卒 業 研 究 報 告 書

題　目　　プログラミング言語の習得及び理解に
　　対するリポジトリマイニングに基づく分析

指導教員　　水野　修　准教授

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　10122505

氏　　名　　山本　滉明

平成27年2月13日提出

プログラミング言語の習得及び理解に
対するリポジトリマイニングに基づく分析

平成 27 年 2 月 13 日　　10122505　山本　滉明

概　要

ソフトウェアを開発する際において、プログラマはソフトウェアを作成する為の専用の言語 (以下、プログラミング言語) を習得する必要がある。しかし、プログラミング言語の文法の非日常性や単語の構成手法等の難しさ等から、その習得は容易ではない。このため、今日のプログラマたちにとって、プログラミング言語の習得・理解に関する容易性の議論は興味の尽きぬ話題となっている。

また、この議論に関連して「プログラミング言語を全く理解していない初心者がプログラミングを始めるためにどんなプログラミング言語を学べば良いのか」という問いに対して、Eric S. Raymond は、Web 上の記事「How to become a hacker」において、「本当に何も知らない場合、HTML を始めに学び、更に興味があるなら Python を、次に perl を学び、そして C 言語を学んでみてはどうだろうか？」と提案している。

本研究では、この提案に関して、「本当に何も知らない場合、HTML の次に何を学ぶべきか」についても考察を行う。

そこで、2012 年時点での Github に存在する約 8000 リポジトリのデータを元に 2 つ以上の言語を扱うプログラマとその利用言語との間における編集行数の比、編集行数の合計、利用人数を比較し、その結果に基づき、プログラミング言語の習得・理解に関する分析・考察を行なった。

結果として、Raymond が主張する「本当に何も知らない場合、HTML を始めに学び、更に興味があるなら Python を、次に perl を学び、そして C 言語を学んでみてはどうだろうか？」という提案に関して若干の議論の余地が存在しうるという事が分かった。

本報告書では本分析の結果と、それらを元に「本当に何も知らない場合、HTML の次に何を学ぶべきか」という議論に関して一例を提案する。

目　次

1. 緒言

本報告書の目的は，2 以上のプログラミング言語を扱うプログラマの書いたコードから．プログラミング言語の習得に関する容易性の例を提案することにある。

プログラマの間では、「全く何も知らない人にどのプログラミング言語を薦めるべきか」だとか、「○○という言語の次にはどの言語を学べばよいのか」という話題が昨今に渡り常時繰り広げられている。

不幸なことに、この議論に関しては其々なプログラマの主観が入り乱れており、加えて、定量的評価を行なっている論文は少ない。従って、この議論について定量的評価を基に一例を提示する事は容易ではない。

ただし、一つの推奨基準として、「伽藍とバザール」[?] を執筆した Eric S. Raymond は Web 上の著書「How To Become A Hacker」[?] において、「本当に何も知らない場合、HTML をまず初めに学び、更に興味があるのであれば、次に Python を、次に Perl を、そして C 言語を学べばよいだろう。」と提案している。

そこで本研究では、2 つ以上のプログラミング言語を扱ったプログラマを 2012 年時点での github に存在する約 8000 リポジトリから抽出し、彼らの扱った言語と、それぞれの言語の編集行数から 2 つの言語間の編集行数の比を計算し、プログラミング言語を習得する際の提案する。

2. 研究背景

「伽藍とバザール」[?] 等を執筆した Eric S. Raymond は，Web 上の著書「How To Become A Hacker」[?] において、「私が初めに学ぶべき言語は何か？」という問いに対して次のように答えている．

> HTML if you don't already know it. There are a lot of glossy, hype-intensive bad HTML books out there, and distressingly few good ones. The one I like best is HTML: The Definitive Guide.
>
> But HTML is not a full programming language. When you're ready to start programming, I would recommend starting with Python. You will hear a lot of people recommending Perl, but it's harder to learn and (in my opinion) less well designed.

ここで，Raymond は HTML をまず薦め，その後に Python を学ぶべきだと述べている．確かに、HTML は完全なプログラミング言語ではないが、しかし、プログラミングを学びたいと考えるユーザーにとっては GUI が言語の一つとして実装されているあたり、推奨できる言語ではある。Python もまた非常に文書化が進んでいるため、推奨される言語ではある。しかし、この答えは主に Raymond 自身の主観的な経験に基づいた推薦であり，文書化の度合い定量的な評価に基づいたものではないため客観性に欠ける．

そこで、「特定のプログラミング言語を学んだ後にどんなプログラミング言語を学ぶべきか」について、本研究では定量的評価に基づいてその一例を提案する。

3. 定量的評価

3.1 各用語の説明

本研究では、幾つかの独自に利用される用語を定めるものとする。

特定の言語で記述されたソフトウェア・プログラムのソースコードを「コード」もしくは「ソースコード」と呼称する。ソースコードの記述言語を「プログラミング言語」もしくは「言語」と呼称する。

ある特定の言語 l に関して、その言語を利用したコードに関して、追加したコードの行数を E_{add} とし、削除したコードの行数を E_{del} とする。この時、式 3.1 を定め、式中の E_l で示された数値を「編集行数」と呼称する。

$$E_l = E_{add} + E_{del} \tag{3.1}$$

ある特定の 2 言語に関して其々の編集行数を E_l 及び E_m とする。この時、式 3.2 を定め、式中の $R_{l,m}$ を「編集行数の比」と呼称する。

$$R_{l,m} = \begin{cases} \frac{E_l}{E_m}, & \frac{E_l}{E_m} < 1 \text{ の場合} \\ 1, & E_l = E_m \text{の場合} \\ \left(\frac{E_l}{E_m}\right)^{-1}, & \frac{E_l}{E_m} > 1 \text{ の場合} \end{cases} \tag{3.2}$$

加えて、この時の式 3.3 にて示される $S_{l,m}$ を「編集行数の合計」と呼称する。

$$S_{l,m} = E_l + E_m \tag{3.3}$$

更に、この時の該当の言語の組み合わせの利用者式 3.4 にて示される X_i に関して、3.5 にて示される $U_{l,m}$ を「利用人数」と呼称する。

$$X_i = 1 \tag{3.4}$$

$$U_{l,m} = \sum_i X_i = \sum_i 1 \tag{3.5}$$

上記数式に加えて、次の用語を定める。

「極めて均一な編集行数の比」もしくは「編集行数の比が極めて均一」であるとは、式 3.2 中の $R_{l,m}$ が式 3.6 を満たす状態とする、

$$R_{l,m} \geq 0.90 \tag{3.6}$$

「一定の編集行数の比が保たれている」が保たれているとは、式 3.2 中の $R_{l,m}$ が式 3.7 を満たす状態とする。

$$R_{l,m} \geq 0.60 \tag{3.7}$$

「十分に大きな編集行数の合計を有する」とは、式 3.3 中の $S_{l,m}$ が式 3.8 を満たす状態とする。

$$S_{l,m} \geq \left(0.3 \times 10^8\right) \tag{3.8}$$

式 3.8 に関して、言い換えれば、編集行数の合計が 3,000 万行を超えれば「十分に大きな編集行数の合計を有する」とする。

「多くの人に利用されている」とは、式 3.5 にて示される $U_{l,m}$、つまり利用人数が式 3.9 を満たす状態とする。

$$U_{l,m} \geq 1000 \tag{3.9}$$

3.2 評価の基準

定量的評価の基準に関しては 2 種類のプログラミング言語の編集行数の比と編集行数の合計、及び利用人数に基づいて評価する。このような評価基準を設けることによって、以下の事柄を検証することが可能である。

- 編集行数の比に基づいて、バランスよく言語を習得していかどうか
- 編集行数の合計に基づいて、十分な量を記述しているかどうか
- 利用人数に基づいて、十分に多くの人に利用されているかどうか

言い換えれば、次のように評価することが可能である。

- プログラミング言語の編集行数の比が 1 に近ければ、バランスよく言語を習得していると思われる。即ち、極めて均一な編集行数の比を有していると言える。
- プログラミング言語の編集行数の合計が大きければ、十分な量を記述していると思われる。即ち、十分に大きな編集行数の合計を有していると言える。
- 利用人数が大きければ、十分に多くの人に利用されている。

3.3 評価の方法

評価の方法については、次の方法にて評価を行うものとする。

1. 2 種類の以上の言語の利用者に関して、利用している全てのプログラミング言語から 2 つの組み合わせ全てに対して、編集行数の比を求める。
2. 上記手順から得られた言語の組み合わせを式 3.10 に基づいてソートする。
3. ソート後、利用者人数に関して、式 3.11 に基づいてフィルタリングを行う。

2 において、対象となる言語の組み合わせ 2 つ、たとえば、「言語 1 と言語 2」、「言語 3 と言語 4」等の組み合わせ 2 つに関して、編集行数の比を其々、R_1、R_2、編集行数の合計を其々S_1、S_2 とした時、そのソート基準 $N(R_1, R_2, S_1, S_2)$ は式 3.10 にて定められる。

$$N(R_1, R_2, S_1, S_2) = \begin{cases} R_2 - R_1, & |R_2 - R_1| > 0.05 \text{ の場合} \\ S_2 - S_1, & |R_2 - R_1| \leq 0.05 \text{ の場合} \end{cases} \tag{3.10}$$

尚、式 3.10 中において、0.05 の「遊び」を設けている。これは、十分に大きな編集行数の合計を有しており、編集行数の比の差が無視できる程度まで小さいにもかかわらず、遊びを設けなかったために候補外へ追い出されるという事態を軽減する事を目的としている。

3 については、利用人数が少ない言語を排除するために必要とされ、式 3.11 を満たす。但し、式 3.11 中の n は 0 を含む自然数であり、$U_{l,m}$ は利用人数を示す。

$$U_{l,m} \geq n \tag{3.11}$$

4. 実験

4.1 実験に利用したハードウェア

実験では、次のハードウェアを利用した。

- Intel(R) Xeon(R) CPU E5506 と 48GB の RAM を搭載したワークステーション

4.2 実験に利用したソフトウェア

実験では、次のソフトウェアを利用した。

- Python
- NodeJS
- pygit2 [?]
- RabbitMQ [?]
- Celery [?]
- Ohcount [?]
- ArangoDB [?]

Python は米国においてデータマイニングを行う際に利用されるプログラミング言語である。データマイニング以外にも、ウェブアプリを構成する際にも利用され、ruby のように汎用性が高いスクリプト言語である。

NodeJS は本来、クライアントサイドスクリプティング言語の側面しか持つことがなかった JavaScript をサーバーサイドスクリプティング、あるいは UNIX プログラミングを行う事を可能とするユニークなイベント駆動型プログラミング言語である。

github では git を用いてリポジトリの管理を行なっているため、リポジトリのコミットログに編集の差分が存在する。この差分を分析するために pygit2 と呼ばれる、libgit の python バインディングを利用する。

尚、本研究ではマイニングを行うデータが非常に膨大であるため、共通した作業を同時並行させる仕組み、及び、万が一、作業が中断されても復旧させる事が可能な仕組みが必要となった。そこで、メッセージキューイングシステムである RabbitMQ を利用し、受信されるメッセージに応じて Celery のタスクを実行するようにする。

このようにすることによって、万が一、Celery のタスクを実行できなかった場合、即ち、タスクを実行するサーバーがダウンしていた場合でも、メッセージが未送信となるため、作業は中断されても復旧させることが可能となる。

Ohcount はソースコードから利用言語、コード行数、コメント行数等を検知することが可能なソフトウェアであり、本研究ではソースコードから利用言語を判別する際に利用される。

言語の検知やコミッタの名前等や、編集行数等の格納にはデータベースを利用する。本来は関連データベースである「MariaDB」、「Oracle Database」、あるいは「PostgreSQL」等を利用する事が一般的とされているが、本研究では見落としがちな潜在的要素を洗い出すために ArangoDB を利用するものとする。このデータベースは、従来の RDBMS とは異なり、スキーマレスかつグラフ的なデータベースを構築することが可能である。言い換えれば、「人」という頂点と「言語」という頂点に「編集行数」という重み付きの辺を定義させることによって、見落としがちな潜在的要素を確認することが可能となる。

4.3 実験に際して作成したソフトウェア

実験に際して、まず、リポジトリのコミット履歴に保存されているソースコードの差分から、次の情報を引き出す必要がある。

- 当該コミットを行なった人物の名前
- 当該コミットを行なった人物のメールアドレス
- 当該コミットで編集したソースコード其々に関して、追加した行数
- 当該コミットで編集したソースコード其々に関して、削除した行数
- 当該コミットで編集したソースコード其々に関して、そのソースコードの利用言語

従って、まず、上記に挙げた情報を抽出し、データベースに記録させるソフトウェアが必要である。

次に、記録したデータの内、2 種類以上の言語を利用するコミッタに着目し、次の情報を引き出す。

- 利用言語のペア
- 編集行数の合計
- 編集行数の比

尚、編集行数の合計及び編集行数の比の関しては式 3.2 及び 3.3 に定義されている通りである。当該情報を引き出した後、何らかのシリアライズ可能なデータ形式にて出力するソフトウェアが必要となる。本研究では JSON 形式にて出力するソフトウェアを作成した。

次に、出力されたデータはソートされていないデータであるため、目視による評価が容易ではない。そこで、式 3.10 に従ってデータをソートするソフトウェアを作成する必要がある。

従って、実験に際して作成したソフトウェアは次のとおりである。

- リポジトリのコミット履歴に保存されているソースコードの差分からコミッタの名前、メールアドレス、編集行数及び利用言語を記録するソフトウェア
- 記録したデータ中、2 種類の言語を利用するコミッタに対して、その言語のペアとの編集行数の比と、編集行数の合計を記録し、JSON 形式にて出力するソフトウェア
- 式 3.10 に従ってデータをソートするソフトウェア

4.4 仮説

実験を行うにあたり、次のような仮説を設定した。

- もし、Raymond の述べたことが真実であるならば、HTML と Python の関係において、式 4.1 及び式 4.2 が成立し、Python と Perl の関係において式 4.3 及び式 4.4 が成立する筈である。
- もし、Raymond の述べたことが真実でないなら、HTML とその近傍の言語において、一定の編集行数の比が保たれているか、極めて均一な編集行数の比を有しており、かつ十分に大きな編集行数の合計を有しているものが存在しているはずである。そしてそれはどれか。

式 4.1 に関して、編集行数の比を $R_{HTML,Python}$ とする。

$$R_{HTML,Python} \geq 0.90 \tag{4.1}$$

式 4.2 に関して、編集行数の比を $S_{HTML,Python}$ とする。

$$S_{HTML,Python} \geq \left(0.3 \times 10^8\right) \tag{4.2}$$

式 4.3 に関して、編集行数の比を $R_{Python,Perl}$ とする。

$$R_{Python,Perl} \geq 0.90 \tag{4.3}$$

式 4.4 に関して、編集行数の比を $S_{Python,Perl}$ とする。

$$S_{Python,Perl} \geq \left(0.3 \times 10^8\right) \tag{4.4}$$

つまり言い換えるなら、もし Raymond の述べたことが真実であるなら、次の関係が成立する。

- HTML と Python において、極めて均一な編集行数の比を有する。
- HTML と Python において、十分に大きな編集行数の合計を有する。
- Python と Perl において、極めて均一な編集行数の比を有する。
- Python と Perl において、十分に大きな編集行数の合計を有する。

4.5 実験の方法

具体的な実験の方法としては、次の通りである。

1. リポジトリのコミット履歴から、コミッタの情報と、ソースコードの利用言語と編集行数を記録する。これに関しては 4.3 節にて説明された、「リポジトリのコミット履歴に保存されているソースコードの差分からコミッタの名前、メールアドレス、編集行数及び利用言語を記録するソフトウェア」を利用する。
2. 記録したデータベースのデータから 2 種類のプログラミング言語を扱うコミッタに着目し、その言語の中から 2 種類の言語の全ての組み合わせに関して、編集行数の比と、編集行数の合計を測定する。これに関しても、4.3 節にて説明された、「記録したデータ中、2 種類の言語を利用するコミッタに対して、その

言語のペアとの編集行数の比と、編集行数の合計を記録し、JSON 形式にて出力するソフトウェア」を利用する。

3. 式 3.10 に従って測定したデータをソートする。

4.6 取得すべき実験の結果

取得すべき実験の結果に関しては次の通りである。

- 利用人数を考慮せずに、単純に編集行数の比と編集行数の合計から何かデータを得ることが可能であるかを検証する。このため、編集行数の比と編集行数の合計のみで評価されたデータを取得する。このデータを「1 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数」と呼称する。
- 利用人数が 100 人以上存在する場合、何か特筆すべき点があるかどうかを検証する。このため、利用人数が 100 人以上存在する場合のデータを取得する。このデータを「100 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数」と呼称する。
- 利用人数が 1000 人以上存在する場合、何か特筆すべき点があるかどうかを検証する。このため、利用人数が 1000 人以上存在する場合のデータを取得する。このデータを「1000 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数」と呼称する。
- 評価対象とする言語のペアの内、どちらか一方に HTML が存在するものから、HTML の次に何を学べばいいかについて検討を行う。このため、利用言語の一方が HTML のデータを取得する。このデータを「利用言語の一方を HTML に絞った場合の 2 種類の言語間の編集行数の比、合計、及び利用人数」と呼称する。

4.7 実験結果

実験結果を次のように示す。

- 表 4.1 に 1 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数
- 表 4.2 に 100 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数
- 表 4.3 に 1000 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数
- 表 4.4 に利用言語の一方を HTML に絞った場合の 2 種類の言語間の編集行数の比、合計、及び利用人数

尚、表中のデータは全て上位 20 位まで示すものとする。

表 4.1　1 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数

言語 1 (l)	言語 2 (m)	編集行数の比 ($R_{l,m}$)	編集行数の合計 ($S_{l,m}$)	利用人数 ($U_{l,m}$)
java	python	0.99	140152368	1039
ruby	csharp	0.97	50113173	288
objective_c	csharp	0.99	11231234	156
javascript	xml	0.93	739662904	4402
python	php	0.93	38726629	604
python	scala	0.96	7178910	139
ada	autoconf	0.95	6542778	69
autoconf	php	0.96	6335490	159
tex	perl	0.98	6311167	188
python	erlang	0.99	4276460	98
fortranfree	tex	0.98	462755	22
visualbasic	make	0.99	241864	50
lua	erlang	1.00	87213	13
objective_j	actionscript	0.99	69631	4
bat	inc	0.99	55338	4
r	ada	0.98	43962	14
make	xaml	1.00	33114	29
awk	idl_pvwave	1.00	30508	11
make	mxml	1.00	10538	11
factor	xslt	0.98	7616	4

表 4.2　100 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数

言語 1 (l)	言語 2 (m)	編集行数の比 ($R_{l,m}$)	編集行数の合計 ($S_{l,m}$)	利用人数 ($U_{l,m}$)
java	python	0.99	140152368	1039
ruby	csharp	0.97	50113173	288
objective_c	csharp	0.99	11231234	156
javascript	xml	0.93	739662904	4402
python	php	0.93	38726629	604
python	scala	0.96	7178910	139
autoconf	php	0.96	6335490	159
tex	perl	0.98	6311167	188
ruby	java	0.91	81738046	710
assembler	html	0.90	56500603	565
autoconf	shell	0.92	31891477	1447
xml	scala	0.87	24054067	422
ruby	objective_c	0.90	20259867	495
vim	css	0.94	2661750	159
bat	visualbasic	0.91	229176	123
xml	ruby	0.85	168199215	1551
css	sql	0.83	33557096	778
perl	css	0.86	23199629	873
ruby	autoconf	0.87	16332176	211
css	lua	0.88	3212147	105

表 4.3　1000 人以上のコミッタが利用している 2 種類の言語間の編集行数の比、合計、及び利用人数

言語 1 (l)	言語 2 (m)	編集行数の比 ($R_{l,m}$)	編集行数の合計 ($S_{l,m}$)	利用人数 ($U_{l,m}$)
java	python	0.99	140152368	1039
javascript	xml	0.93	739662904	4402
autoconf	shell	0.92	31891477	1447
xml	ruby	0.85	168199215	1551
xml	cpp	0.76	701242947	3818
java	xml	0.80	329334930	3324
php	xml	0.82	202982438	1713
xml	csharp	0.70	184348163	1470
perl	autoconf	0.73	23580416	1007
python	html	0.66	220573412	3241
ruby	javascript	0.65	217240299	2537
html	ruby	0.64	141577404	1970
shell	perl	0.58	39662313	1663
php	javascript	0.54	330332935	2088
java	html	0.55	196736357	1853
html	objective_c	0.57	58494947	1125
cpp	javascript	0.49	340405339	2380
javascript	c	0.49	331544296	2384
php	html	0.51	174690684	1839
java	javascript	0.46	243447228	1514

表 4.4　言語の一方が HTML の言語間の編集行数の比、合計、及び利用人数

言語 1 (l)	言語 2 (m)	編集行数の比 ($R_{l,m}$)	編集行数の合計 ($S_{l,m}$)	利用人数 ($U_{l,m}$)
metapost_with_tex	html	0.99	1289	1
assembler	html	0.90	56500603	565
scala	html	0.75	14398712	259
html	scheme	0.76	11422581	133
html	sql	0.72	45018428	865
python	html	0.66	220573412	3241
html	ruby	0.64	141577404	1970
html	erlang	0.63	3667983	114
html	csharp	0.63	80077225	559
html	emacslisp	0.62	24122703	260
java	html	0.55	196736357	1853
html	objective_c	0.58	58494947	1125
haskell	html	0.54	4298478	64
html	tcl	0.55	3854207	80
html	lisp	0.55	2041152	32
php	html	0.51	174690684	1839
html	fortranfree	0.51	711012	46
html	fortran	0.53	9817	11
html	objective_j	0.49	1020203	38
html	autoconf	0.42	45701730	948

5. 結論・考察

5.1 結果全体に関して

結果から Python と Java が極めて均一な編集行数の比と十分に大きな編集行数の合計を有している事が分かる。加えて、この 2 つの言語の組み合わせは多くの人に利用されている。この事は、Python と Java が同じ程度理解されていると考えても差し支えないと言える。

次に、HTML は XML と類似した構文を持つ。このため、HTML が XML として認識されていても、不自然ではない。従って、クライアントサイドスクリプティング言語である Javascript と XML として認識された HTML との間で、極めて均一な編集行数の比と十分に大きな編集行数の合計を有しており、かつ多くの人々に利用されていると考えられる。

そして、HTML と親和性のあるプログラミング言語の筆頭として、TEX の Metapost が上がっているが、これはたった 1 人しか利用しておらず、HTML の次に学ぶべき言語としてこれを学ぶべきという所の信憑性は薄い。

加えて、Raymond が主張する所の「HTML の次に Python を学ぶ」事を推奨している点に関して、HTML の方が Python の 3 倍ほど多く書かれている点と、十分に大きな編集行数の合計を有し、かつ、利用人数も 3,000 を超えているという点で概ね同意できる。しかし、Python の次の言語に学ぶべき言語においては Java が筆頭となっている他、php が候補として上がっており、perl は候補の外にあるという事が分かった。

次に、C 言語等はネイティブコードを生成させるため、コンパイルという作業が必要となる。この作業を自動化させるために GNU Make や Ninja といったオートメーションビルダを利用する事が一般的である。しかし、コンパイルさせるファイル数が非常に膨大であったり、ライブラリ間の依存関係を解決させるためには GNU Make 等のオートメーションビルダでは役不足となる場合が多い。そこで、ビルド司令ファイル(注 1) を生成させる前にライブラリ間の依存関係を解決させ、そしてビルド司令ファイルを作成する必要がある。この時に利用されるのが autoconf 等のソフトウェ

(注 1)：例えば GNU Make なら Makefile、Ninja なら build.ninja

アであり、このソフトウェアはライブラリ間の依存関係の解決と、GNU Make の為のビルド司令ファイルを作成する為のスクリプトである「configure」を生成する。加えて、configure はシェルスクリプトであるため、autoconf 用スクリプトが存在するならば、シェルスクリプトも同時に利用されていることに他ならない。従って、autoconf と shell との間は非常に均一な編集行数の比と、十分に大きな編集行数の合計を有する。

次に、XML はデータのシリアライズ用言語として比較的早期に発明され、利用されている。加えて、C++言語言語もまた、比較的早期に発明・利用されて、かつオブジェクト指向的な言語であるため、XML を比較的容易に扱うことが可能である。このため、XML と C++は一定の編集行数の比を保ちながら、十分に大きな編集行数の合計を有し、多くの人に利用されていると思われる。対して、XML と ruby も一定の編集行数の比を保ち、十分に大きな編集行数の合計を有し、多くの人に利用されているものの、Ruby は C++よりも 10 年ほど後に開発されたものであったため、C++と XML を利用する人よりも利用人数が少ない。

次に、C や C++と Javascript は一見すると何ら関連性は無いように見える。しかし、Qt Company が保有するアセットである Qt と呼ばれるフレームワークは QML と呼ばれる独自の言語を自身のフレームワークに組み込んでおり、その言語から Javascript が利用できる、Microsoft 社の.netFramework の C++バインディングから Javascript を利用する、あるいは、Javascript をプラグイン開発用言語として組み込む等を行うことがあるため、一定の編集行数の比すら有しないものの、十分に大きな編集行数の合計を有し、多くの人に利用されていると考えられる。

以上の事から、次の事が言える。

- Python と Java の組み合わせは極めて均一な編集行数の比と十分に大きな編集行数の合計を有し、多くの人に利用されている。
- XML と Javascript の組み合わせは極めて均一な編集行数の比と十分に大きな編集行数の合計を有し、多くの人に利用されている。
- HTML と TEXMetapost は極めて均一な編集行数の比を有するが、利用人数が 1 しかないため、HTML の次に学ぶべき言語としての信頼性は薄い。
- autoconf と shell は autoconf の性質上、その関係において、極めて均一な編集行

数の比と十分に大きな編集行数の合計を有している。

- XML と C++は一定の編集行数の比を保ちつつも、十分に大きな編集行数の比を有し、多くの人に利用されている。
- XML と Ruby も XML と C++の場合と同様であるが、XML と C++と比べて利用人数が少ない。
- C++と Javascript は十分に大きな編集行数の合計を有し、多くの人に利用されている。
- C と Javascript は十分に大きな編集行数の合計を有し、多くの人に利用されている。

5.2 Python と Java の関連性に関して

Python と Java との間には、極めて均一な編集行数の比と十分に大きな編集行数の合計を有し、多くの人に利用されているという関係性がある。これについては、Python のインタープリタの一つとして Jython と呼ばれるインタプリタが存在する事が理由として考えられる。

Jython は Java で記述された Python のインタプリタであり、Java で記述されているため、マルチプラットフォームで、他の Java アプリケーションと組み合わせて利用する際に便利である。従って、結果のような関連性を生み出すに至ったのではないかと考察される。

5.3 C#と Objective C の関係に関して

C#と Objective C の間に「極めて均一な編集行数の比」を有する関係が現れている事が分かる。C#は元々、Microsoft 社が自社の OS である Windows のプログラミングに特化したフレームワークである、.netFramework を利用するために開発されたプログラミング言語であるに対して、Objective C は Apple 社が自社の OS である Mac OS と OS X をプログラミングするために作成されたプログラミング言語であり、両者とも何らかの互換性を有しているとは思われない。

しかし、C#は現在、ECMA にて標準化されており [?]、Windows 以外のプラットフォームへの移植が可能である。加えて、Windows 以外のプラットフォームにおけ

るC#の実装としてはMono[?]が筆頭としてあげられ、最早C#はWindowsにロックインされた言語ではないと言える。従って、C#でMac向けアプリケーションを作成したが、Apple製モバイルに関する移植性の問題、特に、ライセンスに関して問題が生じうる可能性があるため、Objective Cを利用してアプリを作成するという事が考えられる。

ちなみに、Pythonがサーバーサイドプログラミング言語として利用される事が考案され始めたのは2011年頃という事もあり、PythonとJavascriptは結果に現出するに値するほどの関連性を有していない。

5.4 RubyとJavascriptの関連性に関して

RubyとJavascriptの関係について、一定の編集行数の比を保っており、十分に大きな編集行数の合計を有し、多くの人に利用されているという関係性が現れている。これについてはRuby on Rails[?]というRubyをサーバーサイドアプリケーションの為の言語として利用できるようにするためのフレームワークが存在する事が考えられる。

Ruby on RailsはRubyをサーバーサイドプログラミング言語として扱えるようにするためのフレームワークであるが、ウェブアプリを扱う場合、サーバーサイドプログラミング言語の習得と同時にクライアントサイドプログラミング言語の習得をする必要がある場合がある。今回の関連性はサーバーサイドとクライアントサイドの関係性が顕著に示された為に現出したものであろうと考えられる。

5.5 HTMLの次に学ぶべき言語

結果の統計データから、HTMLの次に学ぶべき言語はPythonもしくはRubyである事が判明した。この2つの言語はPython対HTMLにおいて一定の編集行数の比が保たれている点、Ruby対HTMLにおいても一定の編集行数の比が保たれている点、また何れの言語も十分に大きな編集行数の合計を有することから、よく利用されており、また利用人数も他の言語と比べて多い事が分かる。

加えて、HTMLがXMLと誤認しやすい事を考えると、HTMLの次に学ぶ事が推奨される言語はC++と考えることも大いにありうる。

従って、Raymond の主張する所の「HTML の次に Python を学ぶべき」という点に関しては部分的に正しいと言える。しかし、C++や Ruby についての言及が無かったという点で正しくないとも言える。

5.6 Python の次に学ぶべき言語

Raymond は Python の次に学ぶべき言語として Perl を挙げているが、結果のデータから Python の次に学ぶべき言語は Java もしくは PHP が適切であると言える。両者の言語共、編集行数の比が極めて均一であり、十分に大きな編集行数の合計を有している。しかし、Python と Java はより 1,000 ほどの利用人数を有する事に対して、Python と php は 600 程度しか利用人数を有していないため、Python の後に php を学ぶという所の人気はやや低いと言える。

従って、Raymond の主張する所の「Python の次に Perl を学ぶこと」を薦める旨は正しくないと言える。

5.7 Ruby の次に学ぶべき言語

Raymond は HTML の次に学ぶべき言語の候補として Ruby を上げなかったが、Ruby の次に学ぶべき言語に関しては、結果のデータからその見解を得ることが可能である。

結果のデータによれば、Ruby の次に学ぶべき言語は、Javascript、Java、あるいは C# である。Javascript に関しては人気が高く、かつ一定の編集行数の比が保たれている。Java と C#に関しては Javascript よりも人気が劣る反面、両者の言語とも極めて均一な編集行数の比を有している。

5.8　C++の次に学ぶべき言語

番外の考察の一つとして、C++の次に学ぶべき言語について考察する。

まず、今日のデスクトップアプリケーションはコマンドラインインターフェイスではなく、グラフィカルユーザーインターフェイスを有している場合が一般的である。このため、MVC モデルの導入等が必須となり、それに伴ったライブラリ・フレームワークが多数存在する。具体的には、Microsoft 社の.netFramework や Qt Company 社の Qt が筆頭となる。.netFramework はビューの定義に XAML と呼ばれる言語を、Qt は QML と呼ばれる言語を利用しているが、何れの言語もビューのアニメーション、例えば、メニューをクリックすると項目群がフェードイン・フェードアウトする等のアニメーションを定義する際に、Javascript を利用することが多い。

次に、Javascript は特定のデスクトップアプリケーションだけではなく、NodeJS 等のサーバーサイドプログラミングや、クライアントサイドプログラミングといった分野においても非常に高頻度に利用されている。このため、C++を学んだあとは、結果にも現れているように、Javascript を学ぶことを考慮に入れても良いのではないかと思われる。

しかし、定量的評価としての信憑性は十分に大きな編集行数の合計を有し、多くの人に利用されているのみであって、一定の編集行数の比を保ってるとは言えないという点に注意が必要である。

6. 結言

本研究では、特定の言語の次に学ぶべき言語は何か、2 言語以上の言語を利用するプログラマが利用している言語と言語の間に何らかの関連性があるかどうかについて研究と考察を行なった。

この研究を行うにあたり、2012 年時点での github リポジトリ群、約 8000 グループに対して編集言語の判別と分析を行ない、次のような結果を得た。

- Python と Java は極めて均一な編集行数の比と十分に大きな編集行数の合計を有し、多くの人に利用されている。
- XML と Javascript の組み合わせは極めて均一な編集行数の比と十分に大きな編集行数の合計を有し、多くの人に利用されている。
- autoconf と shell は極めて均一な編集行数の比と十分に大きな編集行数の合計を有している。
- XML と C++は一定の編集行数の比を保ちつつも、十分に大きな編集行数の比を有し、多くの人に利用されている。
- XML と Ruby も XML と C++の場合と同様であるが、XML と C++と比べて利用人数が少ない。
- C++と Javascript は十分に大きな編集行数の合計を有し、多くの人に利用されている。
- C と Javascript は十分に大きな編集行数の合計を有し、多くの人に利用されている。
- HTML の次に習得することが推奨される言語は Python と Ruby
- Python の次に習得することが推奨される言語は Java もしくは PHP
- Ruby の次に習得することが推奨される言語は Javascript、Java、あるいは C#
- C++の次に習得することが推奨される言語は Javascript

次に、本研究では以下の事柄についての検証が今後の課題となる。

1. コードの複雑さの考慮
2. ソート時の編集行数の比に着目する際の「遊び」に関して、より最適な設定方法
3. より最適なソートの手法
4. 時系列の考慮

1 については「コードメトリックス」と呼称される。これについては、例えば、Java で作成されたアプリ等においては、main 関数を作成するためにアプリケーション名と同一のクラスを作成し、その内部に main 関数を作成する必要がある。この時点で、C や Swift 等と比べて高いコードメトリックスが計測される。逆に、Swift には main 関数あるいはそれに準ずるエントリポイントが存在しないため、エントリポイントが存在する C や Java 等よりも低いコードメトリックスが計測される。このため、このコードメトリックスを考慮に入れた場合、結果のデータに無視できない差異が生じる可能性がある。

2 については今回は定数である 0.05 を編集行数の比をソートする際の遊びとして利用したが、この遊びは今後、様々な方法で、例えば、全体的な編集行数の比の分散を基に特定の関数からの値を利用する、等といった設定手法が考えられる。

3 については今回は編集行数の比と合計に基づいてソートを行なったが、今後の課題である 1 を考慮した場合、そのソート基準は異なるものになると思われる。このため、新しくソートの基準を定める必要性があるもとの考えられる。

4 については本研究ではコミッタ毎にどの言語を先に利用し始めたのかを考慮していない。このため、今後の研究において、これを考慮した上で分析を行うべきである。

最後に、プログラミングを学びたい、熱意のあるプログラミング初心者にとって、「初めに学ぶべきプログラミング言語は何か」については熟練したプログラマでも意見が分かれており、彼らにとっても、非常に興味のある話題の一つである。本研究では、定量的評価に基づいて特定の言語の次に学ぶべき言語の一例を示すことができた。

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学部門水野修准教授に厚く御礼申し上げます．貴重なデータをご提供頂きました，本学情報工学部門水野修准教授に深く感謝致します．本報告書執筆にあたり貴重な助言を多数頂きました，本学情報工学専攻ソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します．