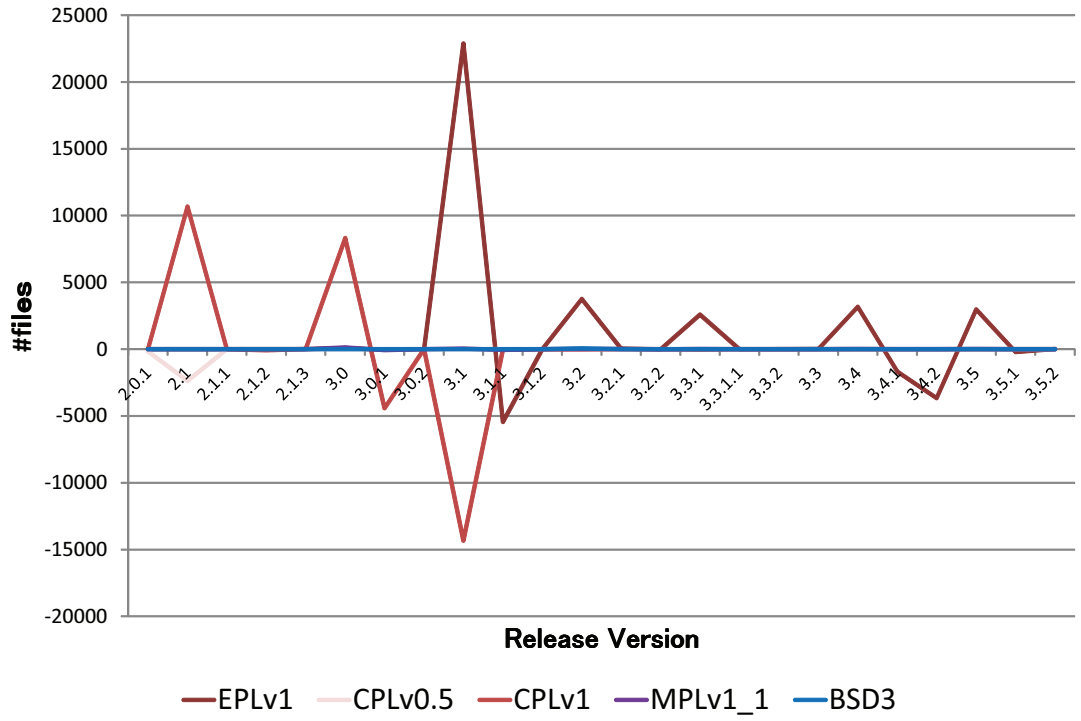

図 3.7: Eclipse におけるファイル数の変化

ンスの変更が起こることを示している．我々は２つのリリース間で大規模な変更が起きていることを示した．加えて，彼らの研究では言及されていない ArgoUML の大規模ライセンス変化を発見した．我々の研究では，そのようなパターンが進化の間を通して起こり，そのライセンス比率の変化は FOSS の種類に基づき，劇的もしくはゆっくりと起こる事を示している．

ライセンス比率を扱った既存研究として，Gobeille[31] の研究がある．Gobeille は bSAM アルゴリズムを用いてソースコードファイルの各ライセンスを特定するライセンス特定ツール FOSSology を提案している．彼は例としてそれを abiword-2.6.4 に適用し，ライセンス比率を示している． Black Duck Software[5]は Black Duck software knowledgebase に従って，FOSS の開発プロジェクトにおいてもっともよく使用されている上位 20 のライセンスを公表している．我々の研究は多くのリリースバージョンを持つ大規模ソフトウェアシステムに焦点を当てているという点でこれらの研究とは異なっている．

また，ライセンスの不整合を扱った既存研究もある．Alspaugh ら [17] はライセンスの条項を ⟨actor，operation，action，object⟩ のタプルで表現する方法とそれを用いて義務の伝搬と衝突を計算する手法を提案している．German ら [29] はソフトウェアライセンスを形式的に定義し，ライセンスの不整合を解決するためのパターンを提案した．German ら [28] はソフトウェアパッケージにおいて起こるライセンス不整合の理解を支援し，Fedora 12 のバイナリパッケージにおけるライセンス不整合を検査するための手法を提案してい

[5]http://www.blackducksoftware.com/oss/licenses#top20

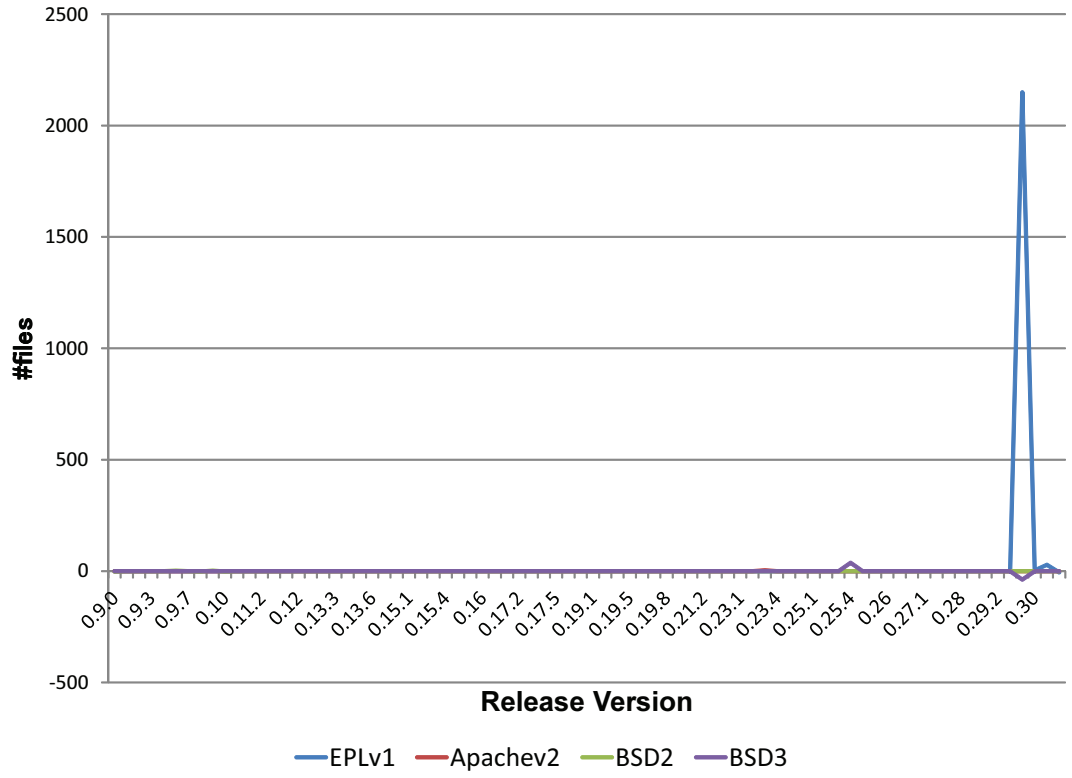

図 3.8: ArgoUML におけるファイル数の変化

る．我々の研究により得られた結果はライセンス不整合の危険性やライセンス特定の重要性をさらに強調する可能性がある．

3.7 まとめ

本章では大規模 FOSS におけるライセンスの分布について調査した．この調査により，各 FOSS において，ライセンスが混在していることを示した．次に，対象とするオペレーティングシステムにおけるライセンスの分布における変化は定期的，かつ連続的に発生することが分かった．また，我々はオペレーティングシステムのライセンス分布における変化の傾向は Eclipse や ArgoUML のような非オペレーティングシステムの傾向と異なることも示した．さらに，ライセンスの分布に変化が起きる際，ソースコードファイルのライセンスが厳しいライセンスに変化する場合と，緩いライセンスに変化する場合の両方があることを示した．

今後の課題として，ライセンスの分布を FOSS の構造と関連付けて調査する必要があると考えている．たとえば，ディレクトリ構造上にライセンスをマッピングし，どの部分にどのライセンスのファイルが存在しているかを調べたい．また，そのためのツールの作成も重要な課題である．

第4章　コードクローンメトリクスに基づくソースコード再利用判定閾値の決定手法

4.1　はじめに

近年，開発コストの低減や納期の短縮を目的として，オープンソースソフトウェア（OSS）を再利用して開発を行うことがある．OSS のソースコードを用いて開発を行う場合，OSS のライセンスに注意する必要があるが，オフショア開発など，開発を外注することにより，開発委託元が意図しない OSS の再利用が発生し，ライセンス違反を犯してしまうケースが報告されている [13, 14, 15, 16]．このようなライセンス違反を予防するため，ソフトウェア開発の委託元には，ソフトウェアの出荷前にソフトウェアの再利用が存在しないことを確認すること，もしくは，ソフトウェアの再利用が存在する場合には，どの部分が再利用が行われた部分であるか確認することにニーズがある．これらのニーズを満たすため，ソースコード再利用検出の技術が必要とされている [2, 10]．

ソースコード再利用検出の既存研究として，ファイル間に含まれる一致または類似したコード片（コードクローン）を用いた手法と，バイナリファイルからソースコード再利用検出を行うソフトウェアバースマークがある．

ソースコード再利用のコードクローンに基いた検出手法として，JPlag[48] が存在する．JPlag は，2 つのソースコード間に含まれるコードクローンの割合（コードクローン含有率）を用いてソースコード間の類似度を算出し，ソースコード再利用検出を行う．ただし，JPlag はソースコードの一部に再利用があるような部分的なソースコード再利用の場合，類似度が小さくなり，部分的なソースコード再利用検出は難しいという問題点がある．

2 つ目に，バイナリファイルからソースコード再利用検出を行うソフトウェアバースマークという既存研究が存在する．ソフトウェアバースマークとは，対象となるソフトウェアが持つプログラムの特徴量を抽象化して表現したものである．2 つのプログラムから，それぞれソフトウェアバースマークを抽出し，ソフトウェアバースマークが類似している場合にソースコード再利用が存在すると判断する．ただし，ソフトウェアバースマークには，どの程度類似していればソースコードの再利用が行われたと判断するかを人に委ねるという問題点がある．

本研究では，これら 2 つの問題点を解決するために，実験的に求めた閾値に基づき，ソフトウェア再利用をコードクローンの量（コードクローンメトリクス）に基づいて検出する．コードクローンに基づくソースコード再利用の検出手法の 1 つである JPlag は，ソースコード間全体での類似度を算出するため，部分的なソースコード再利用を検出できなかっ

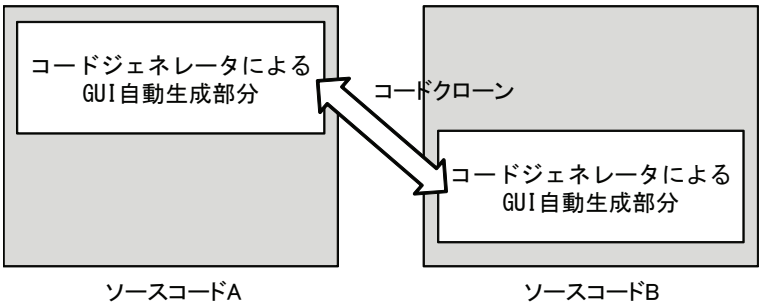

図 4.1: ソースコード再利用が原因ではないコードクローン

たと考えられる．そこで，本論文では，ソースコード間全体の類似度より，粒度の小さいコードクローンメトリクス（コードクローン長，ソースコードの一部分に着目した類似度）を用いることで，部分的なソースコード再利用を検出できるようにする．また，ソースコード再利用の有無の判断を人手に委ねているという問題を解決するために，各コードクローンメトリクスについて，ソースコードの再利用が行われた，もしくは行われていないと判断するための閾値を実験的に導出する．

以降，4.2 節では，コードクローンについて説明した後，ソースコード再利用とコードクローンとの関係について説明し，コードクローンメトリクスについて説明する．4.3 節では，本論文で新たに提案するコードクローンメトリクスについて説明した後，本章で提案するソースコード再利用の検出手法の詳細について述べる．4.4 節にて，提案手法で用いる閾値の導出実験を行った結果と考察を述べた後，導出した閾値がどの程度ソースコード再利用が行われたソフトウェアの組の集合，行われていないソフトウェアの組の集合をカバーしているかを述べる．4.5 節で，関連研究について説明する．最後に，4.6 節でまとめについて述べる．

4.2 コードクローンとソースコード再利用の関係

コードクローンとは，ソースコード中の類似または一致したコード列のことである．コードクローンは，コード列のコピーアンドペースト，コードジェネレータによるコード生成，パフォーマンスを向上させるための意図的なコード記述の繰り返し，偶然の一致，定型処理によって生じる [67].

ソースコードの再利用は，コードクローンが生成される方法と同様にコピーアンドペーストなどにより行われるため，ソフトウェア間でコードクローンがある場合，ソースコードの再利用により生成された可能性がある．ただし，コードクローンの発生理由に偶然の一致や定型処理が存在するため，コードクローンは必ずしもソースコード再利用によって発生するとは限らない（図 4.1）．その為，コードクローンがソースコード再利用によって生じたコードクローンなのか，または，偶然の一致や定型処理などによって生じたコードクローンなのかを判断する必要がある．

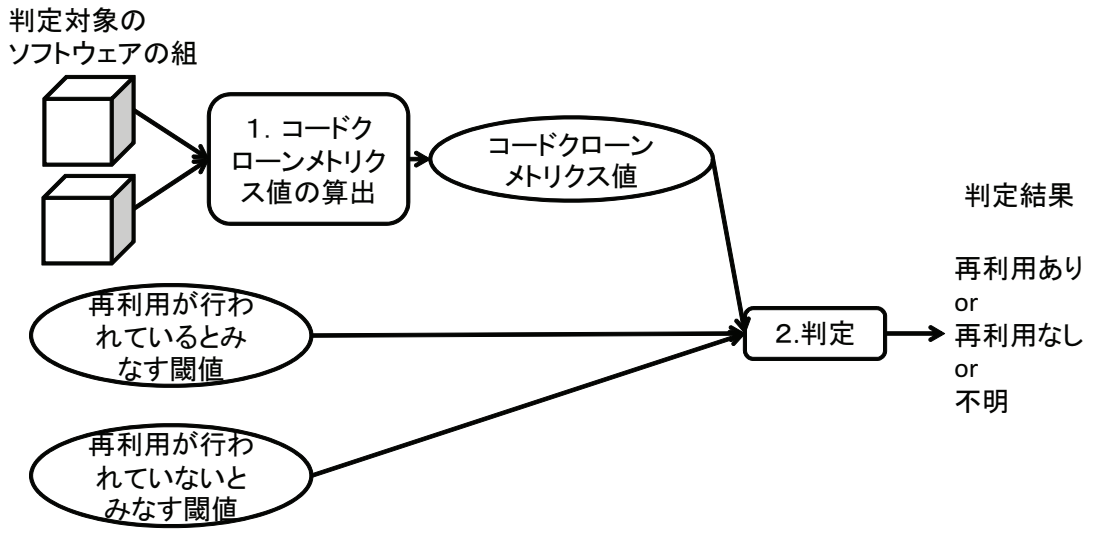

図 4.2: 提案手法が想定するソースコード再利用検出手法の概要

4.3 コードクローンメトリクスに基づくソースコード再利用検出手法のための閾値決定方法

本研究では，ソースコード再利用を検出する際に使用する，コードクローンメトリクスの閾値を，人手により作成されたソースコード再利用が行われているソフトウェアの組の集合と，ソースコード再利用が行われていないソフトウェアの組の集合を用いて，算出する手法を提案する．

提案手法で想定するソースコード再利用検出手法では，学習用のソフトウェア集合と検査対象のソフトウェアの組と再利用があるとみなす閾値，ないとみなす閾値を入力とし，検査対象のソフトウェア間で再利用が行われているかを判定する．ソースコード再利用検出手法の概要を図 4.2 に示す．各ソフトウェアの組について，コードクローンメトリクスの値を算出する．次に，２つの閾値とコードクローンメトリクスの値から，再利用がある，ない，不明のいずれかの判定を出力する．判定の概要を図 4.3 に示す．コードクローンメトリクスの値がソースコード再利用が行われているとみなす閾値を超えていれば，そのソフトウェア間ではソースコード再利用が行われていたと判定する．また，コードクローンメトリクスの値がソースコード再利用が行われていないとみなす閾値を下回る場合は，ソースコードの再利用は行われていないと判定する．どちらでもない場合は不明とみなす．

以降の節では，コードクローンメトリクスのうちソースコード再利用検出手法実現のために選択されたコードクローンメトリクスを紹介する．そして，各コードクローンメトリクスに対し，ソースコードの再利用が行われたと判断できる閾値，ソースコードの再利用が行われていないと判断できる閾値を算出する方法を説明する．

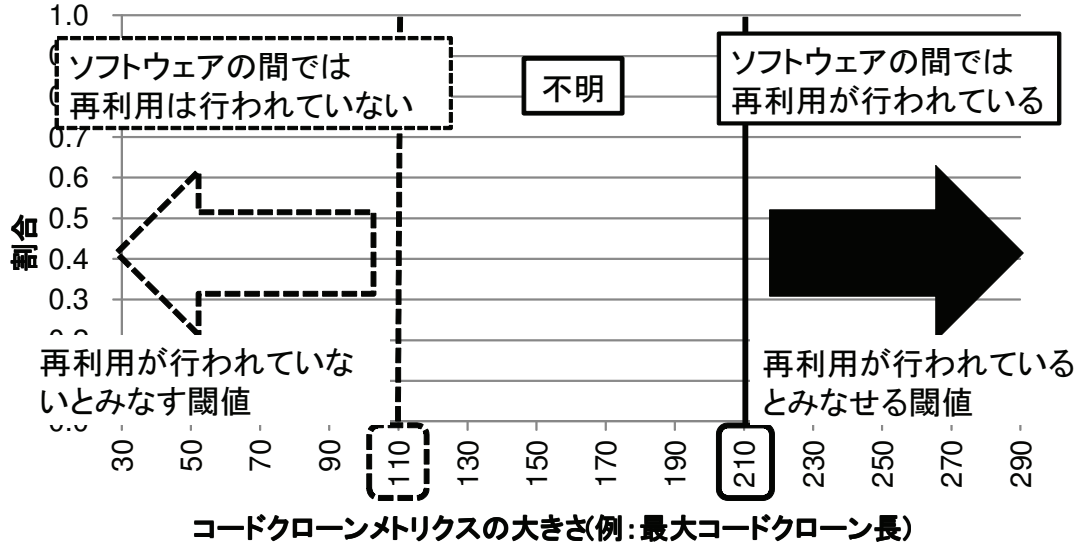

図 4.3: 判定の概要

4.3.1 使用するコードクローンメトリクス

本論文では，ソースコードの再利用検出に用いるコードクローンメトリクスとして，以下のコードクローンメトリクスを用いる．

コードクローン検出数 2 つのソフトウェア間で検出されるコードクローンの検出数

最大コードクローン長 2 つのソフトウェア間で検出されるコードクローンの中で最大のコードクローン長を持つコードクローンのトークン数

部分類似度 最大コードクローン長を含むファイルのコードクローン含有率．ソフトウェア A,B 間の部分類似度を $PSim(A,B)$，a,b をソフトウェア A,B 間で最大コードクローンを含むファイル，$|a|$, $|b|$ をファイル a,b の長さ，a , b 間の最大コードクローン長を $MLCC(a,b)$ とするとき，部分類似度は $PSim(A,B) = \frac{MLCC(a,b)}{|a|+|b|}$ として表せる．

4.3.2 閾値の決定方法

本手法は，ソフトウェア集合を入力とし，閾値を超えた場合にソースコードの再利用が行われたと判断する下限値と，閾値を下回った場合にソースコードの再利用が行われていないと判断する上限値を各メトリクスそれぞれについて出力する．提案手法の概要を図 4.4 に示す．

初めに，ソースコードの再利用が行われたソフトウェアの組と，ソースコードの再利用が行われていないソフトウェアの組をそれぞれ，ソースコード再利用が行われた場合の正解集合，ソースコード再利用が行われていない場合の正解集合として収集する．ソース

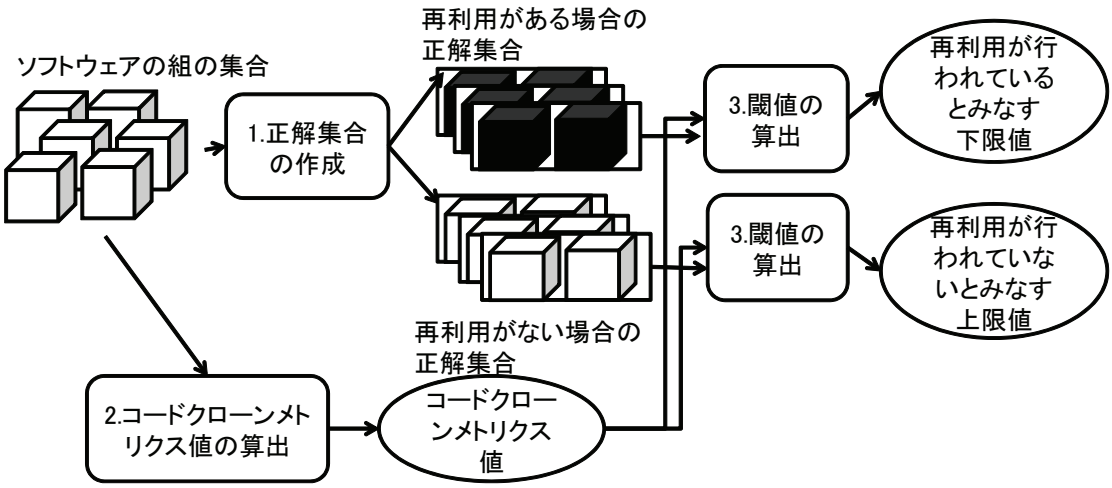

図 4.4: 閾値決定手法の概要

コードの再利用が行われているかいないかは機械的に求められない．そこで，まず，多数の OSS を集め，それぞれの組に対し，人手により再利用が行われているか判定する．その判定に基づき，各組をソースコード再利用が行われた場合の正解集合，ソースコード再利用が行われていない場合の正解集合に分類する．

次に，各ソフトウェアの全ての組み合わせに対して，前述のコードクローン検出数，最大コードクローン長，部分類似度を求める．

各コードクローンメトリクスが求まった後，それぞれの正解集合を用いて，ソースコード再利用が行われている場合の適合率と再現率，ソースコード再利用が行われていない場合の適合率と再現率と各コードクローンメトリクスにおける閾値との関係を求める．適合率は $(\text{適合率}) = \frac{(\text{正検出数})}{(\text{全検出数})}$，再現率は，$(\text{再現率}) = \frac{(\text{正検出数})}{(\text{正解集合の要素数})}$ で定義する．

最後に，ソースコード再利用が行われている場合とソースコード再利用が行われていない場合それぞれについて，適合率が 1 になる閾値の定義域を求め，ソースコード再利用が行われているとみなせる下限値と，ソースコード再利用が行われていないとみなせる上限値を求める．適合率が 1 を用いる理由は，再利用が行われているソフトウェアの組のうち，再利用があるとみなしたソフトウェアの組については，確実に再利用があるとする根拠とするためである．

4.4 実験

提案手法により，再利用を正しく検出することを確認することを目的とし，実験を行った．本実験では，初めに，各コードクローンメトリクスに対し，実験的に閾値を求めた．次に，コードクローンメトリクスを併用することで，正解集合をどの程度カバーできたかを調べた．最後に，ロジスティック回帰モデルを用いて，複数のコードクローンメトリクスをまとめて一つのメトリクスとして扱うことで，閾値を求めることができるか，どの程度正解集合をカバーできているか確かめた．

4.4.1 実験環境

解析対象として本実験では，Free Software Foundation が提供する Free Software Directory[6] で公開されている GPL，LGPL の OSS を 50 件使用した．これらのソフトウェアは主に GPL や LGPL で配布されており，また，Security, Audio, Game などといった様々なドメインから選択されている．これらのソフトウェアは C もしくは C++により開発されている．

また，コードクローン検出には，CCFinderX[3] を用いた．CCFinderX は，トークンに基づくコードクローン検出ツールである．CCFinderX ではコードクローンを検出するだけでなく，コードクローンの長さ (LEN) などのコードクローンメトリクスの算出も行える．

CCFinderX における最小コードクローン長は 30 に設定した．コードクローン長が 30 未満のコードクローンは，2 行，3 行程度のコード列であるため，ソースコード再利用ではないと判断した．そのため，本論文では，コードクローン長 30 未満のコードクローンは，ソースコード再利用なしとして扱う．

4.4.2 閾値の実験的算出

実験内容

4.3.2 節で提案した閾値決定手法を用いて，実験環境で紹介した 50 件の OSS から，ソースコード再利用が行われたとみなす下限値と，ソースコード再利用が行われていないとみなす上限値を求めた．

本実験では，コードクローンメトリクスとして，コードクローンの最大コードクローン長，部分類似度，ソフトウェア間のコードクローン検出数を用いた．

実験の際，作成した正解集合のサイズを表 4.1 に示す．

実験結果

ソースコード再利用が行われている場合の正解集合を用いた結果を説明する．コードクローン検出数を用いた場合の適合率，再現率を図 4.5 に，最大コードクローン長を用いた場合の適合率，再現率を図 4.6，部分類似度を用いた場合の適合率，再現率を図 4.7 に示す．それぞれ，X 軸がクローンメトリクスの値，Y 軸がそのクローンメトリクスの値を閾値とし，閾値以上のメトリクス値を持つソフトウェアの組をソースコード再利用が行われている組とみなした際の適合率，再現率の値である．それぞれの結果について，適合率が 1 に

表 4.1: 作成した正解集合のサイズ

ソースコード再利用の有無	組数
ソースコード再利用あり	121
ソースコード再利用なし	1104

なる場合の X 軸の定義域から，最大コードクローン長における下限値は 590，最大コードクローン長における下限値は 270，部分類似度における下限値は 0.3 と算出できる．

求めた下限値を閾値として設定した場合，再利用が行われている場合の正解集合のうち，下限値以上となる組がをどの程度あるか確認した結果を表 4.2 に示す．最大コードクローン長や部分類似度を用いた場合，それぞれの再利用が行われているソフトウェアの組み合わせのうち，それぞれ，91 組，72 組が閾値を超えていた．一方で，コードクローン検出数では，1 組だけが閾値を超えていた．

次に，ソースコード再利用が行われていない場合の正解集合を用いた結果を説明する．コードクローン検出数を用いた場合の適合率，再現率を図 4.8 に，最大コードクローン長を用いた場合の適合率，再現率を図 4.9 に，部分類似度を用いた場合の適合率，再現率を図 4.10 に示す．それぞれ，X 軸がクローンメトリクスの値，Y 軸がそのクローンメトリクスの値を閾値とし，閾値を下回るメトリクス値を持つソフトウェアの組をソースコード再利用が行われていない組とみなした際の適合率，再現率の値である．それぞれの結果について，適合率が 1 になる場合の X 軸の定義域から，最大コードクローン長における上限値は 50 と算出できる．一方，最大コードクローン長や部分類似度では，適合率が 1 になる場合の X 軸の値域が存在せず，上限値を算出することができなかった．

求めた上限値を閾値として設定した場合，再利用が行われていない場合の正解集合に含まれる組のうち，閾値を下回る組を調べた結果を表 4.3 に示す．最大コードクローン長では，再利用が行われていない組のうち，877 組が閾値以下となっている．

考察

再利用が行われているとみなす下限値を求めた際，最大コードクローン長や部分類似度と比べ，コードクローン検出数では正解集合のうち，下限値を超える組が大幅に少なかった．これは，図 4.6，図 4.7 より，最大コードクローン長や部分類似度では適合率が単調に増加していることから，再利用がある組は大きなコードクローンメトリクスを持つ傾向にあると考えられる．一方，コードクローン検出数では，適合率が増減を繰り返していることから，再利用がある組が大きなコードクローン検出数を持つ傾向はないと考えられる．以上の理由により，結果の差が生じたのではないかと考えられる．

表 4.2: ソースコード再利用ありの場合の下限値を用いた検出結果

	コードクローン検出数	最大コードクローン長	部分類似度
閾値	590	270	0.30
閾値以上の組数	1	91	72
閾値より小さい組数	120	30	49

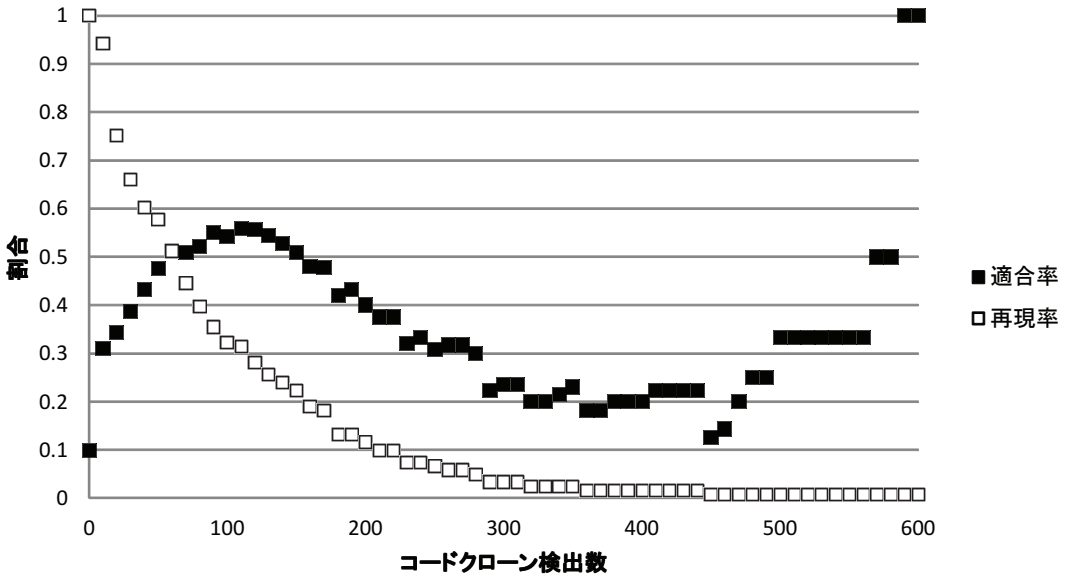

図 4.5: ソースコード再利用ありの場合でのコードクローン検出数と適合率，再現率の関係

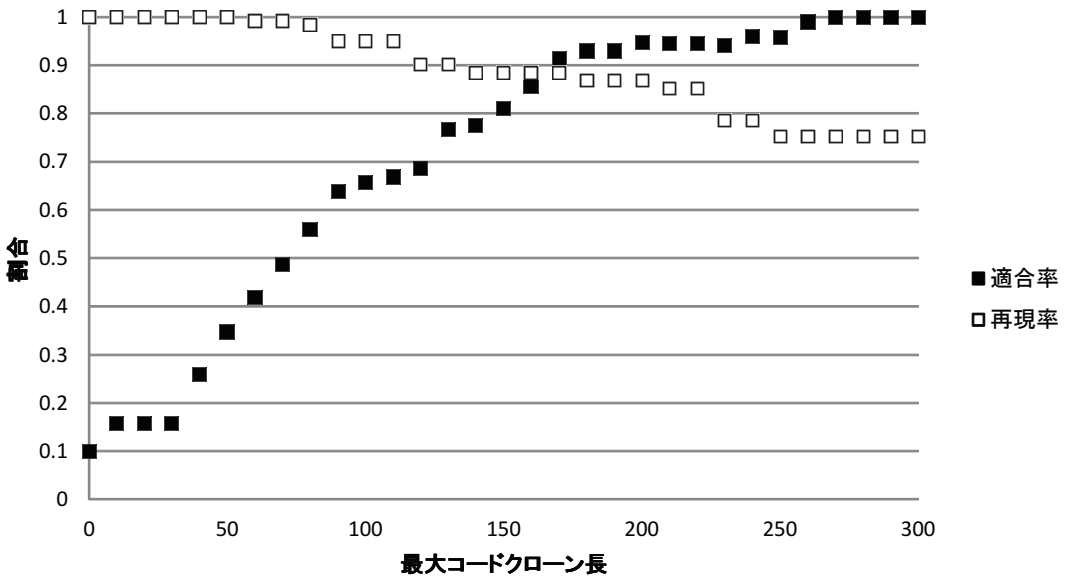

図 4.6: ソースコード再利用ありの場合での最大コードクローン長と適合率，再現率の関係

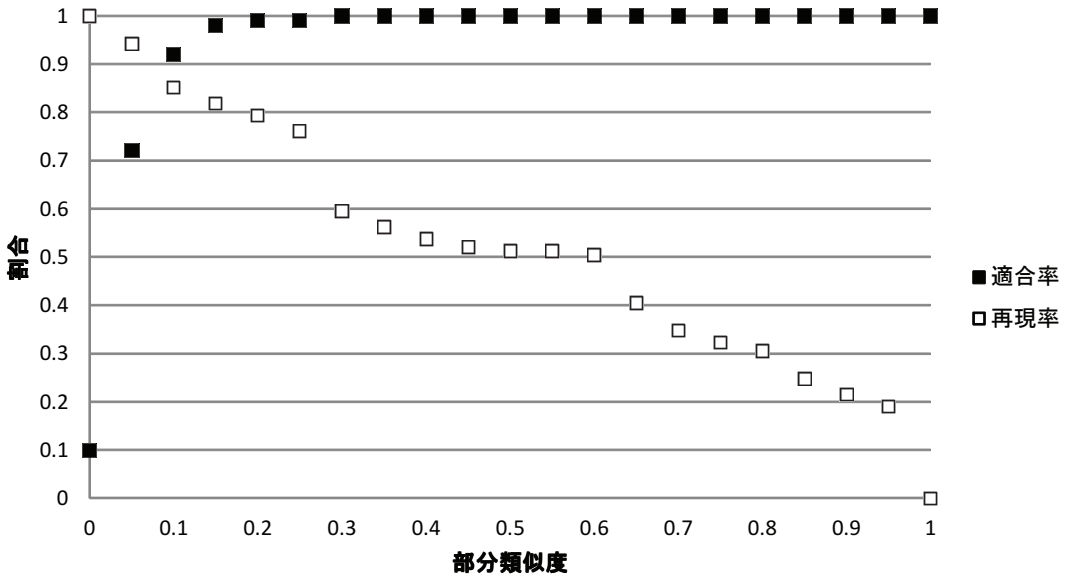

図 4.7: ソースコード再利用ありの場合での部分類似度と適合率，再現率の関係

次に，再利用が行われていないとみなす上限値を求めた際，コードクローン検出数や部分類似度では上限値を求めることができなかった．本章で想定している再利用検出手法では，再利用がないとみなす閾値以下では再利用されている組は存在していないとしている．しかし，これらのコードクローンメトリクスでは，今回正解集合作成に用いた 1225 組のうち，最も最少の値を持つ組で再利用が行われていた．そのため，上限値を求めることができなかった．

4.4.3 コードクローンメトリクスを併用したソースコード再利用検出実験

最大コードクローン長と部分類似度では検出できるソースコード再利用が異なる場合がある．本実験では，検出結果の改善を目標として，最大コードクローン長と部分類似度を併用してソースコード再利用を検出する実験を行った．実験では，ソースコード再利用が行われている場合について，最大コードクローン長，部分類似度を用いて順に判定を加えたとき，再利用ありとみなせる下限値以上となる組が再利用ありの場合の正解集合のうち

表 4.3: ソースコード再利用なしの場合での上限値を用いた場合の検出結果

	コードクローン検出数	最大コードクローン長	部分類似度
閾値	-	50	-
閾値以下の組数	-	877	-
閾値より大きい組数	-	227	-

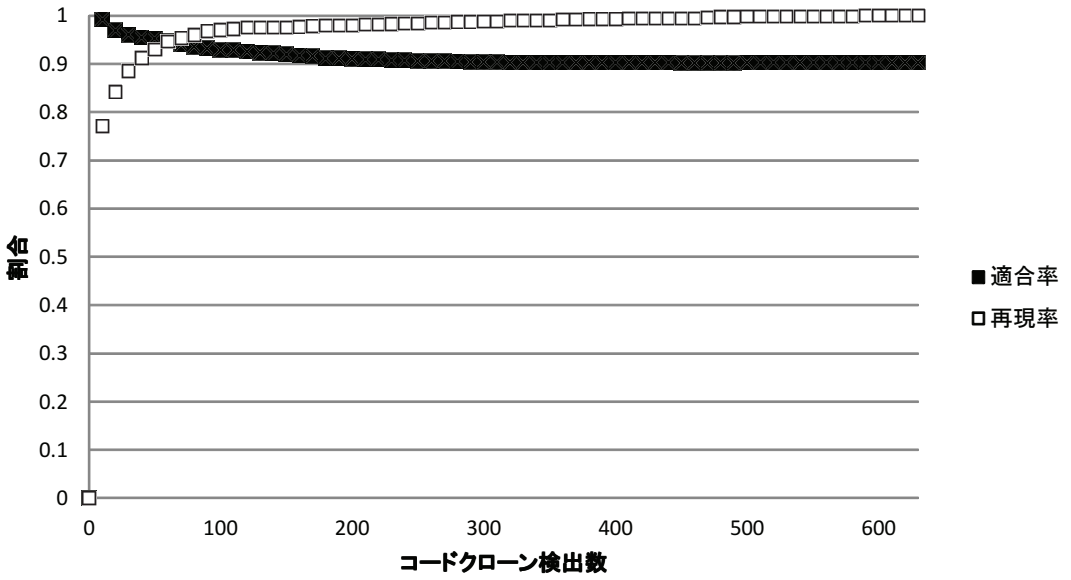

図 4.8: ソースコード再利用なしの場合でのコードクローン検出数と適合率，再現率の関係

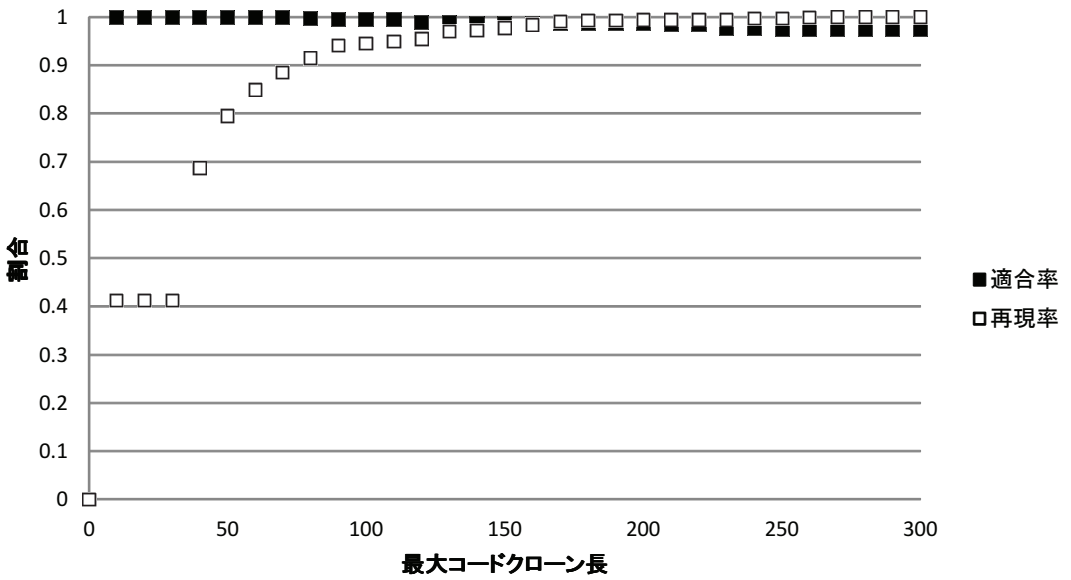

図 4.9: ソースコード再利用なしの場合での最大コードクローン長と適合率，再現率の関係

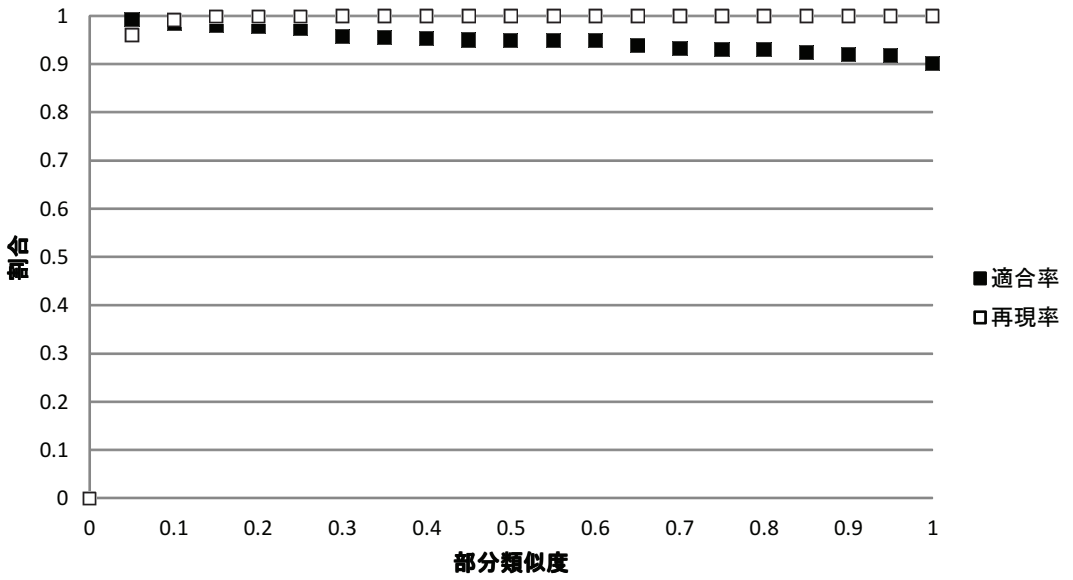

図 4.10: ソースコード再利用なしの場合での部分類似度と適合率，再現率の関係

どの程度あるか調査した．

実験で用いるソースコード再利用が行われている場合の下限値は 4.4.2 節の結果を用いる．

なお，再利用が行われていない場合については，閾値が算出できたメトリクスが一つだけであったため，本実験の対象外とする．

実験結果

ソースコード再利用が行われている場合の正解集合 121 組のうち 102 組がどちらかのメトリクスにおいて，再利用が行われているとみなす下限値以上であった．最大コードクローン長と部分類似度を併用した場合，下限値以上となったソースコード再利用の例を図 4.11 に示す．図 4.11 は，ソースコード再利用を含む片方のファイルの長さが 223 と小さく，最大コードクローン長における再利用とみなす下限値より小さい最大コードクローン長であった．そこで，部分類似度を用いた場合，部分類似度の値が 0.32 となり，部分類似度における再利用が行われているとみなす下限値を超えていた．これにより，最大コードクローン長と部分類似度がカバーできるソースコード再利用の範囲が異なり，特に，部分類似度では部分的な再利用をカバーできているといえる．

しかし，その一方で，図 4.12 のような，検出できなかったソースコード再利用が存在する．図 4.12 は，最大コードクローン長が 78，部分類似度 0.085 となるソースコード再利用である．これは，ファイルの長さが 636，1191 と非常に大きいため，相対的に最大コードクローン長が小さくなることで部分類似度の値も小さくなってしまったため，どちらのメトリクス値も下限値以下となっている．

```
/* Splice in "list" into "head" */
static inline void glame_list_splice(struct glame_list_head *list,
struct glame_list_head *head)
{
  struct glame_list_head *first = list->next;

  if (first != list) {
    struct glame_list_head *last = list->prev;
    struct glame_list_head *at = head->next;

    first->prev = head;
    head->next = first;

    last->next = at;
    at->prev = last;
  }
}
```

```
/*
 * Splice in "list" into "head"
 */
static INLINE void list_splice(struct list_head *list,
struct list_head *head)
{
  struct list_head *first = list->next;

  if (first != list) {
    struct list_head *last = list->prev;
    struct list_head *at = head->next;

    first->prev = head;
    head->next = first;

    last->next = at;
    at->prev = last;
  }
}
```

図 4.11: コードクローンメトリクスを併用した際，ソースコード再利用があると判断できた例

```
if (size == 0)
  return NULL;  /* no allocation required */

/* Allocate combined header + user data storage. */
{
  register pointer new = malloc (sizeof (header) + size);
  /* address of header */

  ((header *)new)->h.next = last_alloca_header;
  ((header *)new)->h.deep = depth;

  last_alloca_header = (header *)new;

  /* User storage begins just after header. */

  return (pointer)((char *)new + sizeof(header));
}
```

```
if (size == 0)
  return NULL;  /* No allocation required.  */

/* Allocate combined header + user data storage.  */
{
  register pointer new = malloc (sizeof (header) + size);
  /* Address of header.  */

  ((header *) new)->h.next = last_alloca_header;
  ((header *) new)->h.deep = depth;

  last_alloca_header = (header *) new;

  /* User storage begins just after header.  */

  return (pointer) ((char *) new + sizeof (header));
}
```

図 4.12: コードクローンメトリクスを併用した際，ソースコード再利用があると判断できなかった例

4.4.4 ロジスティック回帰モデルの判別値を閾値に用いた実験

4.4.2 節，4.4.3 節では，選択したコードクローンメトリクスの閾値をそれぞれ導出し，再利用がある，もしくはないと判定した組については誤りなくソースコード再利用検出を行うことを目的とした．しかし，これまでの例では，コードクローンメトリクスを 1 種のみ使用しているため，様々なソースコード再利用に対応できているわけではない．

そこで，ソースコードの再利用が行われている場合，行われていない場合について回帰モデルを作成し，それぞれのモデルの判別値から，閾値を算出することで，複数のコードクローンメトリクスをまとめて使用する．本実験では，モデルとして，ロジスティック回帰モデルを使用した．ロジスティック回帰モデルにおいて，事象が起こる確率 P は説明変数を X，偏回帰係数を b とすると，式 4.1 として表現される．本実験では，説明変数として，最大コードクローン長と部分類似度を用いた．

$$P = \frac{1}{1 + \exp\{-(b_0 + b_1 X_1 + b_2 X_2 + \ldots + b_n X_n)\}} \tag{4.1}$$

実験は以下の手順で行った．初めに，Leave-one-out 交差検証法を用いて，各ソフトウェアの組について判別値を求める．Leave-one-out 交差検証法は，与えられた入力データから 1 つのデータを抜き出してテストデータとし，残りのデータをモデルを作成するためのデータ（フィットデータ）として用いる．フィットデータを利用してモデルを作成し，作成したモデルの予測精度をテストデータを対象にして求める．これを，全てのデータが 1 回ずつテストデータになるように試行を繰り返し，精度の平均を取ることでモデルの精度として用いる手法である．例として，10 個のデータを入力とした Leave-one-out 交差検証法の概要を図 4.13 に示す．

実験では，各ソフトウェアの組み合わせについて，残りの組み合わせからロジスティック回帰モデルに基づいて，ソースコードの再利用がある場合の回帰式とソースコードの再利用がない場合の回帰式を求める．それらの回帰式を用いて，ソフトウェアの組の判別値を求める．

ソースコードの再利用が行われている場合の正解集合と判別値，ソースコードの再利用が行われていない場合の正解集合と判別値から，ソースコードの再利用が行われている場合の適合率と再現率，ソースコードの再利用が行われていない場合の適合率と再現率を求める．

適合率が 1 になる定義域をソースコード再利用が行われている場合，ソースコード再利用が行われていない場合について求め，ソースコード再利用が行われているとみなせる下限値，ソースコード再利用が行われているとみなせる上限値を求める．

また，求めた閾値が正解集合のうち，どの程度をカバーしているかも調べた．

実験結果

再利用が行われている場合の適合率，再現率を図 4.14 に示す．適合率が 1 になる値域から，再利用が行われているとみなせる下限値は 1.0 と算出される．また，再利用が行われ

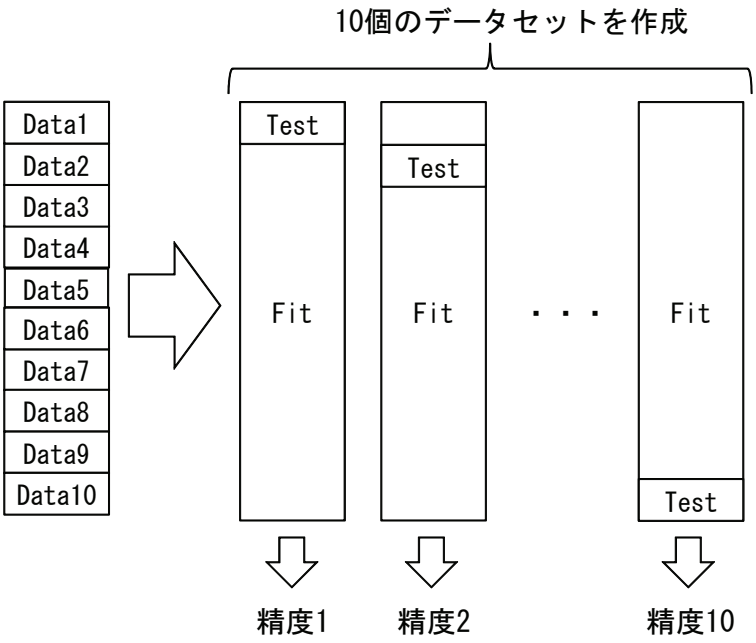

図 4.13: Leave-one-out 交差検証法の概要

ていない場合の適合率，再現率を図 4.15 に示す．グラフから，再利用が行われていないとみなせる上限値は 0.02 と算出できる．

得られた閾値を使用する際，ソースコード再利用が行われている場合の正解集合と，ソースコード再利用が行われていない場合の正解集合をどれだけカバーしているかを表 4.4 に示す．ロジスティック回帰モデルの判別値に基づくソースコード再利用があるとみなせる下限値 1.0 を用いることで，ソースコード再利用が行われている場合の正解集合のうち，約 54%をカバーしていた．再現率が低いが，これは，判別値が 0.995 のときソースコードの再利用が行われている組が存在しないにも関わらずソースコード再利用が行われていると判別したモデルが存在したことが原因だった．この正解集合を作成する際に生じた誤りを除いたところ，ソースコード再利用が行われている場合のロジスティック回帰モデルの判別値の下限値は 0.66 となり，ソースコード再利用が行われているソフトウェアの組のうち，約 88%をカバーできることを確認できた．

次に，ソースコード再利用がないとみなせる上限値 0.02 を用いることで，ソースコード再利用が行われていないソフトウェアの組の約 93%をカバーでき，適合率 1.0 のときに再現率も高い値を示していることが確認できた．適合率の推移はなだらかであることから，ロジスティック回帰モデルに基づくソースコード再利用のない場合の上限値は，安定してソースコード再利用がないことを判別できることを意味しているといえる．

本実験結果から，ソースコード再利用の有無を判断する際，ロジスティック回帰モデルの判別値を用いたソースコード再利用があるとみなせる下限値，ソースコード再利用がないとみなせる上限値を用いて，ソースコードの再利用がある組，ソースコードの再利用がない組を検出することで，複数のコードクローンメトリクスに基づき，より総合的にソー

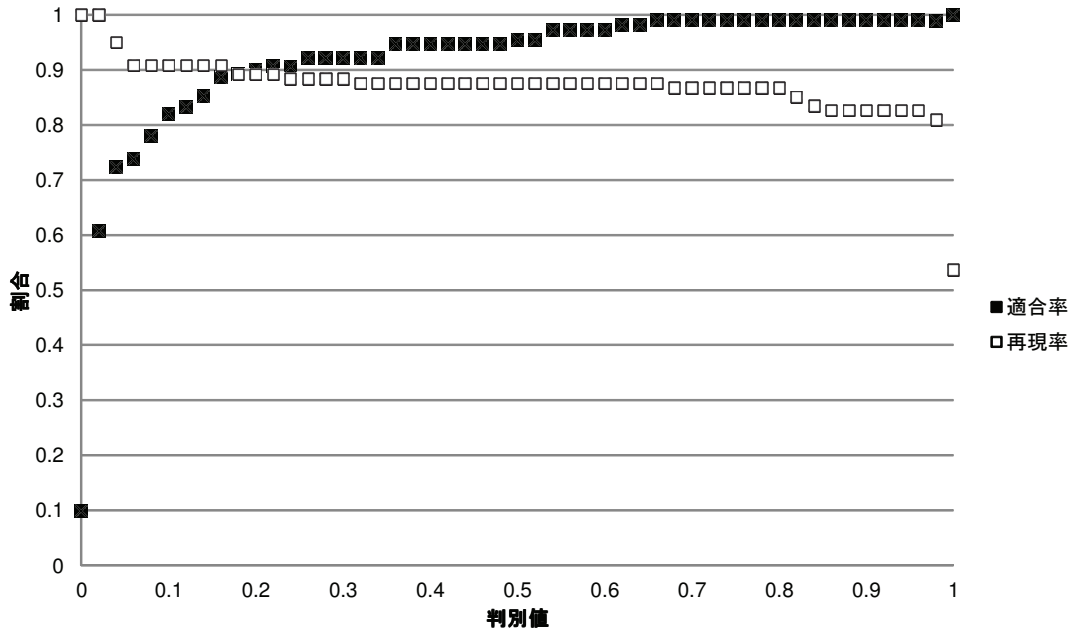

図 4.14: ソースコード再利用ありの場合のロジスティック回帰モデルの判別値と適合率，再現率の関係

スコード再利用の有無を判別できるといえる．

4.5 関連研究

4.5.1 ソフトウェア間におけるコードクローン含有率に基づく手法

JPlag は，Prechelt らによって開発された与えられたソースコード集合から類似したソースコード対を検出する手法である [48]．JPlag はソースコードを構文解析し，トークン列に変換する．トークン列をペアで比較し，ペアごとに類似度を求めることによりソースコー

表 4.4: ロジスティック回帰モデルの判別値を閾値に用いた場合の各正解集合でカバーした組数

	ソースコード再利用がない場合	ソースコード再利用がある場合
閾値	0.02	1.0
カバーしている組数	1026	65
カバーしていない組数	78	56

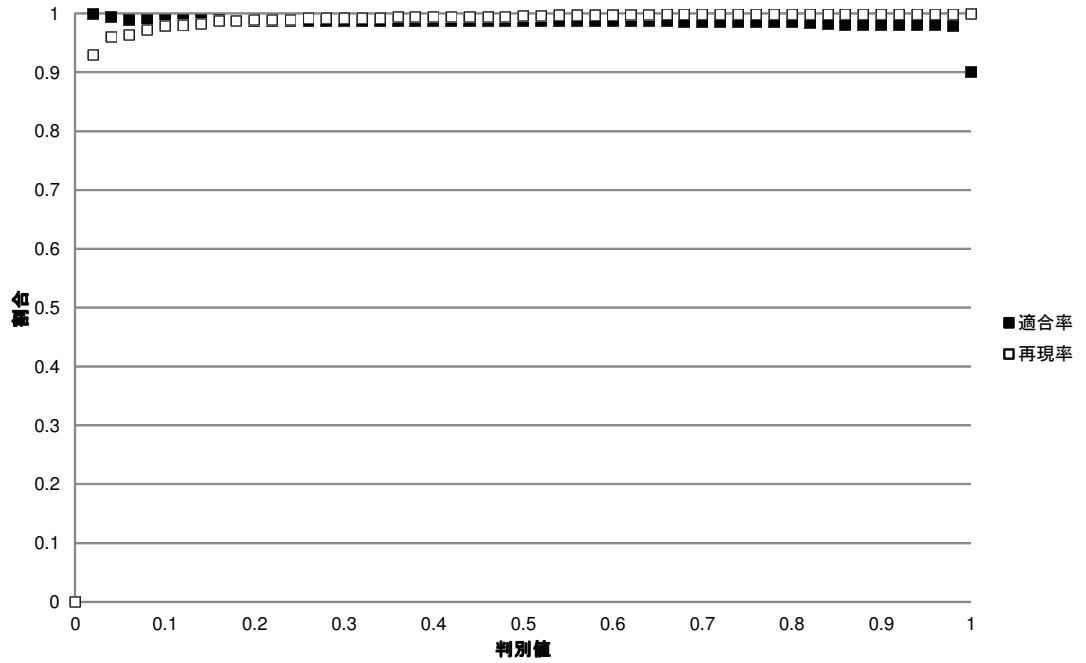

図 4.15: ソースコード再利用なしの場合のロジスティック回帰モデルの判別値と適合率，再現率の関係

ド再利用の検出を行う．類似度は次式（4.2），(4.3）で算出される．

$$Sim(A,B) = \frac{2 \cdot coverage(tiles)}{|A| + |B|} \tag{4.2}$$

$$coverage(tiles) = \sum_{match(a,b,length) \in tiles} length \tag{4.3}$$

類似度比較アルゴリズムには，Wise の “Greedy String Tiling” を用いている [60].
“Greedy String Tiling” は，1 組のテキストをトークン列化し，片方のトークン列中の部分列で，他方をどの程度カバー出来るかにより類似度を導出する手法である．なお，JPlag はプログラミング言語の Java，C，C++，Scheme に対応している．JPlag は，ソースコード全体が類似するソースコード再利用検出に役立つ．しかし，ソースコードの一部分だけを再利用した，部分的なソースコード再利用は検出できないことがあると考えられる．また，ソースコード再利用ありと判断する閾値について議論されていない為，ソースコード再利用の判断が人に委ねられてしまうという問題がある．本章では，部分的なソースコード再利用にも対応し，ソースコード再利用の有無を判断するための閾値を設定した点が異なる．

4.5.2 ソフトウェアバースマーク

ソフトウェアバースマークとは，対象となるソフトウェアが持つプログラムの特徴量を抽象化して表現したものである．2 つのプログラムから，それぞれソフトウェアバースマークを抽出し，ソフトウェアバースマークが類似している場合にソースコード再利用が存在すると判断する．ソフトウェアバースマークは，静的バースマークと動的バースマークの 2 つに分類される．静的バースマークとは，プログラムを実行せずに取得可能な特徴を使用したソフトウェアバースマークである．Tamada らは，初期値代入，メソッドの呼び出し系列，そして継承関係をソフトウェアバースマークとして提案している [54]．動的バースマークとは，プログラムの実行時に得られる特徴を使用したソフトウェアバースマークである．岡本らは，WindowsAPI の実行順序，実行頻度を動的バースマークとして提案している [64]．林らは，Java バイナリ中の実行系列中の処理間隔を動的バースマークとして提案している [65]．処理間隔を動的バースマークとして用いることで，攻撃態勢を向上させた．楓らは，部分的なソースコード再利用を検出するために，ソースコード再利用部分で定義されているメソッド毎に実行系列を抽出し，メソッドの部分系列毎に抽象化を行うことで部分的なソースコード再利用への有効性を示した [63]．ソフトウェアバースマークの性質上，多数のソフトウェアバースマークを複合的に検証する必要がある．そのため，ソースコード再利用ありと判断する際に，ソースコード再利用の判断が人手に委ねられてしまうという問題がある．本論文では，ソースコード再利用の判断を人手に委ねない為に，ソースコード再利用の有無を判断するための閾値を設定した点が異なる．

4.6 おわりに

本章では，2 つのソフトウェア間に存在するソースコード再利用を検出することを目的として，コードクローンメトリクスに基づくソースコード再利用の検出手法を提案した．ソースコード再利用検出に有効なコードクローンメトリクスとして,「コードクローン検出数」,「最大コードクローン長」,「部分類似度」を用いた．また，それぞれのコードクローンメトリクスに対し，ソースコード再利用が行われているとみなす下限値，ソースコード再利用が行われていないとみなす上限値を算出した．

第5章　むすび

本章では，本稿の結びとして，まとめと今後の研究方針について述べる．

5.1　まとめ

本稿では，ソフトウェアライセンス違反検出を目的とし，階層化知識を用いたソフトウェアライセンス特定手法，オープンソースソフトウェアにおけるソフトウェアライセンス分布の調査，コードクローンメトリクスに基づくソースコード再利用判定閾値の決定手法について述べた．

階層化知識を用いたソフトウェアライセンス特定手法では，大規模オープンソースソフトウェアを対象とし，ライセンス特定の課題を挙げ，課題を解決するために必要なライセンス特定ツールの要件を述べた．そして，ライセンス特定ツールの要件に基づくライセンス特定ツールの設計について述べ，実装を行った．評価実験では，既存のライセンス特定ツールと比べ，高い正答率を示した．この成果により，従来と比べ，ライセンス違反検出において重要であるライセンス特定を高速かつ高精度にできるようになったといえる．

オープンソースソフトウェアにおけるソフトウェアライセンス分布の調査では，大規模FOSSにおけるライセンスの分布について調査した．この調査により，各FOSSにおいて，ライセンスが混在していることを示した．次に，対象とするオペレーティングシステムにおけるライセンスの分布における変化は定期的，かつ連続的に発生することが分かった．また，我々はオペレーティングシステムのライセンス分布における変化の傾向はEclipseやArgoUMLのような非オペレーティングシステムの傾向と異なることも示した．さらに，ライセンスの分布に変化が起きる際，ソースコードファイルのライセンスが厳しいライセンスに変化する場合と，緩いライセンスに変化する場合の両方があることを示した．この成果により，ライセンスが時系列に沿って変化することを示し，ライセンス特定の重要性を高めた．

コードクローンメトリクスに基づくソースコード再利用判定閾値の決定手法では，2つのソフトウェア間に存在するソースコード再利用を検出することを目的として，コードクローンメトリクスに基づくソースコード再利用の検出のための閾値を決定する手法を提案した．ソースコード再利用検出に有効なコードクローンメトリクスとして,「コードクローン検出数」,「最大コードクローン長」,「部分類似度」を用いた．また，各コードクローンメトリクスについて，ソースコード再利用があるとみなすための閾値，ないとみなすため閾値をそれぞれ求めた．この成果により，ソースコード再利用検出において，根拠のある閾値を算出できるようになった．

5.2 今後の研究方針

階層化知識を用いたソフトウェアライセンス特定手法では，ライセンス特定に失敗した場合に，その結果を使用して知識を増やす仕組みがない．そのため，ライセンス特定に失敗したソースコードから，容易にライセンス知識を増やせるような仕組みを作る必要がある．

オープンソースソフトウェアにおけるライセンス分布の調査では，各リリースバージョンにおけるライセンスの分布を調査した．しかし，ソフトウェアの構造，つまり，ソースファイルが配置されているディレクトリの構造や，ソースファイル間の利用関係を考慮していない．今後，これらの構造とライセンスがどのように関係があるか調査する方針である．

コードクローンメトリクスに基づくソースコード再利用判定閾値の決定手法では，ソフトウェアの組について，それぞれ再利用が行われたかそうでないかを分類し，その集合に基づき，設定した各メトリクスに対して再利用判定の基準となる閾値を決定した．しかし，今回の閾値決定では一人の判断に基づいており，一般化を行うのは難しい．一般化を行うため，どのような特徴が一致するソースコード間では再利用が行われているとみなせるのかをもっと多数の参加者による判定結果から再利用が行われているか判定する基準となるルールを作成することや，判別分析法などの統計的手法による判別も検討する必要があろう．

参考文献

[1] Black duck koders.com. http://www.koders.com/.

[2] Black duck protex. http://www.blackducksoftware.com/protex.

[3] CCFinderX. http://www.ccfinder.net/ccfinderx-j.html.

[4] Code and license database - black duck knowledgebase. http://www.blackducksoftware.com/knowledgebase.

[5] Eclipse. http://www.eclipse.org/.

[6] Free software directory. http://directory.fsf.org/.

[7] Free Software Foundation. http://www.fsf.org/.

[8] Google code. http://code.google.com.

[9] Google code search. http://www.google.com/codesearch.

[10] Palamida. http://www.palamida.com/.

[11] Sourceforge.net. http://sourceforge.net/.

[12] ビジネスユースにおけるオープンソースソフトウェアの法的リスクに関する調査. http://www.ipa.go.jp/about/jigyoseika/04fy-pro/open/2004-741d.pdf.

[13] Epson pulls linux software following gpl violations. slashdot. http://slashdot.org/article.pl?sid=02/09/11/2225212, 2002.

[14] Playstation 2 game ico violates the gpl. slashdot. http://news.slashdot.org/article.pl?sid=07/11/28/0328215, 2007.

[15] 第 1 回 訴訟が増えている!? OSS ライセンス違反. @IT. http://www.atmarkit.co.jp/flinux/rensai/osslc01/osslc01a.html, 2008.

[16] 「USB 版 Windows 7」作成ツールに GPL コード Microsoft が謝罪. ITmedia news. http://www.itmedia.co.jp/news/articles/0911/16/news026.html, 2009.

[17] Thomas A. Alspaugh, Hazeline U. Asuncion, and Walt Scacchi. Analyzing software licenses in open architecture software systems. In *Proc. FLOSS 2009*, pp. 54–57, Washington, DC, USA, 2009.

[18] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proc. International Conf. on Software Maintenance*, pp. 368–377, Washington, DC, USA, 1998.

[19] Christine L. Braun. 再利用. John J. Marciniak, editor, ソフトウェア工学大辞典, 第 1 巻, pp. 338–405. 朝倉書店, 1994.

[20] Steven Burrows, S. M. M. Tahaghoghi, and Justin Zobel. Efficient plagiarism detection for large code repositories. *Softw. Pract. Exper.*, Vol. 37, No. 2, pp. 151–175, 2007.

[21] Seokwoo Choi, Heewan Park, Hyun-il Lim, and Taisook Han. A static api birthmark for windows binary executables. *J. Syst. Softw.*, Vol. 82, pp. 862–873, May 2009.

[22] Jorge Colazo and Yulin Fang. Impact of license choice on open source software development activity. *Journal of the American Society for Information Science and Technology*, Vol. 60, No. 5, pp. 997–1011, 2009.

[23] Amit Deshpande and Dirk Riehle. The total growth of open source. In Barbara Russo, Ernesto Damiani, Scott Hissam, Bjorn Lundell, and Giancarlo Succi, editors, *Open Source Development, Communities and Quality*, Vol. 275 of *IFIP International Federation for Information Processing*, pp. 197–209. 2008.

[24] Massimiliano Di Penta, Daniel M. German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the evolution of software licensing. In *Proc. ICSE 2010*, Cape Town, South Africa, 2010.

[25] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In *Proceedings of the IEEE International Conference on Software Maintenance*, ICSM '99, pp. 109–, Washington, DC, USA, 1999. IEEE Computer Society.

[26] Eclipse Foundation. Eclipse Development Process. `http://www.eclipse.org/projects/dev_process/development_process.php`, 2010. Accessed June. 2010.

[27] Raimar Falke, Pierre Frenzel, and Rainer Koschke. Empirical evaluation of clone detection using syntax suffix trees. *Empirical Softw. Engg.*, Vol. 13, pp. 601–643, December 2008.

[28] Daniel M. German, Massimiliano Di Penta, and Julius Davies. Understanding and auditing the licensing of open source software distributions. In *Proc. ICPC 2010*, pp. 84–93, Braga, Portugal, 2010.

[29] Daniel M. German and Ahmed E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *Proc. ICSE 2009*, pp. 188–198, 2009.

[30] Daniel M. German, Yuki Manabe, and Katsuro Inoue. A sentence-matching method for automatic license identification of source code files. In *Proc. ASE 2010*, pp. 437–446, 2010.

[31] Robert Gobeille. The FOSSology project. In *Proc. MSR 2008*, pp. 47–50, New York, NY, USA, 2008.

[32] Yoshiki Higo and Shinji Kusumoto. Code clone detection on specialized pdgs with heuristics. In *CSMR*, pp. 75–84, 2011.

[33] James Howison, Megan Conklin, and Kevin Crowston. Flossmole: A collaborative repository for floss research data and analyses. *International Journal of Information Technology and Web Engineering*, Vol. 1, pp. 17–26, 07/2006 2006.

[34] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pp. 96–105, 2007.

[35] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Software Engineering*, Vol. 28, pp. 654–670, July 2002.

[36] Raghavan Komondoor and Susan Horwitz. Using slicing to identify duplication in source code. In *Proceedings of the 8th International Symposium on Static Analysis*, SAS '01, pp. 40–56, 2001.

[37] Jens Krinke. Identifying similar code with program dependence graphs. In *Proc. 8th Working Conf. on Reverse Engineering*, pp. 301–309, Washington, DC, USA, 2001.

[38] Mu-Woong Lee, Jong-Won Roh, Seung-won Hwang, and Sunghun Kim. Instant code clone search. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pp. 167–176, 2010.

[39] Jingyue Li, R. Conradi, C. Bunse, M. Torchiano, O. Slyngstad, and M. Morisio. Development with off-the-shelf components: 10 facts. *IEEE Software*, Vol. 26, No. 2, pp. 80–87, March-April 2009.

[40] Hyun-il Lim, Heewan Park, Seokwoo Choi, and Taisook Han. A method for detecting the theft of java programs through analysis of the control flow information. *Information and Software Technology*, Vol. 51, pp. 1338–1350, September 2009.

[41] Wayne C. Lim. *Managing software reuse: a comprehensive guide to strategically reengineering the organization for reusable components*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[42] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. Gplag: detection of software plagiarism by program dependence graph analysis. In *Proc. 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 872–881, 2006.

[43] Yuki Manabe, Yasuhiro Hayase, and Katuro Inoue. Evolutional analysis of licenses in foss. In *Proc. IWPSE-EVOL '10*, pp. 83–87, 2010.

[44] Jean Mayrand, Claude Leblanc, and Ettore Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the 1996 International Conference on Software Maintenance*, ICSM '96, pp. 244–, 1996.

[45] Ginger Myles and Christian Collberg. Detecting software theft via whole program path birthmarks. In Kan Zhang and Yuliang Zheng, editors, *Information Security*, Vol. 3225 of *Lecture Notes in Computer Science*, pp. 404–415. Springer Berlin / Heidelberg, 2004.

[46] Ginger Myles and Christian Collberg. K-gram based software birthmarks. In *Proc. 2005 ACM symposium on Applied computing*, pp. 314–318, 2005.

[47] Heewan Park, Hyun-il Lim, Seokwoo Choi, and Taisook Han. A static java birthmark based on operand stack behaviors. In *Proceedings of the 2008 International Conference on Information Security and Assurance (isa 2008)*, pp. 133–136, Washington, DC, USA, 2008. IEEE Computer Society.

[48] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with jplag. *Journal of Universal Computer Science*, Vol. 8, No. 11, pp. 1016–1038, 11 2002.

[49] C. Ruffin and C. Ebert. Using open source software in product development: a primer. *IEEE Software*, Vol. 21, No. 1, pp. 82–86, 2004.

[50] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international con ference on Management of data*, pp. 76–85, New York, NY, USA, 2003. ACM.

[51] Richard W. Selby. Enabling reuse-based software development of large-scale systems. *IEEE Trans. Software Engineering*, Vol. 31, No. 6, pp. 495–510, June 2005.

[52] Katherine J. Stewart, Anthony P. Ammeter, and Likoebe M. Maruping. Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects. *Info. Sys. Research*, Vol. 17, pp. 126–144, June 2006.

[53] Haruaki Tamada, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. Java birthmarks : Detecting the software theft. *IEICE Trans. Inf. & Syst.*, Vol. 88, No. 9, pp. 2148–2158, 2005.

[54] Haruaki Tamada, Masahide Nakamura, Akito Monden, and Ken-Ichi Matsumoto. Java Birthmarks —Detecting the Software Theft—. *IEICE - Trans. Inf. Syst.*, Vol. E88-D, pp. 2148–2158, September 2005.

[55] Timo Tuunanen, Jussi Koskinen, and Tommi Karkkainen. Asla: Reverse engineering approach for software license information retrieval. In *Proceedings of the Conference on Software Maintenance and Reengineering*, pp. 291–294, 2006.

[56] Timo Tuunanen, Jussi Koskinen, and Tommi Karkkainen. Retrieving open source software licenses. In Ernesto Damiani, Brian Fitzgerald, Walt Scacchi, Marco Scotto, and Giancarlo Succi, editors, *Open Source Systems*, Vol. 203 of *IFIP International Federation for Information Processing*, pp. 35–46. Springer Boston, 2006.

[57] Timo Tuunanen, Jussi Koskinen, and Tommi Karkkainen. Automated software license analysis. *Automated Software Engineering*, Vol. 16, pp. 455–490, 2009.

[58] Timo Tuunanen, Jussi Koskinen, and Tommi Kärkkäinen. Automated software license analysis. *Automated Software Engineering*, Vol. 16, No. 3-4, pp. 455–490, 2009.

[59] Peter M. Watt-Morse. データ権. John J. Marciniak, editor, ソフトウェア工学大辞典, 第1巻, pp. 927–950. 朝倉書店, 1994.

[60] Michael J. Wise. String Similarity via Greedy String Tiling and Running Karp Rabin Matching. Technical report, Basser Department of Computer Science University of Sydney, 2006.

[61] Wuu Yang. Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, Vol. 21, pp. 739–755, June 1991.

[62] 古谷栄夫, 松下正, 畫島宏明, 鶴本祥文. 知って得するソフトウェア特許・著作権改定五版知って得するソフトウェア特許・著作権. アスキー, 2008 3.

[63] 楓基晴, 真野芳久. Javaプログラムの部分盗用に対する動的バースマーク. 電子情報通信学会総合大会講演論文集, Vol. 2007, p. 219, 2007-03-07.

[64] 岡本圭司, 玉田春昭, 中村匡秀, 門田暁人, 松本健一. Api 呼出しを用いた動的バースマーク (ソフトウェア基礎, プログラム理論). 電子情報通信学会論文誌. D, 情報・システム, Vol. 89, No. 8, pp. 1751–1763, 2006-08-01.

[65] 林晃一郎, 楓基靖, 真野芳久. 特徴抽出と抽象化による動的バースマークの構成とその検証. 情報処理学会研究報告. CSEC, [コンピュータセキュリティ], Vol. 2005, No. 122, pp. 31–36, 2005-12-09.

[66] 玉田春昭, 神崎雄一郎, 中村匡秀, 門田暁人, 松本健一. Javaクラスファイルからプログラム指紋を抽出する方法の提案. 情報処理学会研究報告. CSEC, [コンピュータセキュリティ], Vol. 2003, No. 74, pp. 127–133, 2003-07-17.

[67] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌. D, 情報・システム, Vol. 91, No. 6, pp. 1465–1481, 2008.

[68] 可知豊. ソフトウェアライセンスの基礎知識. ソフトバンククリエイティブ, 9 2008.

[69] 真鍋雄貴, Daniel M. German, 井上克郎. ライセンス知識に基づくライセンス特定ツールの設計. ウィンターワークショップ 2010・イン・倉敷 論文集, 第 2010 巻, pp. 19–20, January 2010.

[70] 横森励士, 梅森文彰, 西秀雄, 山本哲男, 松下誠, 楠本真二, 井上克郎. Java ソフトウェア部品検索システム SPARS-J. 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理, Vol. 87, No. 12, pp. 1060–1068, 2004.