Vol.2015-SE-190 No.2
2015/12/15

コードクローンと使用ライブラリに着目した オープンソースソフトウェアの進化の定量化

若林洸太[†1]　門田暁人[†1]　伊原彰紀[†2]　玉田春昭[†3]

概要：本稿では，オープンソースソフトウェア（OSS）開発におけるソースコードの進化を定量化することを目的として，(1)コードクローン，(2)使用ライブラリの 2 つに基づいた方法を提案する．まず，(1)については，OSS のバージョン間の type 2 コードクローンを計測することで，ごく軽微な変更やコードの重複の影響を除外した開発規模の推移，および，変更規模の推移を定量化する．また，(2)については，プログラムの機能の進化，という新たな切り口での定量化を可能とする．一般に，ソフトウェアライブラリは，それぞれ固有の機能を提供していることから，使用ライブラリの変遷を定量化・可視化することで，機能的な側面からのソフトウェア進化の分析に役立つと期待される．

キーワード：開発規模，ソフトウェア機能，ソフトウェアライブラリ

Quantifying the Evolution of Open Source Software Focusing on Code Clone and Used Libraries

Kota WAKABAYASHI[†1]　Akito MONDEN[†1]
Akinori IHARA[†2]　Haruaki TAMADA[†3]

Abstract: This paper aims to quantify and visualize the evolution of source code in an open source software development based on (1) code clone and (2) used library classes. As for (1), this paper measures type-2 code clones between two succeeding versions of source code; and then, compute clone-based development size that can ignore both trivial changes (e.g. renaming variable and/or function names) and code cloning. As for (2), this paper presents evolution of functionality rather than program size or complexity. Generally, each software library class provides different and unique functionality; thus, we expect that visualizing the evolution of used library classes helps us to observe evolution of software functionality.

Keywords: Development size, software functionality, software library

1. はじめに

今日の多くのソフトウェア開発は，オープンソースソフトウェア（Open Source Software; OSS）に依存している．例えば，Web アプリケーションの開発では，LAMP（Linux, Apache, MySQL, PHP/Perl/Python）と呼ばれる OSS 群を用いることが定番となっている．また，（株）富士通研究所の野村氏は，「最近の開発は全て OSS からはじまる」と述べており，ソフトウェアの開発速度を上げるために OSS を数多く採用したい，という要求が企業側にあるという 11). 独立行政法人情報処理推進機構（IPA）によるオープンソースソフトウェア活用ビジネス実態調査 3)では，回答を寄せた 916 社の 51.6%が，顧客向けシステムの開発において OSS を利用しており，自社内での利用も含めればその割合は 66.8%に達する．

その一方で，商用ソフトウェア開発における OSS の採用には多くの課題があることも指摘されている（表 1）．これら課題には，OSS の機能や品質に関するものに加え，開発体制や意思決定プロセス等に起因するものも多く含まれる．

表 1. OSS 利用における主な課題（文献 3)より抜粋）

機能・品質等に関する課題	
種類が多すぎて，本当に利用価値のある OSS が分かりにくい.	32.8%
個別技術・ソフトウェアの評価・成熟度が把握しにくい.	29.5%
開発体制や意思決定プロセス等に関する課題	
緊急時の技術的サポートを得にくい.	67.3%
利用している OSS がいつまで存続するか分からない.	58.8%
バグの改修や顧客からの要請対応に手間がかかる.	43.4%

このことは，直接的な利用対象である OSS の現在のソースコードに加えて，OSS の開発プロセスや過去から現在に至るソフトウェア進化の過程についてよく知ることが OSS 利用において重要であることを意味している．

従来，OSS の成熟度や開発の存続性の評価に役立てることを目的として，OSS の進化の過程を定量化・可視化する

†1 岡山大学
Okayama University
†2 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
†3 京都産業大学
Kyoto Sangyo University

研究が行われてきた．2013 年に発表された OSS プロジェクトの進化に関する Systematic Review 14)によると，OSS 進化の研究は増加傾向にあり，ソースコードの進化を対象としたものが多いことが示されている．また，ソースコードの進化の過程を測るメトリクスとして，ソースコードメトリクス（行数，関数の数など），コードの複雑さメトリクス（Cyclomatic 数，Halstead の尺度など），オブジェクト指向メトリクス（C&K メトリクス，Lorenz and Kidd のメトリクスなど）が用いられているとの結果が示されている．

本稿では，ソースコードの進化を定量化する新たなアプローチとして，(1)コードクローン，(2)使用ライブラリの 2 つに着目する．まず，(1)については，ソースコードメトリクスの改良であり，OSS のバージョン間の Type 2 コードクローンを計測することで，ごく軽微な変更やコードの重複の影響を除外した開発規模の推移，および，変更規模の推移を定量化する．また，(2)については，プログラムの機能の進化，という新たな切り口での定量化を可能とする．一般に，ソフトウェアのクラスライブラリは，それぞれ固有の機能を提供いることから，使用クラスライブラリの変遷を定量化・可視化することで，機能的な側面からのソフトウェア進化の分析に役立つと期待される．

以降，2 章では，従来研究について．3 章では提案手法．4 章では，結果を記載していく．

2. 従来研究

2.1 開発量の計測

一般に，企業におけるソフトウェアの改良開発では，ソフトウェア開発量に関する尺度として，追加規模（新規規模），修正規模（変更規模），母体規模（流用規模），削除規模の 4 つを区別して計測する 4)．それぞれが開発コスト，テスト工数，品質等に与える影響が異なるためである．OSS も改良開発の一種であるから，これらを区別して計測することが望ましい．なお，企業におけるソフトウェア開発では，追加規模と修正規模の和を「開発量」とすることも多い．この場合，母体規模（流用規模）や削除規模は開発量とはみなされないこととなる．

従来のソフトウェア進化研究においても，追加，変更，修正を区別した計測が行われている．例えば，Canfora ら 2)は，バージョン管理システムからコミット毎の追加行数，削除行数，変更行数を正確に計測する方法を提案し，ArgoUML に適用した事例を示している（図 1）．

ただし，従来研究では，次の点が問題であった．

(1) 本質的でない変更を，変更としてカウントしてしまう．例えば，改行の挿入や削除，行の入れ替え，ソースファイル名，クラス名，メソッド名，関数名などの変更である．
(2) コピー&ペーストによるコードの水増しが行われた場合，見かけ上の開発量が増えてしまう．
(3) (2)とは逆に，コードの重複をなくすような改善（リファクタリング）が行われた場合，見かけ上の開発量が減ってしまう，あるいは，マイナスとなる．

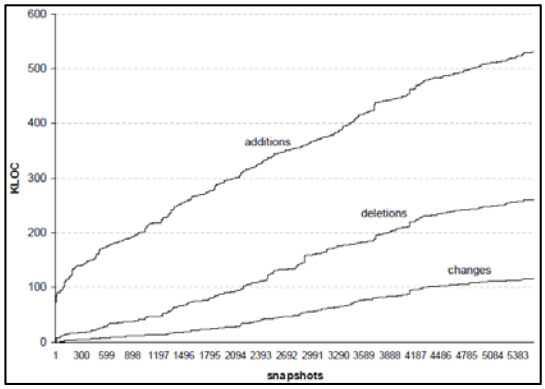


図 1　ArgoUML のおけるソースコード進化（文献 2)より）

本稿では，(1)～(3)の解決を目的として，ソフトウェアのバージョン間の Type2 コードクローン（後述）を計測することで，本質的でない変更やコードの重複を考慮した開発量の計測を可能とすることを目指す．

2.2 ソフトウェアの特徴量の計測

ソフトウェアの開発量の計測だけでは，開発によってソフトウェアの内容や性質がどのように変化したのかを知ることができない．そこで，従来，ソフトウェアの様々な特徴量の変化の計測が行われてきた．例えば，Halstead の尺度や Cyclomatic 数 12)，凝集度 13)，オブジェクト指向メトリクス 5)などである．ただし，これらの研究では，ソフトウェアの構造や複雑さの変化をとらえることはできるが，ソフトウェアの機能の変化についてはうかがい知ることはできない．また，従来，ソフトウェアに含まれる関数やグローバル変数の増減の時系列分析も行われている 10)が，関数の内容までは分からないため，どのような機能が追加・変更されたのかを知ることはできない．

本稿では，ソフトウェア中で使用されているクラスライブラリの変化に着目し，可視化・定量化を行う．クラスライブラリはそれぞれ固有の機能を提供していることから，使用ライブラリの系列変化を明らかにすることで，ソフトウェアの機能の変化をある程度推定できると期待される 6)16)．従来，ソフトウェアの分類を目的として，使用ライブラリの情報が用いられてきた．例えば，文献 16)では，多数の Java プログラムを使用ライブラリによって分類した結果，メールの送受信の機能を持っているプログラムは JavaMail ライブラリ（javax.mail）を使用している，といった関連を明らかにしている．また，文献 6)では，一つのソフトウェアを構成するクラス群を，それぞれの使用クラス

に基づいてクラスタリングを行い，各クラスタがどのような機能に関連するかを分析している．

これらの研究に対し，本稿では，ソフトウェア進化の分析に使用ライブラリの時系列変化を計測する点が異なる．使用ライブラリの変遷を可視化・定量化することで，機能の大きな変化をとらえることができると期待される．

3. 提案方法

3.1 コードクローンに基づく開発量の計測

本稿では，あるバージョンのソフトウェアを対象とし，その 1 つ前のバージョンからの差分となる開発量の計測を目的とする．開発量の計測にあたっては，コードクローン計測技術を利用する．コードクローンとはソースコード中に存在する同一，もしくは，類似した部分のことである．重複コードとも呼ばれる．コードクローン含有率が高いソフトウェアは，あるコードを修正したなら，その重複コード全てに修正を食わなければならない可能性があるので保守作業において問題になりやすい．

提案方法では，2 つのバージョン間のコードクローンを検出し，新しい方のバージョンのプログラムにおいて，コードクローンに含まれていないコード断片のトークン数をカウントし，開発量とみなす．図 2 に示すように，コードクローンは，類似するコード断片の組である「クローンペア」として検出されるため，まず，2 つのバージョン間の全てのクローンペアを検出する．図の例では，2 つのクローンペアが検出されている．次に，新しい方のバージョン（version n）において，いずれのクローンペアにも含まれない部分を，新たに開発されたコードであるとみなす．2.1 節で述べたように，一般に，開発量は追加規模と修正規模の和としてとらえられることがあるが，提案方法もこの考えに沿っている．

提案方法では，コピー&ペーストによるコードの水増しが行われた場合であっても，開発量が増えたとはみなされない．図 3 に例示するように，古いバージョンにおけるコード断片(B)がコピー&ペーストにより追加された場合，その部分はクローンペアに含まれるため，新たに開発されたコードとはみなさない．そのため，コピー&ペーストによって見かけ上の開発量が増えるという問題を回避できる．また逆に，コードの重複をなくすような改善（リファクタリング）が行われた場合であっても，クローンペアに含まれるコードは開発量とみなさないため，見かけ上の開発量が減るというような問題は起こらない．

開発量の計測にあたっては，改行の挿入や削除，ソースファイル名，クラス名，メソッド名，関数名などの変更，といった本質的でない変更を無視するため，本稿では，type 2 クローンを計測することで，開発量を導出する．一般に，コードクローンは，次に示すように type 1，type 2，type 3 に分類される 1).

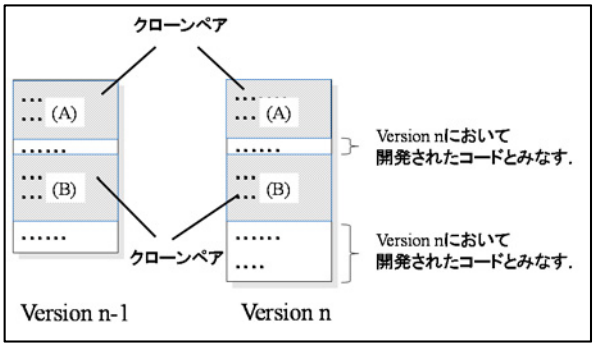


図 2　バージョン間のコードクローン計測に基づく開発量の導出

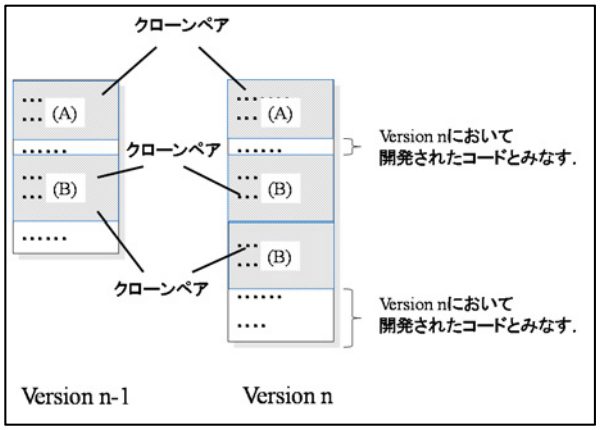


図 3　バージョン間のコードクローン計測に基づく開発量の導出

- Type 1 クローンペア : 完全に一致するコード断片のペアである．ただし，空白とコメントの不一致は許容する．
- Type 2 クローンペア : Type 1 において変数名や関数名の違いを許容したコード断片のペアである．
- Type 3 クローンペア : Type 2 において一部の文の違いを許容するコード断片のペアである．小規模な文の挿入，削除，変更がなされた場合にもクローンとして検出する．どの程度の違いを許容するかを，コード断片間の類似度により与える．

本稿では，開発量を計測するという観点から，type 2 クローンペアを計測することとした．

開発量を定量化するにあたっては，まず，計測対象のバージョンのソフトウェアにおいて，開発されたコード（つまり，旧バージョンからの差分）の占める割合を表す尺度として，式(1)の通り変化率を定義する．

$$\text{変化率} = 1 - \text{CVR}_{inter\_ver} \quad \cdots(1)$$

この変化率をもとに，開発量を定義する．ただし，その単位としてトークン数を用いるより，ソースコード行数を用いる方がより直観的である．そこで，次の(2)式により，行数を単位とする開発量を定義する．

$$\text{開発量}[\text{行数}] = (1 - \text{CVR}_{inter\_ver})\text{SLOC} \quad \cdots(2)$$

ここで，SLOC は，新バージョンのプログラムにおけるソースコード行数であり，CVR_{inter_ver} は，新バージョンのプログラムにおけるバージョン間クローンの含有率（プログラムを構成する全トークン数に対する，バージョン間クローンペアに含まれるトークン数の割合）である．

また，本稿では，任意のスナップショットにおけるソースコード規模の尺度として，コードクローンによる重複コードを除外した非重複 SLOC を式(3)の通り定義する．

$$\text{非重複 SLOC}[\text{行数}] = \frac{1 - CVR_{in\_ver}}{2}\text{SLOC} \quad \cdots(3)$$

ここで，CVR_{in_ver} は，新バージョンのプログラムにおけるバージョン内クローンの含有率（プログラムを構成する全トークン数に対する，バージョン内のクローンペアに含まれるトークン数の割合）である．

3.2 使用ライブラリの計測

本稿では，Java プログラムを対象とし，分析対象ソフトウェアの各バージョンから使用クラスライブラリを抽出する．Java のライブラリは多種多様であるが，本稿では，多くのソフトウェアで利用されているものとして，JDK，すなわち Oracle が提供する標準拡張ライブラリ（java, javax で始まるパッケージ名のもの），及び，Apache が提供するクラスライブラリ（org.apache で始まるパッケージ名のもの）を抽出する．使用クラスの抽出には，ソフトウェアバースマーク抽出ツール Stigmata15)を用いる．

本稿では，ソフトウェアの各バージョンで追加または削除された使用クラスを明らかにすることで，機能の追加／変更を推定する際の判断材料とする．

4. ケーススタディ

4.1 対象プロジェクトと使用ソフト

対象プロジェクトとして，オープンソースで開発されている Java 向けロギングソフト SLF4J を採用する．SLF4J のバージョンは 1.0.0~1.7.12 の間の進化を計測する．

ソースコード行数の計測には，多機能ステップカウンタかぞえチャオ 9)を用いる．また，コードクローンの計測には CCfinderX7)8)を用いる．

4.2 コードクローンに基づく開発量の計測

4.2.1 開発量と変化率の推移

提案方法に基づく開発量，及び， SLOC 差分の推移を図 4 に示す．ここで，SLOC 差分とは，１つ前のバージョンからの SLOC の増加量である．図 4 より，SLOC 差分と開発量には大きな違いがあることが見て取れる．SLOC 差分が極めて小さい，もしくは，マイナスの場合であっても，開発量が大きい場合がある．逆に，SLOC 差分は極めて大きいが，開発量は小さい場合もあった．このことから，SLOC の差分は，開発量の実体を表していないことがうかがえる．

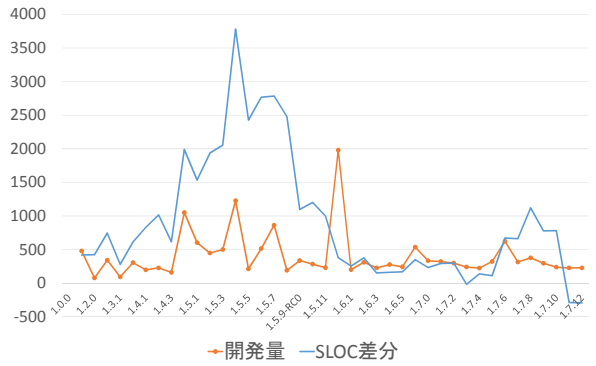


図 4　SLF4J の開発量と SLOC 差分の推移

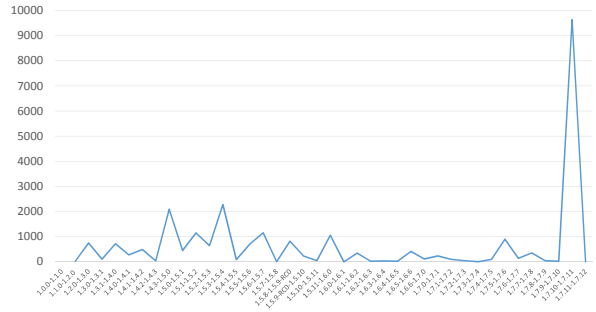


図 5　SLF4J の追加規模と修正規模の和の推移

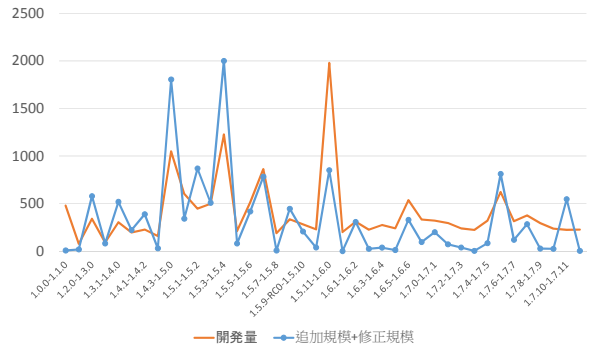


図 6　SLF4J の追加規模と修正規模の和（空行とコメントを無視したもの）の推移と，提案方法による開発量の推移

また，図 5 に従来の開発量の尺度である追加規模と修正規模の和（SLOC）の推移を示す．図 5 より，提案方法による開発量の尺度と全く異なる推移となっていることが分った．特に，あるバージョンでは 9000SLOC を超える値となっており，大規模な削除と修正が行われていた．ただし，その中身を見ると，空行やコメント行に対する削除や修正が大部分であった．そこで，追加規模と修正規模の計測に

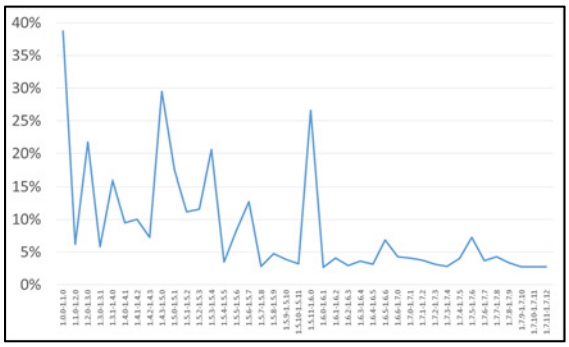


図 7　SLF4J の変化率の推移

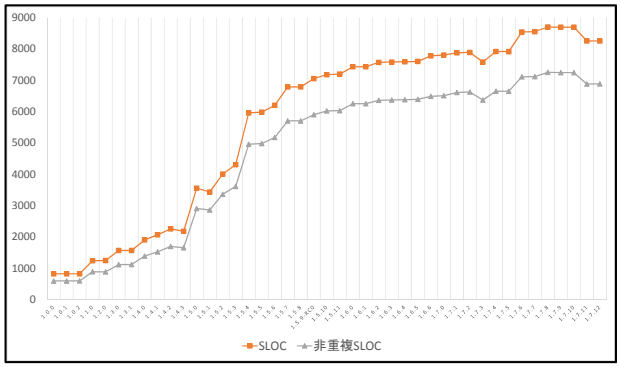


図 8　SLF4J の SLOC と非重複 SLOC の推移

あたって，空行やコメント行を無視した場合の結果を図 6 に示す．図 6 では，提案方法による開発量も同時に示す．図 6 においても，提案方法による開発量と，従来法による開発量（追加規模＋修正規模）は少なからず異なっていることが分かる．このことから，従来法による開発量の計測値は，開発の実体を必ずしも表していないことがうかがえる．

また，変化率を図 7 に示す．バージョン 1.4.x 周辺は変化率が大きく，旧バージョンからの差分が大きいことがわかる．逆に 1.6.3 以降はほぼ 5%程度に収まっており，大きな追加や変更が加えられていないことが分かる．

4.2.2 非重複 SLOC の推移

非重複 SLOC の推移を図 8 に示す．SLOC と非重複 SLOC の行数を比較すると，10～15%ほど非重複 SLOC の方が小さくなっている．なお，SLOC の計測にあたっては，空行とコメント行を含めていない．

4.3 使用ライブラリクラスの推移

SLF4J に使用されているライブラリクラスは表 2 の通りであった．使用ライブラリクラス数の推移を図 7 に示す．バージョンアップとともに機能が増えているため使用クラスライブラリ数も増加している．大きくクラス数が増えているのは，1.4.3-1.50，1.5.6-1.5.7 の部分である．

表 2　SLF4J における使用ライブラリクラス

種別		
java	io	Closeable
		PrintStream
		Serializable
	lang	Boolean
		Class
		ClassLoader
		InheritableThreadLocal
		Object
		SecurityManager
		StackTraceElement
		String
		StringBuffer
		StringBuilder
		System
	net	URL
	util	ArrayList
		Arrays
		Collection
		Collections
		concurrent.ConcurrentHashMap
		concurrent.ConcurrentMap
		Enumeration
		HashMap
		Iterator
		LinkedHashSet
		List
		logging.Level
		logging.Logger
		logging.LogRecord
		Map
		Set
		Vector
org	apache	commons.logging.Log
		commons.logging.LogFactory

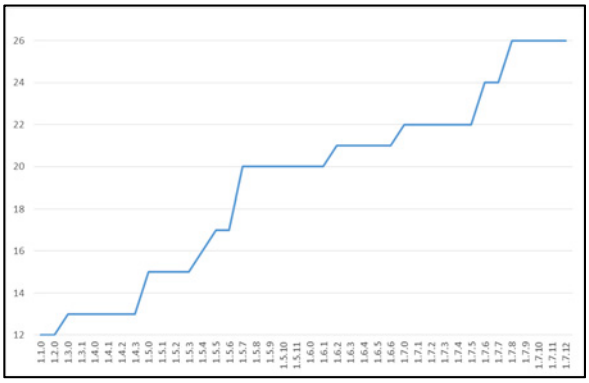
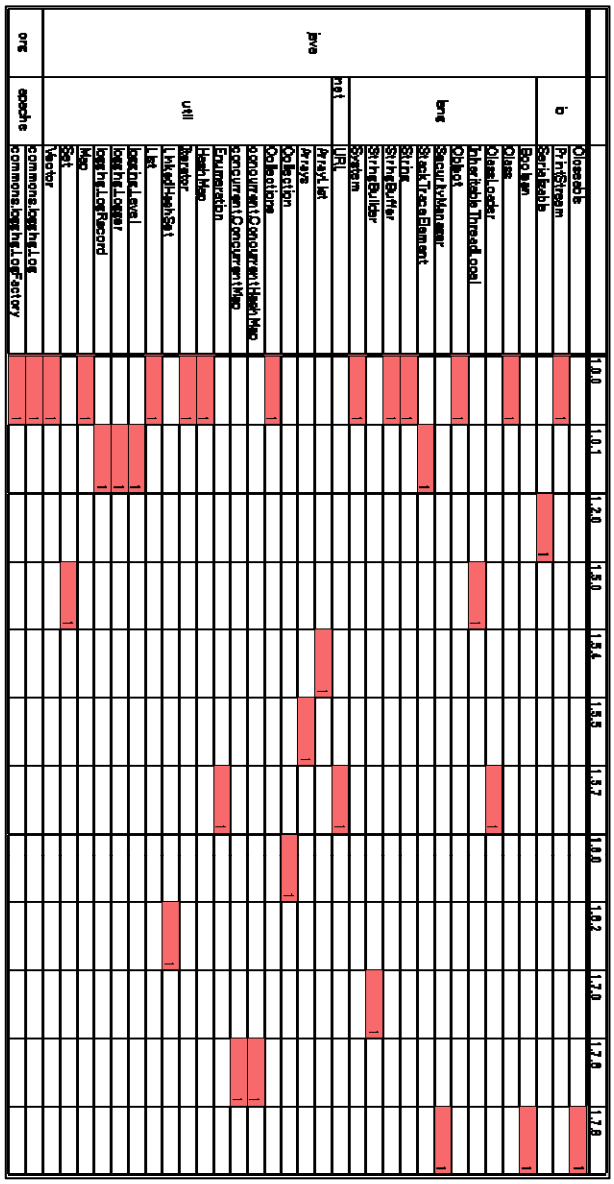


図 9　SL4J の使用ライブラリクラス数の推移

使用ライブラリクラスの推移を表 3 に示す．表より，例えば 1.7.6 において concurrent.ConcurrentHashMap および concurrent.ConcurrentMap が新たに使用されていることが分かる．これらのクラスは，HashMap や Map クラスを使った場合と比較して，複数のスレッドが走っている場合の安全性が増しており，性能も向上しているため，設計・実装上

表 3　SLF4J における使用ライブラリクラスの推移



の改良が行われたことがうかがえる．

5. おわりに

本稿では，ソフトウェア開発を定量化・可視化する 2 つの方法を提案した．今後は，今回対象としたプロジェクトのようによく使われているプロジェクトではなく，あまり使われていないプロジェクトなど様々なプロジェクトを計測し特徴を明らかにし，ソフトウェア間の評価を行いたい．

謝辞　本研究の一部は，JSPS 科研費基盤研究（C）課題番号 26330086 の補助を受けた．

参考文献

1) Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E.: Comparison and Evaluation of Clone Detection Tools, IEEE Trans. Soft. Eng., Vol.33 No.9, pp.577-591 (2007).
2) Canfora, G., Cerulo, L., Penta, M. D.: Identifying Changed Source Code Lines from Version Repositories, *Proc. 4th International Workshop on Mining Software Repositories* (MSR2007), pp.14-22 (2007).
3) 独立行政法人情報処理推進機構: 第 3 回オープンソースソフトウェア活用ビジネス実態調査　調査報告書 (2010).
4) 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター: ソフトウェア改良開発見積もりガイドブック～既存システムがある場合の開発～，オーム社 (2007).
5) Johari, K. and Kaur, A.: Effect of software evolution on software metrics: an open source case study, ACM SIGSOFT Software Engineering Notes, Vol. 36, No. 5, pp.1-8 (2011).
6) Kakimoto, K., Monden, A., Kamei, Y., Tamada, H., Tsunoda, M., Matsumoto, K.: Using software birthmarks to identify similar classes and major functionalities, Proc. 3rd International Workshop on Mining Software Repositories (MSR2006), pp. 171-172 (2006).
7) Kamiya, T.: CCFinderX, http://www.ccfinder.net/ccfinderx-j.html
8) Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, IEEE Trans. Software Engineering, Vol. 28, No. 7, pp. 654-670 (2002).
9) かぞえチャオ - 多機能ステップカウンタ, http://homepage2.nifty.com/fortissimo/
10) Neamtiu, I., Foster, J. S., Hicks, M.: Understanding source code evolution using abstract syntax tree matching, Proc. International Workshop on Mining Software Repositories, pp.1-5 (MSR2005).
11) 野村佳秀: 米国大学および国内企業での OSS プロジェクト活用の実際，ソフトウェアエンジニアリングシンポジウム 2013 ワークショップ，オープンソースソフトウェア工学セッション (2013). https://sites.google.com/site/sesjp2013/workshop#ws-5
12) Simmons, M., Vercellone-Smith, P. and Laplante, P.: Understanding open source software through software archeology: The case of nethack, Proc. SEW ’06, pp. 47–58 (2006).
13) Suh, S. and Neamtiu, I.: Studying software evolution for taming software complexity, Proc. Australian Software Engineering Conference, pp. 3–12 (2010).
14) Syeed, M., Hammouda, I., and Syatä, T.: Evolution of Open Source Software Projects: A Systematic Literature Review, Journal of Software, Vol.8, No.11, pp.2815-2829, (2013).
15) Tamada, H.: Stigmata – Java birthmark toolkit, available at http://github.com/tamada/stigmata/
16) 牛窓朋義，門田暁人，玉田春昭，松本健一: 使用クラスに基づくソフトウェアの機能面からの分類，信学技法，ソフトウェアサイエンス研究会, No. SS2009-17, pp.31-36, (2009).