B-013

プログラミング初学者向け Git/GitHub 操作支援フレームワークの設計と実装
A Design and Implementation of Git/GitHub Operation Supporting Framework for the Novice Programmers

増田　亜里紗† 　　玉田　春昭†
Arisa Masuda　Haruaki Tamada

概要 プログラミング初学者にとって，Git の操作の学習はハードルが高い．プログラミングだけでも大変であるのに，プログラミング以外のことも考える必要があるためである．しかし，昨今の開発では，git などを用いたバージョン管理は当たり前のことであり，初学者もできるだけ早い段階で身につけることが望ましい．そこで，適切なタイミングで git 操作を推薦し，git 操作の簡易ヘルプを自動で出してくれる環境を提案する．これにより，作業のタイミングや作業内容を推薦に従って操作することで，初学者が git 操作に習熟することを目指す．そのために，提案システム kani は git リポジトリ内での CUI 操作を記録しておき，記録ごとに推薦エンジンを走らせる．推薦エンジンは，これまでのコマンドの実行履歴（ステータスコードや git のステータスを含む）から適切な git の操作を推薦する．

1. はじめに

昨今の開発では，プログラミング以外のことも開発の重要な要素である．例えば，バージョン管理，バグ管理，CI の利用，ビルド・テストの自動化，ドキュメントの執筆などであ[1]．これは，ソフトウェア開発者の育成には，プログラミング以外の教育も行う必要があることを表している．しかし，ソフトウェア開発を十分に経験していない開発者（開発初学者）が，プログラミング以外の要素の必要性を実感することは難しい．なぜなら，プログラミング以外の要素が必要になる場面に直面したことがあまりないことや，プログラミングのことを考えることに精一杯であり，他のことを考える余裕がないためである．特にバージョン管理は必要ではあるものの，複雑かつ最終成果物に直結するような利点がない[2]．そのため，開発初学者の学習モチベーションは高いとは言い難い．

そこで我々は，開発初学者のバージョン管理システム（VCS; Version Control System）の利用を支援するフレームワークを提案する．今日，VCS は様々なツールが公開されているが，その中でも我々は git に着目する．Stackoverflow（2018）[*1] や RhodeCode（2016）[*2] の調査からもわかるように，圧倒的なシェアであるためである．また，git はターミナル上で動作する CUI が基本的なシステムとして提供されているが，GUI で操作できるツールも存在する．しかし，GUI による git の操作は，学生には混乱をもたらす一方で，ターミナル上での操作が受け入れられたとの報告もある[3] [4]．そのため，提案フレームワークは，ターミナル上で動作するものとする．なお，このフレームワークは開発初学者の VCS 操作の支援であり，将来的に開発初学者が VCS の操作に慣れてくると提案フレームワークを卒業していくものとする．

以降，第 2 節で提案フレームワークに求められる要件を整理し，第 3 節で実装の詳細について記した後，第 4 節で，実装したフレームワークを評価する．第 5 節で関連研究について述べ，最後に第 6 節で本稿をまとめる．

2. 提案フレームワークに求められる要件

ここでは，提案フレームワークに求められる要件を整理する．前提として開発初学者は，VCS の存在や必要性，そして，コミット，プッシュの操作の存在は知っているものとする．ただし，どのようなタイミングでコミット，プッシュを行うかなど，その理解は十分である必要はない．そして，操作には慣れておらず，コミット，プッシュの操作方法は漠然とした理解で，何らかの資料を見ながらでなければ操作できないものとする．このような開発初学者を支援するためのフレームワークには，次に挙げる要件が必要となろう．

R1. 適切なタイミングでの VCS 操作の推薦（指示）を出せる．
R2. 操作方法を能動的に提示できる．
R3. 推薦のタイミング・内容を自由にプログラミングできる．

VCS は操作が煩雑であり，いつ，何をすれば良いかの理解が開発初学者には難しい．そのため，適切なタイミングで，どのような操作を行えば良いかの推薦（指示）を行う（R1）．そして，その操作を具体的にどのように行えば良いのかの簡易ヘルプをシステムが能動的に表示するようにする（R2）．能動的な表示とは，ユーザが自ら調べなくても，必要なヘルプメッセージを自動的に表示するものである．開発初学者の支援方法は一つではなく，対象者ごとに様々なものが考えられよう．そのため単一の支援方法ではなく，様々な支援方法を提供する必要がある．しかしその一方で，完璧な支援方法の集合を用意することは非常に困

† 京都産業大学　Kyoto Sangyo University
*1 https://insights.stackoverflow.com/survey/2018#work-_-version-control
*2 https://rhodecode.com/insights/version-control-systems-2016

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

難である．そのため，支援方法（シナリオ）を後から追加できるような枠組みの構築を目指す（R3）．

例えば，コンパイルが成功したあと，コンパイルに成功したプログラムの実行が正常に終了すると，テストが終わったと考えられるため，コミットを推薦するというシナリオを考える．このシナリオを実現するためには，ターミナル上のコマンドとそのステータスコードを記録しておく必要がある．さらに，コマンドを実行した後に，シナリオを実現するスクリプトを実行するような枠組みが必要となる．

3. 実装

3.1 非機能要求の整理

提案フレームワークを現実のシステムとして構築するに当たり，第 2 節で述べた機能要求とともに非機能要求も整理する必要がある．提案フレームワークの非機能要求を以下に挙げる．

r1. 必要最低限のデータのみ収集する．
r2. ユーザが能動的に kani を有効化する．
r3. 入力コマンドの出力を変更しない．
r4. 提案フレームワークからの出力を最小限にする．

r1 はプライバシの観点からの要件である．提案システムを実現するために収集する主なデータはコマンド履歴となる．そして，コマンド履歴を収集するための方法には様々なものがある．ただし，キーロガーのようなプライバシ侵害の恐れに繋がるような方法は採用しない[5]．また収集が不要であった場合は，即座に収集内容を破棄するほか，収集内容の外部サーバへの送信は，利用者に対して明示的に行う必要がある．以上のことから，必要最低限の収集範囲とは，Git/GitHub の操作支援フレームワークであるため，特定の git リポジトリ以下のコマンド履歴とそのステータスコードとする．

r2 も r1 と同様にプライバシの観点からの要件であり，ユーザに隠れてコマンド履歴を収集しないためである．加えて，容易に収集の on/off を切り替えられることも要件として挙げられよう．

r3 は，学習支援フレームワークであるため，本来のコマンドの出力結果を変更しないことを表している．提案フレームワークの利用者は，将来的には提案システムから卒業していくため，利用前後で本来のコマンドの出力結果が変わることは，ユーザの混乱に繋がるためである．

最後の r4 は，提案フレームワークからの出力が長すぎると，本来の出力結果がターミナル上で流れてしまい，確認するためにスクロールを要するようになる．このような手間をユーザに掛けさせないためにも，提案システムからの出力内容は，多くても 5 行程度に納めるようにする．

3.2 Git/GitHub 学習支援フレームワーク kani

第 2 節で述べた要件を満たすシステム kani を構築した[*3]．ここでは，実装の詳細について述べる．

3.2.1 コマンド履歴収集方法

kani がターミナル上のコマンド履歴とステータスコードを収集する方法として，zsh[*4] のフック機能を利用する．具体的には，コマンドの入力後，実行する直前に動作する `preexec` フックとプロンプトを表示する直前に動作する `precmd` フックを利用する．`preexec` フックでは，入力されたコマンドを取得し，`precmd` でステータスコードを取得する．また，bash[*5] では precmd は実現できるものの，preexec に相当する機能が標準では提供されていない．ただし， bash-preexec[*6] を利用することで，zsh の場合と同様のことが実現できる．fish[*7] も，イベントハンドラを利用することで同様のことが実現できる．

3.2.2 コマンド履歴の保存

R3 で挙げたように，推薦のタイミング・内容を自由にプログラミングするためには，これまでのコマンド履歴に自由にアクセスできる必要がある．そのために git リポジトリごとにデータベースを作成し，そこにコマンド履歴を保存するものとする．つまり，kani が有効化された git リポジトリの `.git` ディレクトリと同じ階層に `.kani` ディレクトリを作成する．そして `.kani/kani.sqlite` の SQLite データベースにコマンドライン履歴を保存する．保存する内容は，コマンドの内容，ステータスコードのほか，実行日時，ブランチ名，リビジョン番号である．

3.2.3 分析スクリプト

kani がインストールされたディレクトリ（`KANI_HOME`）に，`analyses` ディレクトリを用意した．この `analyses` ディレクトリ直下にある実行可能ファイルが分析スクリプトである．`precmd` フックでコマンド履歴を収集し，データベースに保存した後，分析スクリプトを辞書順に実行する．同様に，ホームディレクトリの `.config/kani/analyses` ディレクトリ直下の実行可能ファイルも分析スクリプトとして扱う．こちらの分析スクリプトは，`KANI_HOME` の分析スクリプトの実行後に辞書順に実行する．

分析スクリプトにより何らかの行動推薦があった場合は 0 以外のステータスコードで終了するものとする．そして，実行した分析スクリプトのうちのいずれかが 0 以外のステータスコードであった場合，規定の簡易ヘルプを表示する．簡易ヘルプは予めファイルに記載しておき，`$KANI_HOME/resources` 以下に置いておく．

3.3 シナリオ

ここでは初期の kani のリリースで提供している行動推薦のシナリオを示す．なお，マジックナンバーの部分はデフォルト値としている部分であり，実際の利用に際してはユーザや管理者の意向により自由に設定可能である．

*3 https://github.com/tamadalab/kani/
*4 https://www.zsh.org
*5 https://www.gnu.org/software/bash/
*6 https://github.com/rcaloras/bash-preexec
*7 https://fishshell.com/

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

各シナリオを示す前に，kani が前提とする開発初学者の行動を示す．開発初学者は，何らかのエディタを用いてソースコードを編集するものとする．そして，コンパイルやテストの実施，バージョン管理などの操作はターミナル上で行うものとする．

3.3.1 シナリオ 1: git add の推薦（変更行数が多くなった）

変更行数が多くなった場合，`git add` で編集したファイルをステージングエリア（インデックス）に置くことを推薦する．ステージングエリアとは，コミット対象となるファイルを置く場所のことであり，`git add` で対象ファイルをステージングエリアに置くことができる．なお，git ではコミットの際，変更したファイル全てをコミット対象にするオプションが存在する（`git commit -a`）．しかし，ステージングエリアの存在を意識させることで，どのファイルをコミットするのかを意識付けるため，`git add` を推薦するようにしている．

変更行数は，`git diff --numstat` でファイルごとの追加・削除行数（変更行数）が取得できる．1 つのファイルで変更行数が 3 行以上の場合，`git add` を推薦する．

3.3.2 シナリオ 2: git add の推薦（取り組んでいた問題が解決した）

ソフトウェア開発を実施している間，幾度となくプログラムをコンパイルすることになる．このシナリオでは，コンパイルに成功したあと，コンパイル後のプログラムを実行し，実行に成功するとコミットを推薦する．

コンパイルの実行直後に `.kani/compilations/`コマンド名を以下に実行日時，ステータスコードと，コマンドラインを記録しておく．また，コンパイル時に出力ファイルが指定された場合は，`.kani/completions/`*execfile* に記録しておく．*execfile* に記録されたコマンドが実行された場合，終了ステータスを確認し，0 であれば，`git add` を推薦する．

3.3.3 シナリオ 3: コミットの推薦（全てのファイルがステージングされた）

編集ファイル全てがステージングエリアに置かれた時に，`git commit` によるコミット操作を推薦する．`git status -s` で各ファイルのリポジトリ上の状態を取得し，ワークツリー上のファイル数が 0，かつ，ステージングエリアに置かれたファイル数が 0 より大きい場合にコミットを推薦する．

3.3.4 シナリオ 4: git pubh の推薦（全てのファイルがコミットされた）

コミットが行われ，ワークツリーやステージングエリアに編集済みのファイルが置かれていない時にプッシュを推薦する．

現在のブランチ名を取得し，対応するリモートブランチ名を取得する．対応するリモートブランチが存在しない場合，まだ push していないため，プッシュを推薦する．そうでない場合は，ローカルブランチのコミットのうち，未 push のコミットを数え，0 より大きい場合にプッシュを推薦する．

Program 1 Headの雛形プログラム

```c
#include <stdio.h>
#include <stdlib.h>
#define DEFAULT_LINE 10

void print_head(FILE *fp, int limit) {
  // この関数を実装する．
}
void perform_each(char *file_name, int limit) {
  FILE *fp = // 読み込み専用でファイルを開く．
  if(fp == NULL) {
    perror(file_name);
    return;
  }
  print_head(fp, limit);
  fclose(fp);
}
void print_help() {
  printf("head <NUMBER> [FILEs...]\n\
NUMBER   表示する先頭行．\n\
FILE     表示するファイル．未指定の場合は標準入力．");
}
int perform(int argc, char *argv[]) {
  if(argc == 3) {
    int limit = atoi(argv[1]);
    perform_each(argv[2], limit);
  } else {
    print_help();
  }
  return 0;
}
int main(int argc, char *argv[]) {
  int status = perform(argc, argv);
  return status;
}
```

4. ケーススタディ

4.1 概要

提案フレームワークを用いて，開発初学者が 2 つの C プログラムの作成に取り組み，kani からの推薦にどの程度従ったかを測定する．測定に先立ち，git にどれくらい慣れ親しんでいるかを被験者の自己申告で 4 段階に分類している．

被験者には，1 つの練習プログラムに取り組んだ後，2 つの C プログラムの修正課題に取り組んでもらう．それぞれ，3 つのステップを設けており，各ステップに取り組む時には，当該ステップが完成した時の振る舞いも併せて示している．なお，プログラムで躓くことがないよう，取り組んでいる時にある程度手が止まる他，被験者からの要請があれば，当該ステップの正解プログラムを開示している．また，被験者はターミナルと使い慣れたエディタ，課題の手順を記した文書を読むためのブラウザを用いるものとする．このケーススタディは，情報科学を専攻する大学 4 年生 2 名（$B4_1$，$B4_2$）に実施してもらった．

4.2 課題プロジェクト

それぞれの課題プロジェクトは，GitHub Classroom を用いて配布する．被験者は次の手順で課題に取り組む．

(1)　kani をインストールする．

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

(2) 練習プロジェクト，2 つの課題プロジェクトに対して以下の内容を順に実行する．
  (a) GitHub Classroom でリポジトリを fork する．
  (b) Fork したリポジトリを clones する．
  (c) 対象リポジトリ内で kani を有効化する．
  (d) 課題の各ステップに取り組む．

4.2.1 練習プロジェクト cat

被験者に示した問題文は以下の通りである．

> コマンドライン引数でファイル名が複数個指定されます．それぞれのファイルの内容を標準出力に出力するプログラムを作成しなさい．ただし，`-n` オプションが指定された場合は，行番号とともに出力するものとします．また，コマンドライン引数に何も与えられなかった場合は，標準入力から受け取るものとします．

このプロジェクトでは，`fopen` のみが抜けている状態のプログラムが被験者に提供される．被験者は `fopen` を書いた上で，kani の指示に従い `git add`， `git commit`，`git push` の一連の流れを実施するよう実験管理者から伝えられる．そして実際に，その流れを体験した上で，次の課題プロジェクト(1)，(2)に取り組む．

4.2.2 課題プロジェクト(1) Head

被験者には，Program 1 に示すプログラムを雛形として与え，次の 3 つのステップでプログラムを改良するよう指示が与えられる．コミットのタイミングや方法の指示は行わない．

Step 1. 第 1 引数に数値，第 2 引数にファイル名が与えられます．引数に与えられたファイルを先頭から指定された行数だけ出力するプログラムを作成しなさい．

Step 2. Step 1 で作成したプログラムを改良し，第 2 引数以降に複数のファイル名を受け取るように修正しなさい．

Step 3. Step 2 で作成したプログラムを改良し，ファイル名が与えられなかった場合は，標準入力から受け取るようにしなさい．

4.2.3 課題プロジェクト(2) Copy

課題プロジェクト(1)と同様に，Program 2 に示すプログラムを雛形として与え，次の 3 つのステップでプログラムを改良してもらう．

Step 1. 引数に 2 つのファイル名が与えられます．第 1 引数で与えられたファイルの内容を第 2 引数で与えられたファイルにコピーしなさい．

Step 2. 入力ファイル名が-であった場合は標準入力から受け取り，出力ファイル名が-であった場合，標準出力に出力するようにしなさい．

Step 3. コマンドライン引数に n 個のファイルが与えられ，コマンドライン引数の最後にディレクトリ名が指定された場合，n 個のファイルを全て最後のディレクトリにコピーしなさい．

Program 2 Copyの雛形プログラム

```c
#include <stdio.h>

void print_help() {
  printf "copy <INPUT_FILE> <OUTPUT_FILE>\n\
copy <INPUT_FILEs...> <DIR>\n\
INPUT_FILE   入力ファイル．- の場合は標準入力．\n\
OUTPUT_FILE  出力ファイル．- の場合は標準出力．\n\
DIR          ファイルのコピー先のディレクトリ．\n");
}
void copy_impl(FILE *from, FILE *to) {
  // この関数を実装する．
}
int copy(char *from_file, char *to_file) {
  FILE *from = // 読み込み専用でファイルを開く．
  if(from == NULL){
    perror(from_file);
    return 1;
  }
  FILE *to = //  書き込み専用でファイルを開く．
  if(to == NULL){
    perror(to_file);
    return 2;
  }
  copy_impl(from, to);
  if(from != NULL) {
    fclose(from);
  }
  if(to != NULL) {
    fclose(to);
  }
  return 0;
}
int perform(int argc, char *argv[]) {
  if(argc == 3) {
    return copy(argv[1], argv[2]);
  }
  print_help();
  return 0;
}
int main(int argc, char *argv[]) {
  int status = perform(argc, argv);
  return status;
}
```

4.3 測定結果

4.3.1 アンケート結果

事前と事後のアンケート結果を表 1 に示す．各行に質問項目，列に被験者を並べている．

いずれの被験者も，git の習熟度に関係なく，git は難しく感じている．そして，kani の導入は容易であり，git 操作の推薦は便利であると答えている．

表 1 アンケートの結果

			$B4_1$	$B4_2$
事前	git の習熟度	高(1)〜低(4)	3	3
	git の難易度	難(1)〜易(4)	2	2
事後	kani の導入容易性	難(1)〜易(4)	2	1
	kani の利便性	高(1)〜低(4)	1	2

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

4.3.2 kani による推薦に対する行動

kani により被験者の行動は記録されており，また，kani がどのような推薦を行ったのかも記録されている．これらから，推薦回数，推薦率，そして，kani により行動が推薦された時に，どの程度その推薦に従ったかを表す推薦フォロー率を算出した．結果を表 2，3 に示す．表 2, 3 の行にあるシナリオは第 3.3 節で示した通りであり，合計行はいずれかのシナリオで何らかの推薦が行われたことを表す．

表 2, 3 をみると，シナリオ 3, シナリオ 4 の推薦回数，推薦率自体は低いものの，推薦された場合，次の行動で推薦されたコマンド（`git commit`, `git push`）を実行している．これは，推薦内容が適切であった，もしくは，被験者はこれらのコマンドの流れを一つのまとまった作業であると認識している可能性がある．また，修正すべき箇所は各ステップで多くて 6 行程度であるという作業量の小ささも，一連のコマンドを続けて行うことに繋がっている可能性がある．

一方でシナリオ 1 は，推薦率がいずれのケースであっても 0.5 より大きくなっている．その反面，フォロー率は実験 1 で 0.3 以下，実験 2 では 0.15 以下と非常に低い値になっている．これは，シナリオ 1 は少しのファイルの修正で敏感に推薦していると考えられる．被験者の行動を確認してみると，コンパイルに失敗した場合や動作確認前であっても，シナリオ 1 の推薦が行われている．このように敏感に推薦されると，被験者にとって煩わしく感じられるようになろう．それを示すように，kani に求められる機能を被験者にインタビューした結果，シナリオごとに推薦の可否を決められるようにしたい，や\texttt{git add}の推薦を消したい，という要望が出た．

シナリオ 2 の推薦率は一番高い場合でも，実験 1 の $B4_2$ で 0.448 であり，それほど頻繁に実施されていない．一方で，シナリオ 1 の場合と同じくフォロー率は実験 1 で 0.333，0.231，実験 2 では 0.1 以下と低くなっている．これは，動作確認を複数回実施し，その度にシナリオ 2 の推薦が行われているためであろう．

5. 関連研究

井上らはブランチ戦略の一つである GitHub flow[*8] の遵守率を計測する手法を提案している[6]．学生プロジェクトを対象に遵守傾向を調査し，一つのルールを違反しているとその他のルールにも違反している傾向があり，また，シンプルで開発状況が把握しやすいリポジトリは遵守率が高いことを示した．

Elsen らは開発者の良いバージョン管理活動のための自動評価ツールを作成している[7]．リポジトリ管理の良し悪しやコミットメッセージの評価などを様々なメトリクスから導出している．GitHub 上の 2 つのプロジェクトに対して適用したところ，開発者の生産性には大きな隔たりがあり，バージョン活動には一貫性がないことが示された．

これらの手法は，kani と同じく git 教育の支援として利用可能であり，併用も可能である．ただ，kani はプロジェクト活動の実施中に適用されるのに対して，これらの手法は開発活動の振り返りの時に利用するものの違いがある．

表 2 kani の推薦に対する被験者の行動

		実験 1		実験 2	
		$B4_1$	$B4_2$	$B4_1$	$B4_2$
推薦回数	シナリオ 1	15	17	22	39
	シナリオ 2	9	13	12	13
	シナリオ 3	4	4	3	3
	シナリオ 4	4	4	3	3
	合計	19	25	25	45
推薦率	シナリオ 1	0.600	0.586	0.688	0.796
	シナリオ 2	0.360	0.448	0.375	0.265
	シナリオ 3	0.160	0.138	0.094	0.061
	シナリオ 4	0.160	0.138	0.094	0.061
	合計	0.760	0.862	0.781	0.918
フォロー率	シナリオ 1	0.200	0.294	0.136	0.077
	シナリオ 2	0.333	0.231	0.083	0.000
	シナリオ 3	1.000	1.000	1.000	1.000
	シナリオ 4	1.000	0.750	1.000	1.000
	合計	0.368	0.480	0.240	0.200

初学者向けの Git のチュートリアルやガイドは世の中に数多くリリースされている[*9] [*10]．これらのどれもがチュートリアル用のプロジェクトでの学習である点が提案手法と異なる．

また，GitHub Learning Lab[*11] では，多くの学習教材がオンラインで提供されている．Git-it[*12] のように GUI アプリとして提供される場合もある．これらは課題が与えられ，その課題を指示に従って取り組むことで学習する．提案手法のように学習者の行動に対して，アクションがあるような形態では提供されていない．

6. まとめ

本稿では，初学者の git/GitHub 操作を支援するためのフレームワークを提案した．そして，フレームワークに従い，zsh/bash 上で動作する支援システム kani を実装し，4 つの git コマンドの推薦シナリオを示した．ケーススタディでは，2 名の大学生が kani を利用した時の被験者の行動を測定し，kani の動作内容を確認した．結果として，推薦率が高いものの，フォロー率が低いシナリオ，推薦率は低いものの，フォロー率が高いシナリオがあることがわかった．

今後の課題として，より大規模な評価実験の実施やシナリオの見直しが挙げられる．さらに，git 熟練者の行動を測定し，git の習熟度の計測にも取り組む．

*8 https://guides.github.com/introduction/flow/
*9 https://www.atlassian.com/git
*10 https://learngitbranching.js.org/
*11 https://lab.github.com
*12 https://github.com/jlord/git-it-electron

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

参考文献

[1] Filippo Lanubile. Collaboration in distributed software development. In *International Summer School on Software Engineering, Lecture Notes in Computer Science*, 2009.
[2] David A. Lippa. Get out of git hell: Preventing common pitfalls of git. In *Proc. 4th International Workshop on Release Engineering*, p. 22, 2016.
[3] John Kelleher. Employing git in the classroom. In *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014.
[4] Delbert Hart. A survey of source code management tools for programming courses. *Journal of Computing Sciences in Colleges*, Vol. 24, No. 6, pp. 113-114, June 2009.
[5] Seref Sagiroglu and Gurol Canbek. Keyloggers: Increasing threats to computer security and privacy. *IEEE Technology and Society Magazine*, Vol. 28, No. 3, pp. 10--17, 2009.
[6] 井上拓海, 小島遥一郎, 藤原賢二, 井垣宏. 版管理システム利用時のソフトウェア開発フロー遵守状況可視化手法の検討. 信学技法, No.SS2017-55, January 2018.
[7] Rickard Elsen, Inggriani Liem, and Saiful Akbar. Software versioning quality parameters: Automated assessment tools based on the parameters. In *Proc. 2016 International Conference on Data and Software Engineering (ICoDSE)*, 2016.

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.