GitHub を用いた開発フローの判別分析

プロジェクトマネジメントコース
ソフトウェア開発管理グループ
矢吹研究室
1242132
若月　純

謝辞

本研究を進めるにあたり，矢吹研究室矢吹太朗准教授には，多くの時間をご指導にさいて頂きました．また，先行研究を行った小野寺航己先輩をはじめ矢吹研究室の皆様には，多くの知識や示唆を頂きました．協力していただいた皆様に感謝の気持ちと御礼を申し上げます．

目次

第 1 章

序論

当研究は，GitHub を用いた開発フローの判別分析を行う．開発フローとは，開発の手順，ルールである．

GitHub を用いた開発フローは 13 種類ある．それぞれの開発フローには，メリット・デメリットがある．そのため，適切な開発フローを選択できる基準が必要である．

先行研究は，プロジェクトをリスクにより分類していた．そのため，選択基準が定性的になっていた．例を 1 つあげる．メンバのスキルが高く，大規模なプロジェクトの場合，フローが自動化されているイストフローが選択される [1].

そこで当研究は，定量的な選択基準を提供することを目指す．

GitHub 上のプロジェクトの性質を調査し，決定木分析を行う．そうすることで，同一の開発フローを選択しているプロジェクトの共通項を見つけることができる．共通項を見つけることで，開発フローを選択する基準を割り出せると考える．

第 2 章

背景

ソフトウェア開発では，複数のメンバが同時に開発する場合がある．そのため，様々な問題が発生する．課題をチーム間で適切に共有できず，進捗が見えにくくなったり，複数の人たちで 1 つの製品のソースコードを編集するため，開発内容が競合したりすることもあります．さらに複数の人たちが関わることによって，コードの質を均一化することも難しくなりますし，製品のコードの全容を把握することも難しくなります [2].

このような問題を解決するため，バージョン管理システムを用いる．バージョン管理システムとは，変更履歴を管理するシステムのことである．

バージョン管理システムの種類とトレンドを次に記述する．

2.1 バージョン管理システム

バージョン管理システムには，CVS，Apache Subversion，Git 等がある．Apache Subversion は，以下 subversion と記述する．

CVS とは，Concurrent Versions System の略称で，並行バージョン管理システムを意味します．1990 年に開発された比較的歴史の古いバージョン管理ツールです．こちらは集中型のバージョン管理システムになります [3].

Apache Subversion とは，多くの開発現場で使われていた CVS の問題点を改善するために作られたツールです．2000 年に開発され，操作が CVS と似ていることから，CVS から Subversion に乗り換えるユーザが多くいました．CVS と同じく集中型バージョン管理システムです [3].

Git とは，2005 年に開発された分散型のバージョン管理ツールです．Linux カーネルのソースコード管理のために開発されたもので，今では多くの開発者に使われている人気のバージョン管理ツールです [3].

図 2.1 と図 2.2 は，GoogleTrend を用いて描かれたものである．それぞれの検索の割合を示している．

図 2.1 と図 2.2 は，2004 年 1 月 4 日から 2016 年 1 月 23 日まで，コンピュータ，電化製品，を対象としている．図 2.1 は，すべての世界を対象としている．図 2.2 は，日本のみを対象としている．

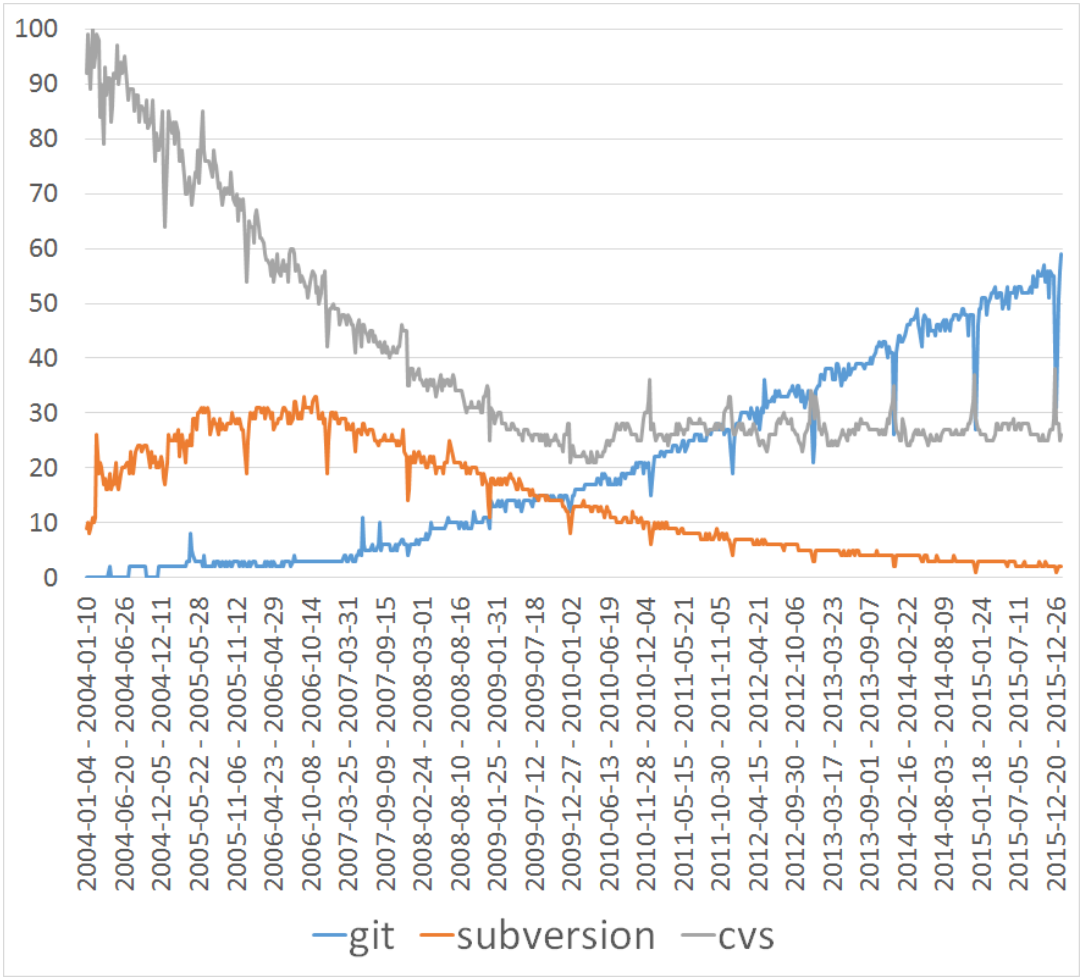


図 2.1　世界でのバージョン管理システムのトレンド

図 2.1 より，世界では，最も Git がトレンドになっていることがわかる．2004 年は CVS が最も人気が高かった．しかし，subversion の台頭により，2004 年以降人気が落ちていっていることがわかる．

Git が台頭してきたのは，2005 年ごろである．このころから，人気は上昇を続ける一方である．2009 年に，Git は，subversion の人気を超えた．2011 年に，Git は，CVS の人気を超えた．

こうして，2015 年は，最も Git が人気になっている．

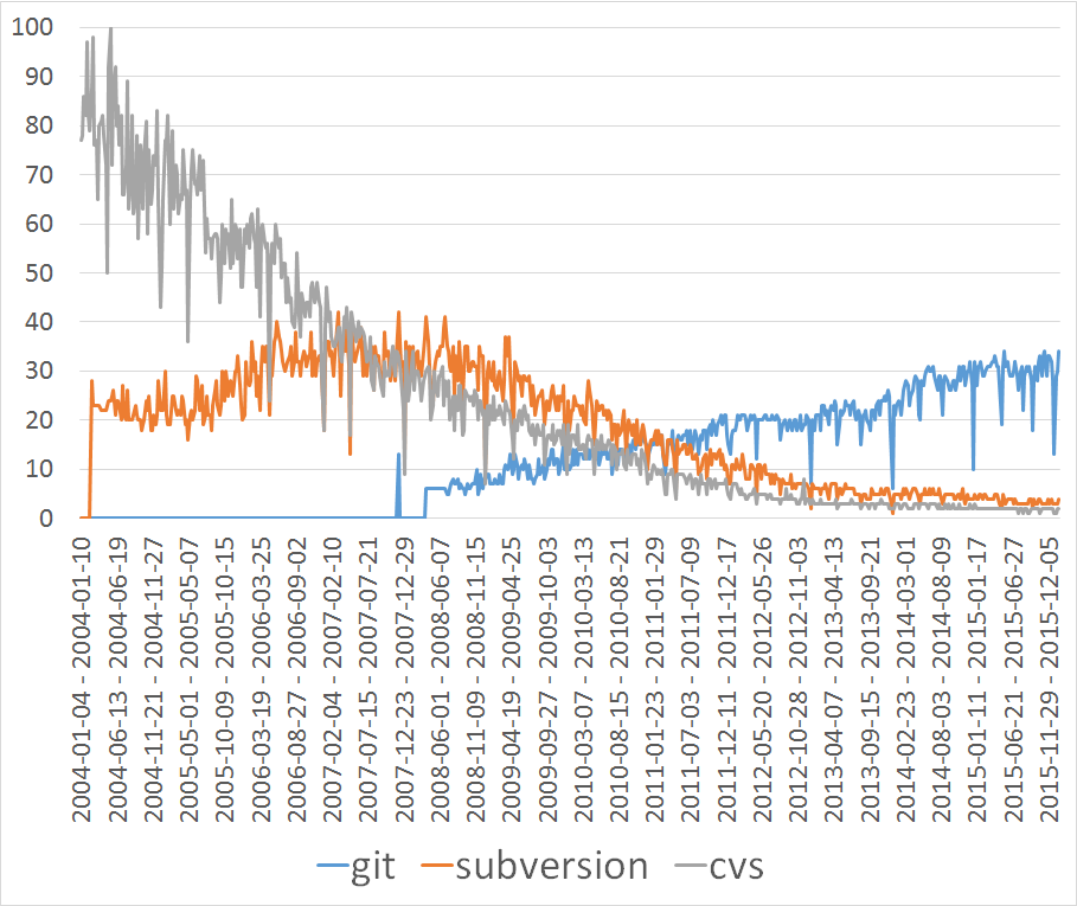


図 2.2　日本でのバージョン管理システムのトレンド

図 2.2 より，日本でも，最も Git がトレンドになっていることがわかる．2004 年は CVS が最も人気が高かった．しかし，subversion の台頭により，2004 年以降人気が落ちていっていることがわかる．

Git が台頭してきたのは，2008 年ごろである．このころから，人気は上昇を続ける一方である．2010 年に，Git は，CVS の人気を超えた．2011 年に，Git は，subversion の人気を超えた．

こうして，2015 年は，最も Git が人気になっている．

この世界と日本の状況を踏まえ，バージョン管理システムのトレンドについて考察する．

まず，世界と日本の状況を比較する．

図 2.1 と図 2.2 の技術が伸びを見せる年式に注目する．世界では，Git が 2005 年に台頭してきた．しかし，日本では Git が台頭してきたのは 2008 年からである．

ここから，日本に新しい技術が入ってくるのは，世界よりも遅いことが言える．これは，日本が海外から新しい技術を輸入しているためだと考えられる．海外の技術は，その土地の言葉で書かれている．そのため，翻訳する必要があるので，技術を輸入し，定着するまでに 3 年かかっている．

また，日本独自の開発手順があることが言える．世界のトレンドでは，Git，CVS，subversion の順である．しかし，日本のトレンドでは，Git，subversion，CVS の順である．

以上の調査により，最も人気があるバージョン開発システムは，Git であることがわかった．

そのため，本研究では，バージョン管理システムの Git を提供するサービスの GitHub を用いる．GitHub についての詳しい説明を，次に記述する．

2.2 GitHub

GitHub とは，Git リポジトリのホスティング機能をもつ．また，スローガンに Social-Coding をかかげているように，複数人で共同開発を行いやすくするための機能を提供している．機能とは，トラッキングや管理を行うための Issue，ソースコードの差分を論議するための Pull Request 等がある．

GitHub は 2008 年のサービス開始以来，年率 100 %という成長率で登録ユーザー数やレポジトリ数を伸ばしてきて，オープンソースの世界ではデファクトのプラットフォームとなっている．2015 年 6 月現在，GitHub の登録ユーザー数は 970 万，レポジトリ数は 2330 万．最近では Microsoft や Oracle といったトラディショナルな IT 企業も GitHub にレポジトリを用意するようになってきているし，広く知られたメジャーなオープンソース製品の多くが GitHub をプロジェクトのホスト先に選ぶのがここ数年のトレンドだ [4].

主な具体例として，アップルが Swift の OSS 化を GitHub で行った事例がある．Apple は，iOS や OS X の開発言語として提供してきたプログラミング言語の Swift をオープンソースとして公開した．今回オープンソース化されたのは，ソースコードを実行形式に変換するコンパイラと，プログラムの基本機能をまとめた標準ライブラリなどである．従来は，iOS/OS X 向けのソフトウェアしか開発できなかったが，Linux への移植版も公開している．そして，Apple が Swift を公開した場所も GitHub であった．Chacon 氏は，今回の Swift のオープンソース化の特徴として，コミュニティーとの連携を強調した．Swift に対するプルリクエストがすでに社外から 500 件以上も寄せられており，そのうち 349 件が統合済みだという．中には，誤字の指摘なども含まれるが，Apple の開発ツール部門のシニアディレクターで，Swift の主要な開発者である Chris Lattner 氏が「小さな改善でも，多くの人が参加するきっかけになる」とツイートしていると紹介した [5].

日本でも，セミナーや，行政活動に GitHub を用いる等，盛り上がりをみせている．具体例として，和歌山県が自治体として最初に GitHub 公式アカウントを取得した例をあげる．和歌山県は自治体としては初めて公共アカウントを取得し，ウェブサイトで公開した情報を GitHub でも公開するなど実際に活用を進めている．今回の訪問は和歌山県庁での活用に注目しているという GitHub 側が，現場の話を直接知り，今後の支援につなげたいとの要望から実現したもので，知事との対談に先駆けて，庁内のオープンデータ化を担当する職員や活動を支援する関係者らとも意見を交換した [6].

また，初心者向けのイベントが豊富なことも，人気の一つである．初心者から参加できる GitHub 習得イベント「GitHub Patchwork」が神戸で開催された．GitHub Patchwork は，GitHub を基礎から学べるオンライン教材「Git-it」をみんなで一緒に学ぶというイベント．2 年前にサンフランシスコの GitHub 本社で実施されたのをきっかけに世界各地で開催されるようになり，現在 14 カ国で 44 件のイベントが実施されている．参加者からは「以前か

ら GitHub に興味があったが，なかなか勉強する機会がなかったため，今回の参加をきっかけに活用したい」という声が多く聞かれた [7].

しかし，盛り上がりをみせている GitHub にも，問題点はある．GitHub でホスティングされたプロジェクトで作業する開発者が，自分たちが「無視され」，サポートが得られないとの苦情を公開書簡にて表明している．この公開書簡は GitHub の Web サイトの管理とサポートチャネルへの不満を綴ったもので，1000 人以上の開発者が賛同している．書簡では，「もし GitHub そのものがオープンソースだったら，われわれはコミュニティーとしてこれらの機能を実装する．われわれはこういうことを得意としているんだ！」と記している．GitHub の代表者は米 ZDNet に対し，「オープンソースは GitHub にとって極めて重要で，今回のフィードバックを真剣に受け止めている．議論されたイニシアティブの幾つかに取り組んでおり，オープンソースのメンテナーたちと積極的な関係を持ち，彼らのコミュニティーにとって GitHub が素晴らしい体験となるようにしたい」と語った [8].

ここまで，GitHub が注目を受けている背景について紹介した．様々なコミュニティが GitHub を使っている．

GitHub 特有の開発を補助する機能が多い．機能を一部抜粋して，説明を記述する．

2.3 GitHub 用語

2.3.1 リポジトリ

ファイルやディレクトリの状態を記録する場所.

2.3.2 リモートリポジトリ

手元に置いてあるローカルなリポジトリ以外の，ネット上に置かれたリポジトリのこと.

2.3.3 commit

ファイルやディレクトリの変更をリポジトリに記録する機能である.

2.3.4 clone

ネット上にあるリポジトリをローカルにコピーする機能である.

2.3.5 Origin

clone 元のリモートリポジトリのこと.

2.3.6 Push

リモートリポジトリに自分の変更履歴がアップロードされ，リモートリポジトリ内の変更履歴がローカルリポジトリの変更履歴と同じ状態にする機能である.

2.3.7 branch

履歴の流れを分岐して記録していくためのもの．分岐したブランチは，他のブランチの影響を受けないため，同じリポジトリ中で複数の変更を同時に進めていける機能である．

2.3.8 pull

リモートリポジトリから最新の変更履歴をダウンロードしてきて，自分のローカルリポジトリにその内容を取り込む機能である．

2.3.9 Pull Request

相手に対して自分の変更ををを pull してもらうように要求する機能である．

2.3.10 Revert

ステージングエリアに追加した変更を取り消す機能である．

2.3.11 タグ

コミットを参照しやすくするために，わかりやすい名前を付ける機能である．

2.3.12 Label

自由に作成でき，Issue をフィルタリングできる機能である．

2.3.13 Merge

当該ブランチに対して別のブランチの差分を取り込むことである．

2.3.14 Fork

GitHub のサービスで，相手のリポジトリを自分のリポジトリとしてコピー・保持できる機能ある．

2.3.15 Issue

ソフトウェア開発におけるバグや議論などをトラッキングして管理するために発行する．

2.3.16 デプロイ

ソフトウェアの分野で，開発したソフトウェアを利用できるように実際の運用環境に展開する．

2.3.17　リリース

プロセスを次の段階に進めることを認める機能である．

2.3.18　Watch

リポジトリに関する情報を Notifications に表示する機能である．

2.3.19　Star

リスト一覧からリポジトリを探すことが出来るようにする機能である．また，注目度を表す指標にもなる．

2.3.20　Fork

GitHub 側にある特定のリポジトリを自分のアカウント以下のリポジトリに複製する機能である．

2.3.21　人数

開発人数のことである．ここでは，Origin リポジトリにコミットした人数のことを示す．

2.3.22　MileStone

やるべきタスクの管理に Issue を用いることができるようにする機能である．

2.3.23　Wiki

簡単な記法によってドキュメントを作成，編集するための機能である．

2.4　GitHub を用いた開発フロー

GitHub を使用する手順を開発フローと呼ぶ．開発フローを網羅的に調査した先行研究がある．先行研究で調査され，本研究で用いた開発フローと，選択基準を以下に記述する．

2.4.1　GitHub フロー

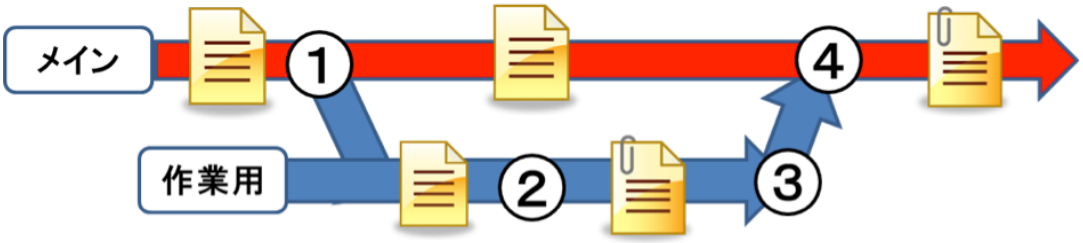


図 2.3　GitHub フロー図　出典：[1]p17

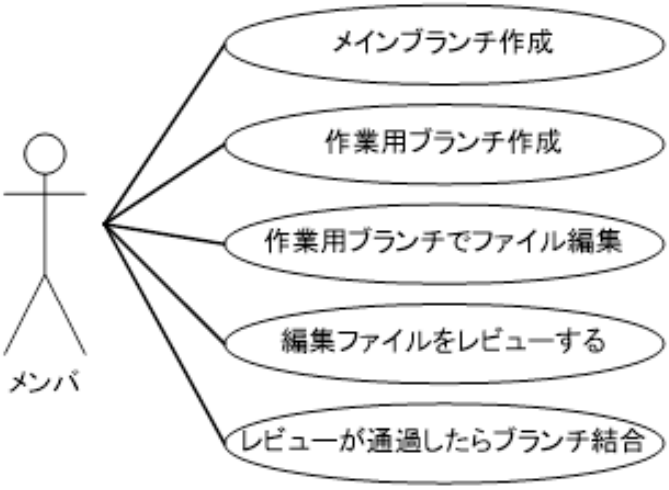


図 2.4　GitHub フローユースケース図　出典：[1]p18

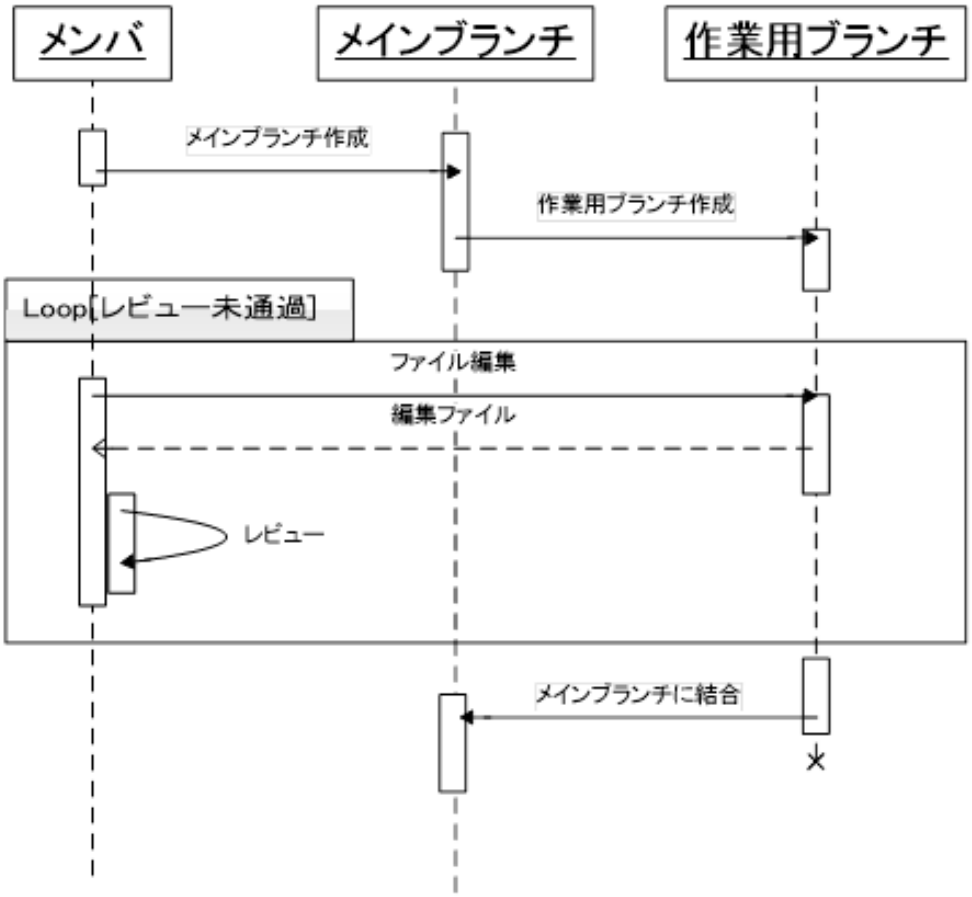


図 2.5　GitHub フローシーケンス図　出典：[1]p18

GitHub フローは GitHub 社が実践しているシンプルなワークフローである．基本的には特定の作業をするブランチを作成するだけなので，作業を始めてデプロイするまでの過程がとてもシンプルです．これはワークフローを実施するまでの学習コストを抑えられるという利点があります．さらに大きな利点として，シンプルであるからこと，多くの開発者がすばやく行うことを可能にします．そして，小さな変更などにも柔軟に対処できるようになります [9].

2.4.2　Git フロー

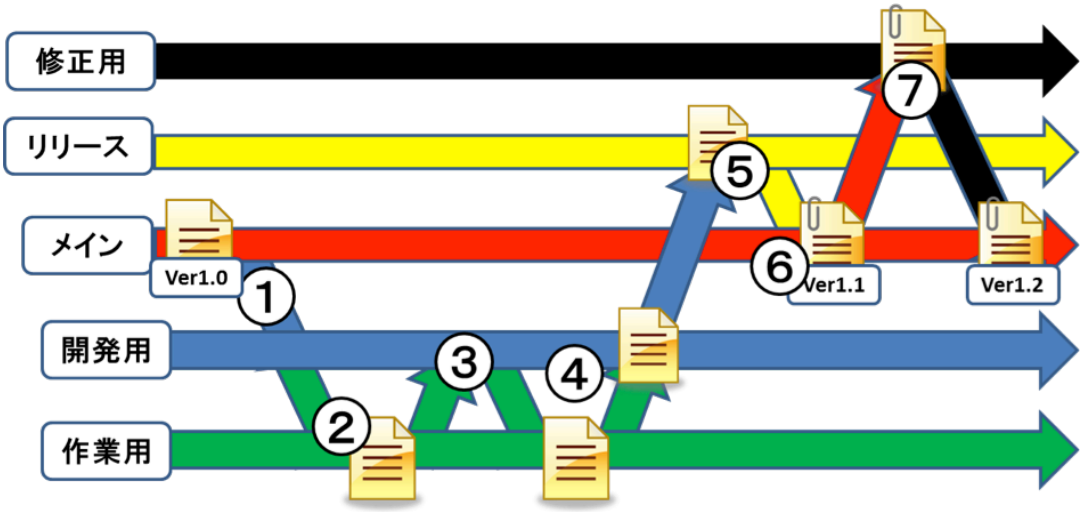


図 2.6　Git フロー図　出典：[1]p20

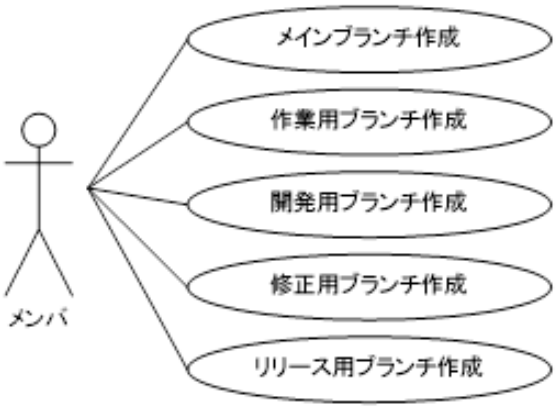


図 2.7　Git フローユースケース図　出典：[1]p20

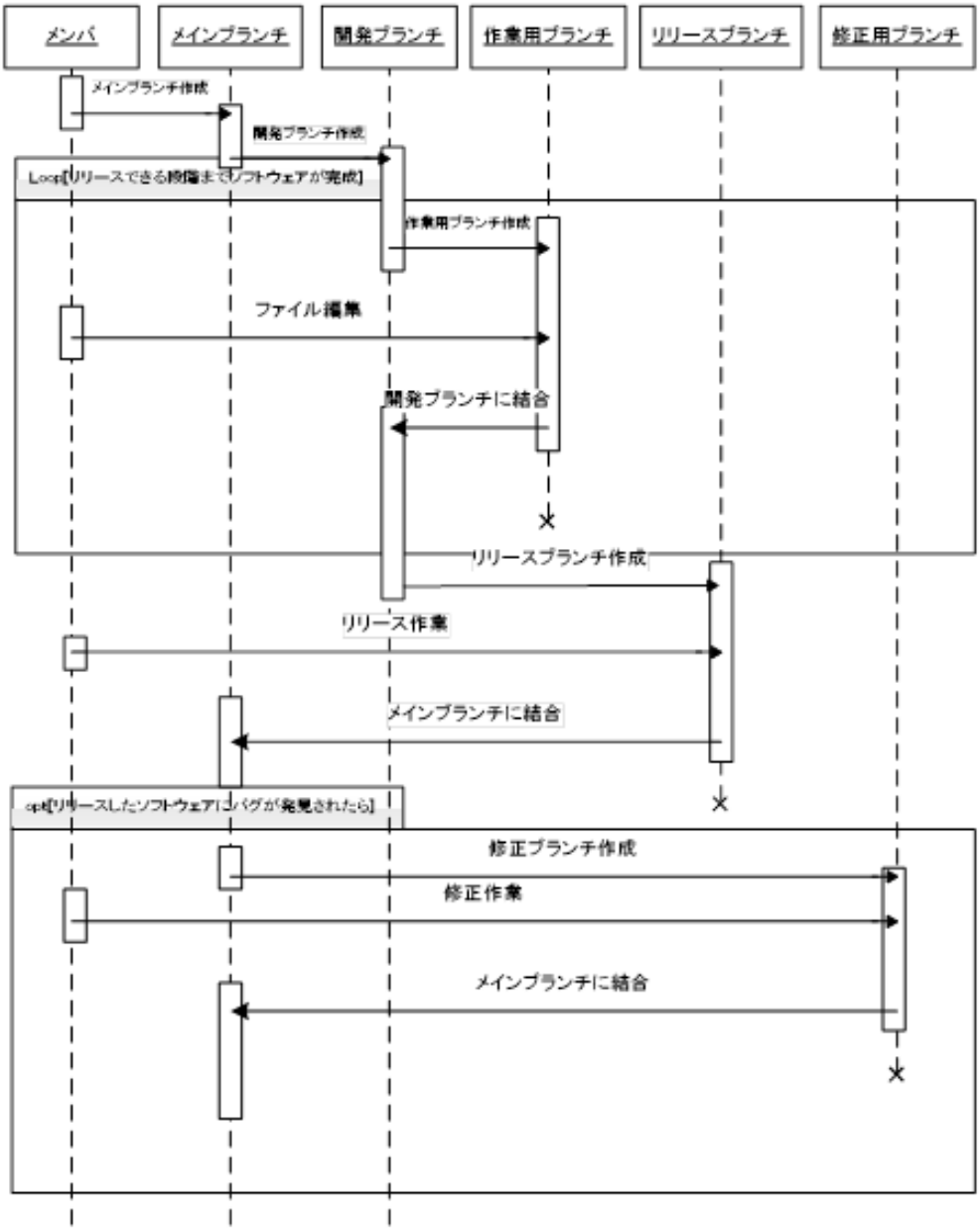


図 2.8　Git フローシーケンス図　出典：[1]p20

Git フローはリリース中心の開発スタイルである．この開発フローは，それぞれのブランチがコードの状態を表している．ソフトウェアのリリースを管理するリリースマネージャーなどが存在し，リリースを中心としたソフトウェア開発に向いている．しかし，覚えるブランチの状態が多く，開発フロー全体を事前に学習する必要がある．そのため，git フローなどのツールのサポートを受けることにより，強制的にフローからはずれない工夫が必要である [9].

2.4.3　GitLab フロー

GitLab フローは，GitHub フローをベースにしている．

特徴は，branch に stable branch を採用している点である．GitLab branch とは，安定状態の branch を作ることにより，バグが新たに入り込むリスクを減らす効果がある．GitLab branch は，できるだけ最新の master ブランチから分岐することで，バグを新たに入り込むリスクを減らせる [4].

2.4.4　日本 CAW フロー

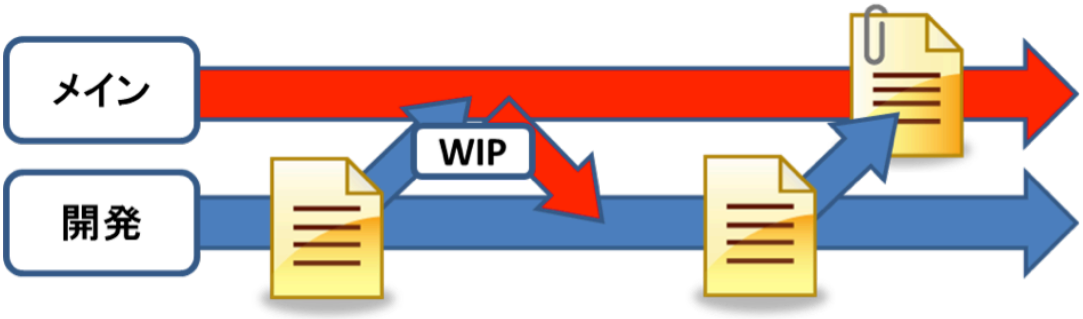


図 2.9　日本 CAW フロー図　出典：[1]p23

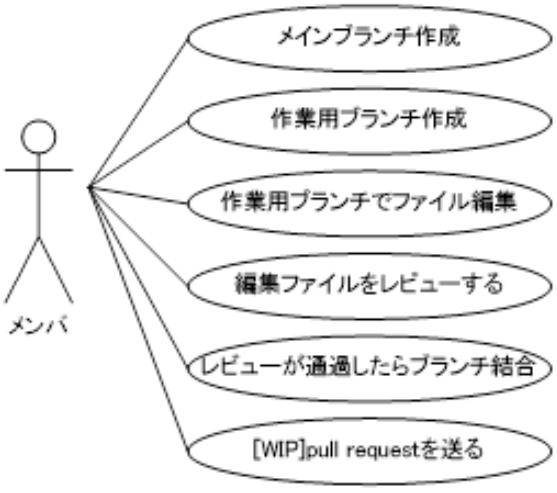


図 2.10　日本 CAW フローユースケース図　出典：[1]p24

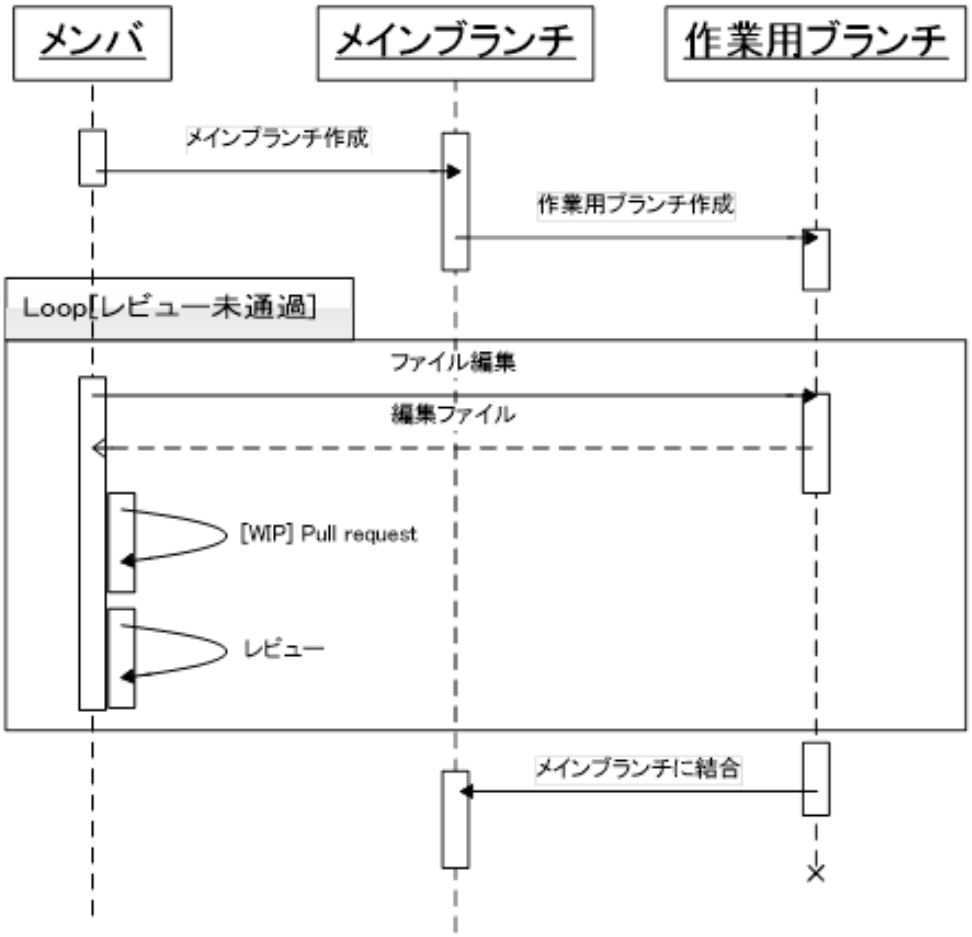


図 2.11　日本 CAW フローシーケンス図　出典：[1]p24

日本 CAW フローは，日本 CAW 株式会社が採用しているフローである．このフローは，GitHub フローをベースにして Pull Request を活用したスタイルを採用している．特徴は，WIP PR を採用している点である．WIP PR(Work In Progress Pull Request) とは，実装の仕方や，コードの設計など，コードに紐づく議論をしやすくするために用いられる．Pull Request 作成時に頭に [WiP] を入れる．[WIP] がついた Pull Request はマージされず，議論や確認のために参照される．作成者は議論や確認が済んだら Close する，といったフローである [10].

2.4.5　LINE フロー

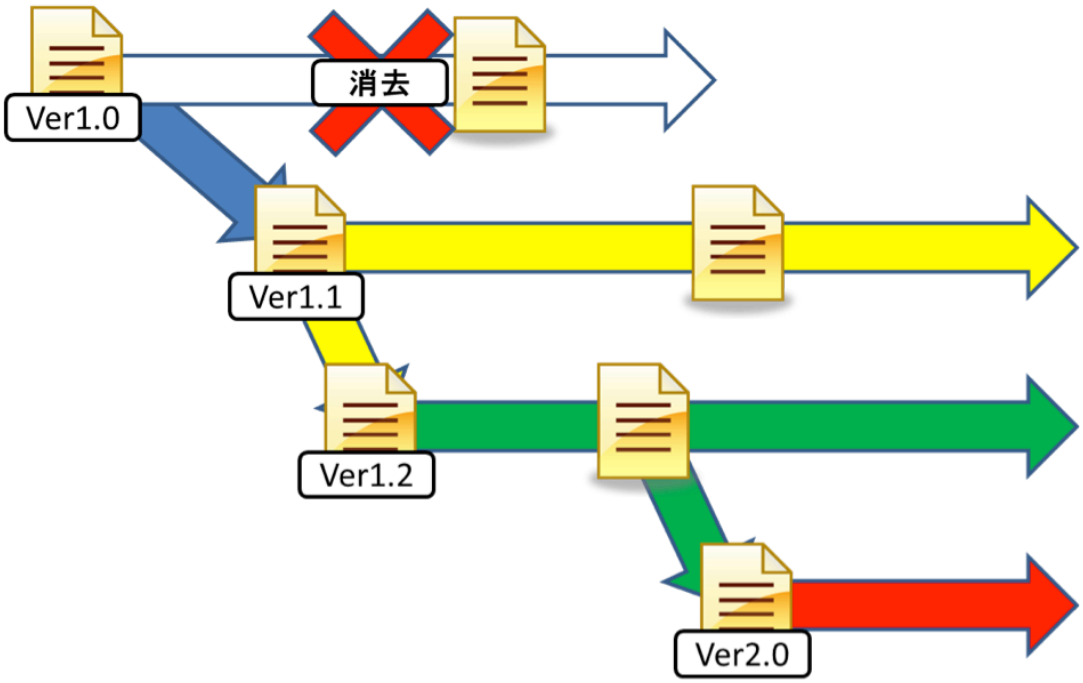


図 2.12　LINE フロー図　出典：[1]p31

図 2.13　LINE フローユースケース図　出典：[1]p32

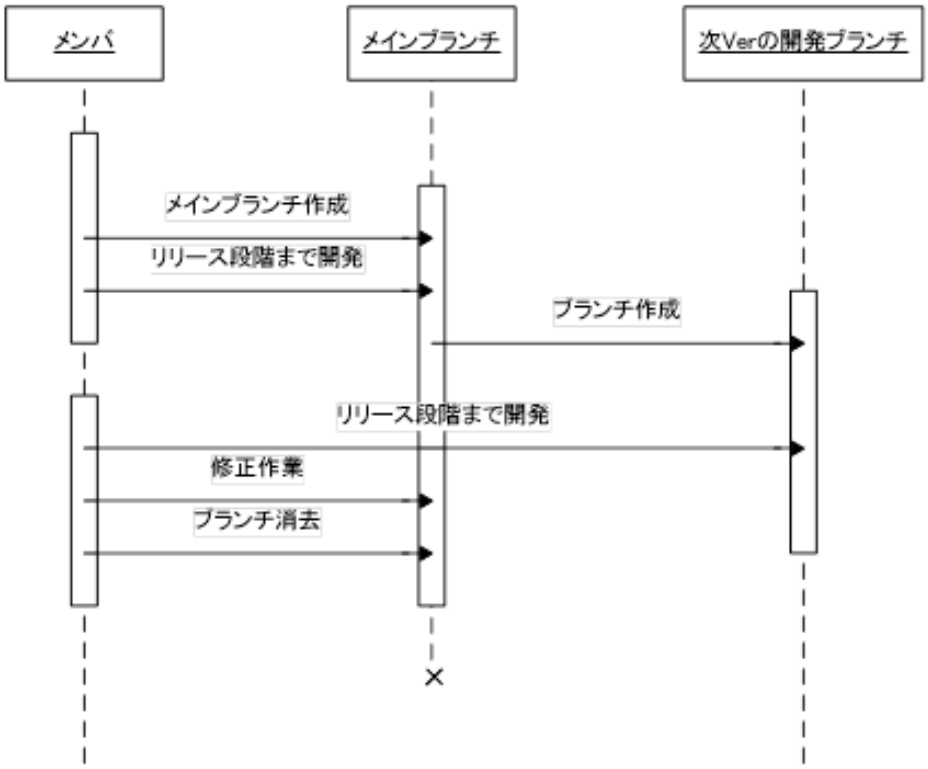


図 2.14　LINE フローシーケンス図　出典：[1]p32

LINE フローは Pull Request などを活用して行われる開発フローである．常に deploy 可能なバージョン毎の master を持っているよいう形で運用している．それぞれがリリース可能なブランチで，開発状況によっては下位のバージョンのコミットが上位のバージョンのコミットより新しい場合がある．LINE の IOS アプリではバージョン毎に提供する機能や修正を決めていて，QA 対象の切り分けがしやすいなどのメリットもあり，1 つの master に commit を繋げいく形では運用していない．現在開発中の最も下位のバージョンの修正がリリースされたら，そのブランチを上位のブランチにマージして下位のブランチを消していく流れで行われる [11].

2.5　GitHub を用いた開発フローの選択基準

それぞれ異なったリスクがあるため，最適な開発フローを選択する基準をリスクの面から分類された．研究の結果が表 2.1 である．プロジェクトの特徴から最適な開発フローが選択できるようになった.

表 2.1 プロジェクトと開発フロー出典：[1]p62

プロジェクトの特徴	開発フローの特徴	該当する開発フロー
メンバのスキルが高い 大規模なプロジェクト	フローが自動化されている	フィヨルドフロー イストフロー
参加メンバが多い	複数のリポジトリを使用する	Aming フロー サイボウズフロー 矢吹研フロー① 矢吹研フロー②
アジャイル型のソフトウェア開発	アジャイル開発のような 流れが可能	Git フロー キャスレーフロー
GitHub を今まで導入した経験が無い プロジェクト	使用するブランチが少ない	GitHub フロー 日本 CAW フロー はてなブログフロー
ソフトウェアに何を盛り込むのかが明確	使用済みブランチを破棄	ラクスルフロー LINE フロー

しかし，メンバのスキルが高い，大規模なプロジェクト等，定性的な表現が多い．

そこで当研究では，選択する基準を定量的に分けられるようにすることを目指す．具体例を二つあげる．開発人数が 5 人の場合かつ，言語が Ruby の場合は，GitHub フローが最適である．開発人数が 15 人の場合かつ，言語が Java の場合は，Git フローが最適である，

そうすることで，既存の基準より，適切な開発フローを選択できるようになると考える．

第 3 章

目的

GitHub を用いたソフトウェア開発プロジェクトの性質において，適切な開発フローを選択できるようにするための基準を提供する．

第 4 章

手法

調査準備，調査対象，調査方法，分析準備，分析方法について記述する．

環境は，2 種類用いた．一つ目は，OS X Yosemite　バージョン 15.15.5 である．二つ目は，Ubuntu である．

4.1　調査準備

調査は，GitHubAPI を用いる．そのため，GitHubAPI の準備について，記述する．

ここで行う準備の環境は，Ubuntu を想定している．

curl のインストール

```
sudo apt-get install curl
```

ログイン情報入力を省略

```
echo 'ユーザ名:パスワード' > github.passwd
chmod 600 github.passwd
```

Python と HTTP アクセスのための requests のインストール

```
sudo apt-get install python python-setuptools
sudo easy_install requests
```

api.py のダウンロード

```
curl -s -u $(cat github.passwd)
https://raw.githubusercontent.com/taroyabuki/yabukilab/master/library/github/api.py
 > api.py
```

api.py について

```
#!/usr/bin/python
# coding: UTF-8
```

```
import sys, json, requests

#GitHub のログイン情報をファイルから取得する
#TODO:パスワードに「:」を使っているとダメ
tmp = open('github.passwd').readline().rstrip('\n').split(':');
username = tmp[0]
password = tmp[1]
#print >> sys.stderr, username,password

#API の URL はコマンドライン引数で与える
url = sys.argv[1]

count = 0
while (url is not None):
  print >> sys.stderr, url
  r = requests.get(url, auth=(username, password))
  print >> sys.stderr, r.headers['status'],
  items = r.json()['items'] if 'items' in r.json() else r.json()
  for item in items:
      count = count + 1
      print json.dumps(item)
  if (r.links.has_key('next')):
    url = r.links['next']['url']
  else:
    url = None
  print >> sys.stderr, count, 'items'
```

4.2　調査対象

調査対象プロジェクトと調査対象指標を記述する.

4.2.1　調査対象プロジェクト

調査するユーザ名とプロジェクト名を記載する.

表 4.1 調査対象プロジェクト

ユーザ名	リポジトリ名
zedapp	zed
LearnBoost	stylus
spine	spine
sirjuddington	SLADE
sass	sass
Polymer	polymer
play	play
ossec	ossec-hids
NTU-CCSP	ntu-ccsp.github.io
neovim	neovim
aol	moloch
rackerlabs	mimic
melonjs	melonJS
nasa	mct
waysome	libreset
knockout	knockout
KnpLabs	FriendlyContexts
freifunkMUC	freifunkmuc.github.io
admc	flex-pilot-x
dfm	emcee
adamhjk	dynect_rest
ashesi-SE	datasaver
pyca	cryptography
karpathy	convnetjs
tlycken	Contour.jl
ellisonleao	clumsy-bird
mozbrick	brick
tyoshii	bms
sampsyo	beets
openstf	adbkit
CyberAgent	android-gpuimage

4.2.2　調査対象指標

調査する指標は, Watch 数, Star 数, Fork 数, commit 数, branch 数, Release 数, 人数, 言語, 行数, ファイル数, バイト数, OpenIssues 数, ClosedIssues 数, Issues 数, OpenPull Request 数, ClosedPull Request 数, Pull Request 数, Label 数, OpenMileStone 数, ClosedMileStone 数, MileStone 数, Wiki 数, 言語, 日数, 開発フローである.

4.3　調査方法

4.3.1　Watch 数，Star 数，Fork 数，commit 数，branch 数，Release 数，人数の調査

調査は，手動で行う.

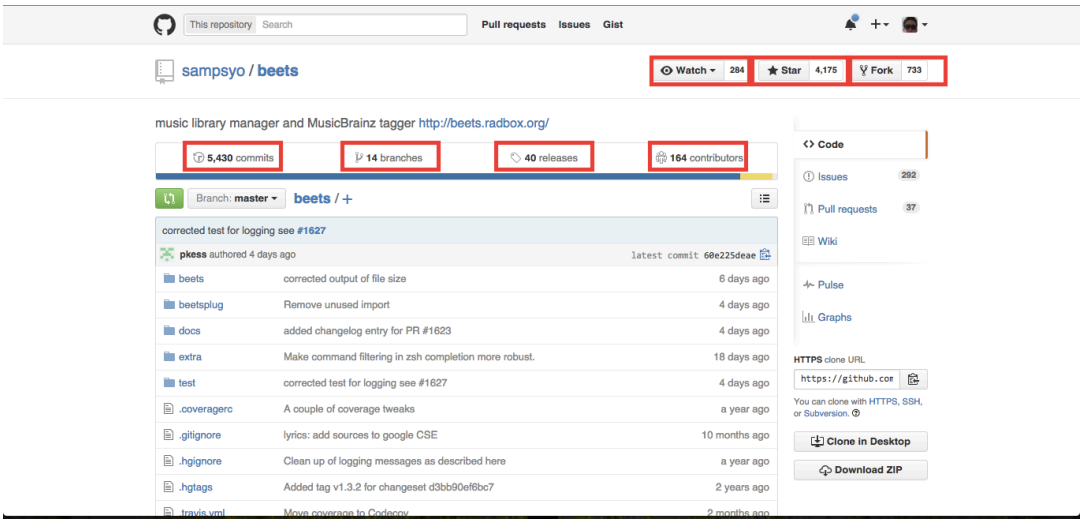


図 4.1　Watch 数，Star 数，Fork 数，commit 数，branch 数，Release 数，人数の調査

図 4.1 は Watch 数，Star 数，Fork 数，commit 数，branch 数，Release 数，人数の調査画面である．commits が commit 数，branches が branch 数，releases が Release 数，contributors が人数である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する.

4.3.2　言語の調査

調査は，手動で行う.

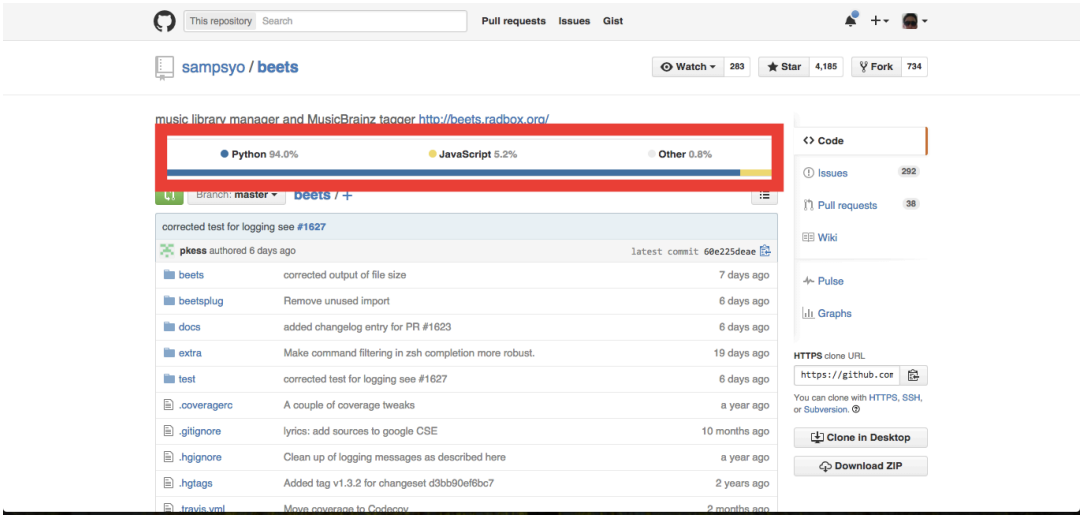


図 4.2　言語の調査

図 4.2 は言語の調査画面である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する．

4.3.3　OpenIssues 数，ClosedIssues 数，Issue 数の調査

調査は，手動で行う．

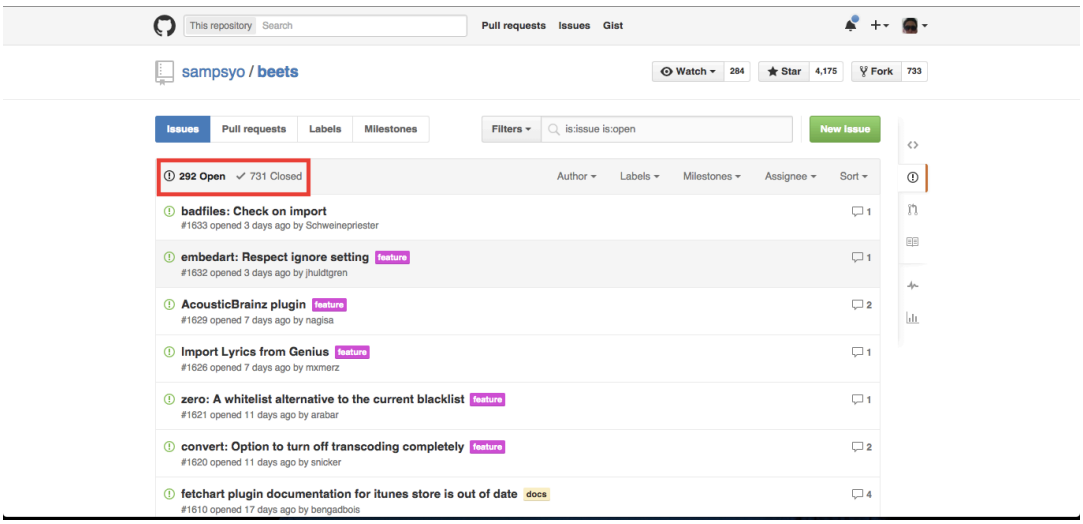


図 4.3　Issues 数の調査

図 4.3 は OpenIssues 数, ClosedIssues 数, Issue 数の調査画面である．Open が OpenIssues 数，Closed が ClosedIssues 数である．Open と Closed の合計が Issue 数である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する．

4.3.4　OpenPull Request 数，ClosedPull Request 数

調査は，手動で行う．

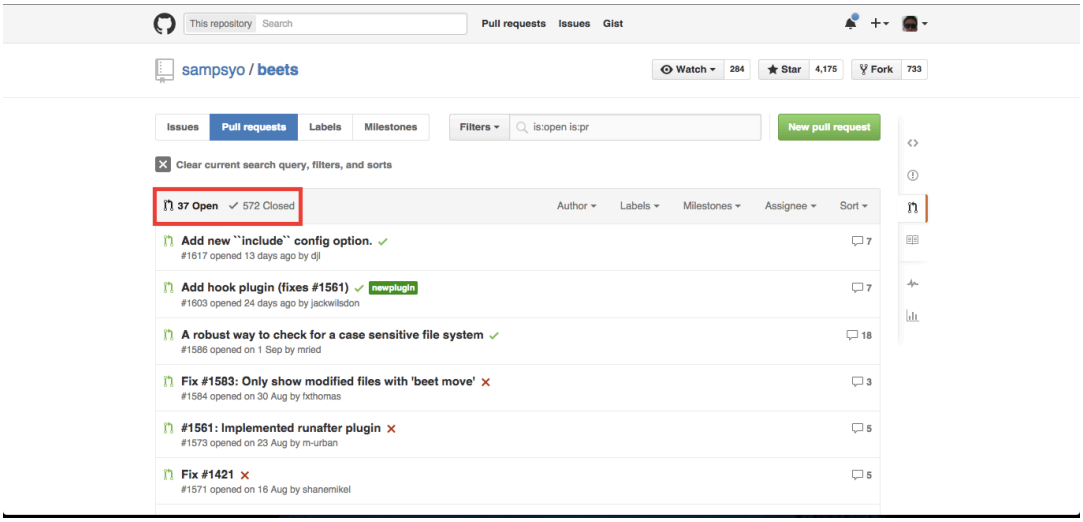


図 4.4　Pull Request 数の調査

図 4.4 は OpenPull Request 数，ClosedPull Request 数の調査画面である．Open が OpenPull Request 数，Closed が ClosedPull Request 数である．Open と Closed の合計が Pull Request 数である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する．

4.3.5　Label 数

調査は，手動で行う．

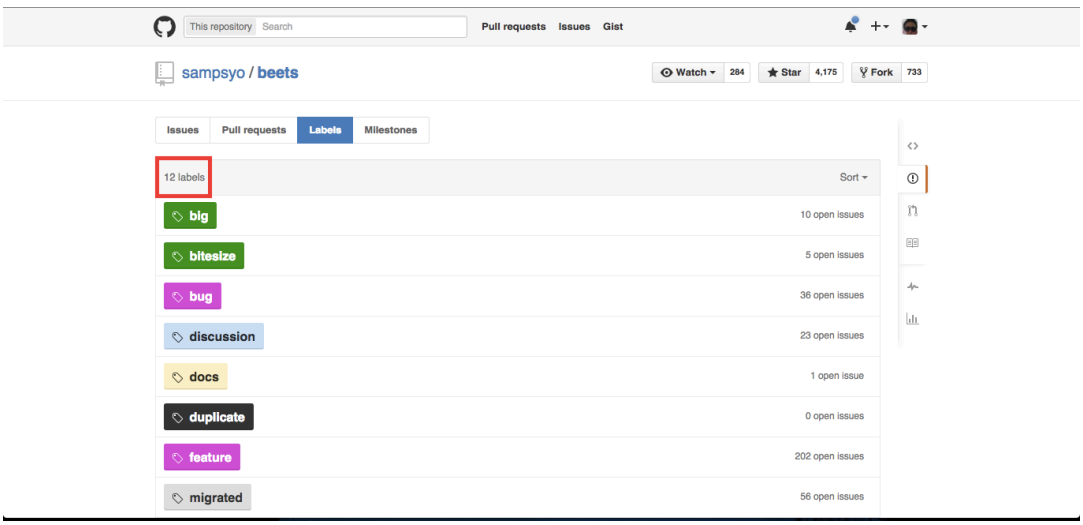


図 4.5　Label 数の調査

図 4.5 は Label 数の調査画面である．labels が Label 数である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する．

4.3.6　OpenMilestone 数，ClosedMilestone 数，Milestone 数

調査は，手動で行う．

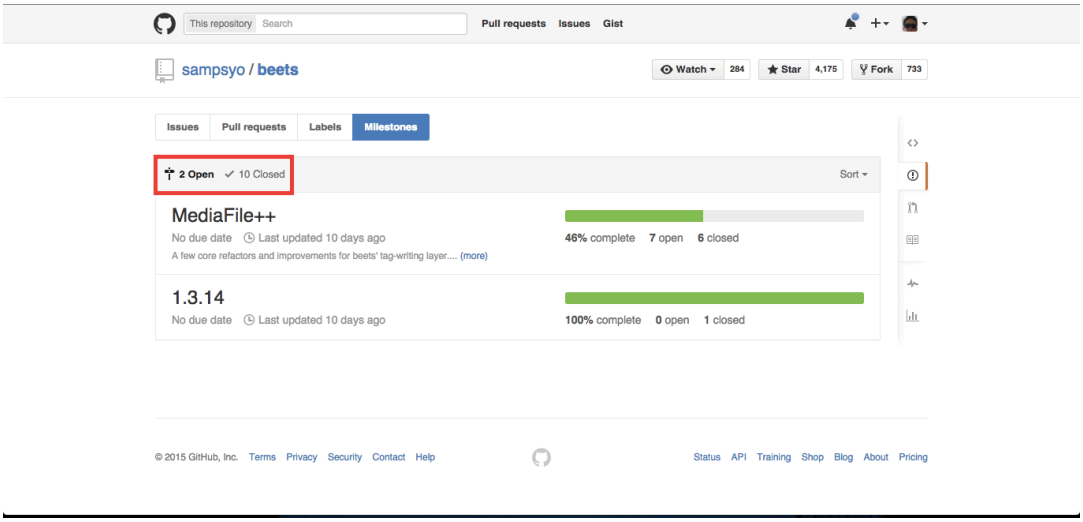


図 4.6　Milestone 数の調査

図 4.6 は OpenMilestone 数，ClosedMilestone 数，Milestone 数の調査画面である．Open が OpenMilestone 数，Closed が ClosedMilestone 数である．Open と Closed の合計が Milestone 数である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する．

4.3.7　Wiki 数

調査は，手動で行う．

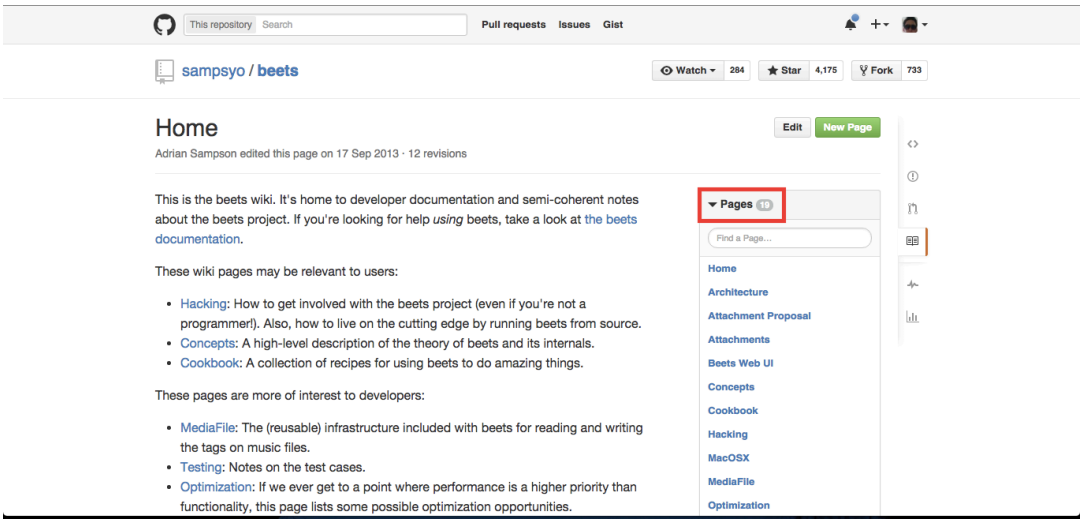


図 4.7　Wiki 数の調査

図 4.7 は Wiki 数の調査画面である．Pages の隣の数値が Wiki 数である．赤枠に囲まれている数値を取得する．所得した数値は，"データ一覧.csv"に保存する．

4.3.8　ファイル数，バイト数

指定のリポジトリを clone する．

```
git clone git@github.com:アカウント名/リポジトリ名
```

clone したリポジトリの情報を確認する．右クリックし，「情報をみる」を選択する．選択すると，バイト数とファイル数が表示される．表示された数値を，"データ一覧.csv"に保存する．

4.3.9　行数

指定のリポジトリを clone する．

```
git clone git@github.com:アカウント名/リポジトリ名
```

clone したリポジトリをターミナルで開く．ターミナルで，以下のコマンドをうつ．

```
grep -rI '' リポジトリ名 | wc -l
```

行数が出力される．
出力された行数を，"データ一覧.csv"に保存する．

4.3.10　日数

GitHub が提供している API を用いる．
最初の commit した日をプロジェクト開始日とする．最後の commit した日をプロジェクト終了日とする．これは，すでに終了しているプロジェクトや，commit が長期間されて

いないプロジェクトを考慮しているためである．
まず．API を用いて，すべての commit を取得する．

```
python api.py https://api.github.com/repos/ユーザ名/リポジトリ名/commits?per_page=100 > ユーザ名.リポジトリ名.-commits.txt
```

すべての commit から最初と最後の commit を特定し，日付を取得する．取得した数値を，"データ一覧.csv"に保存する．

4.3.11 一日あたりの行数

一日あたりの行数は，行数をプロジェクト経過日数で割る．

4.3.12 1 人日あたりの行数

1 人日あたりの行数は，行数をプロジェクト経過日数と人数で割る．

4.3.13 一日あたりの commit 数

一日あたりの commit 数は，commit 数をプロジェクト経過日数で割る．

4.3.14 1 人日あたりの commit 数

1 人日あたりの commit 数は，commit 数をプロジェクト経過日数と人数で割る．

4.3.15 開発フロー

開発フローの特徴とプロジェクトの特徴を照らし合わせる．

GitHub フロー

master branch から記述的な名前の branch がある場合は，GitHub フローである．

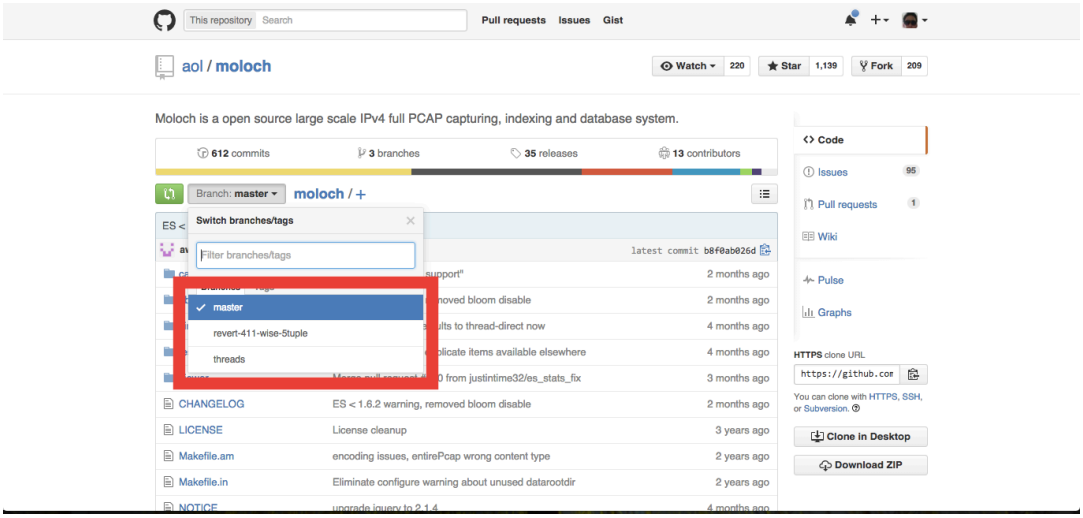


図 4.8　GitHub フローの調査

Git フロー

develop branch と release branch がある場合は，Git フローである．

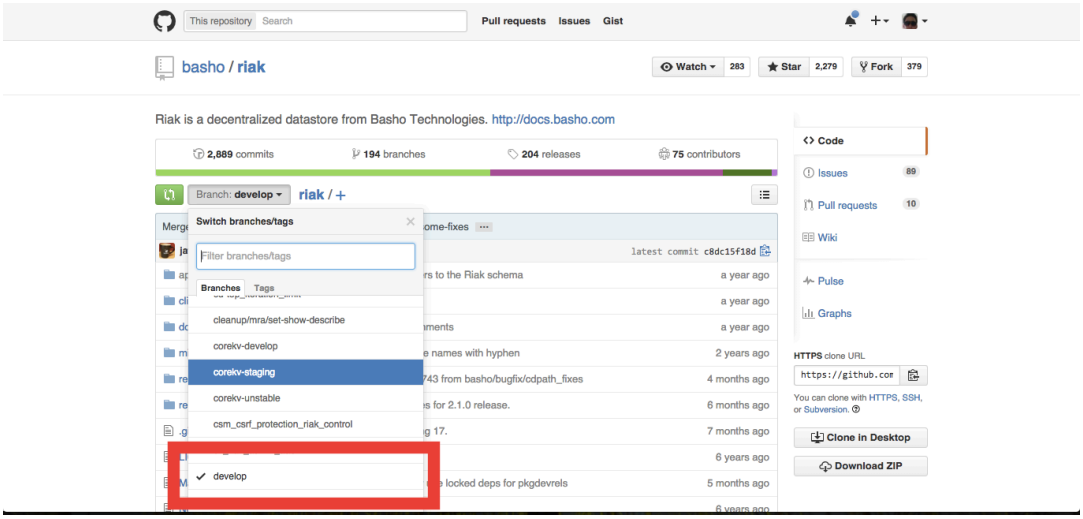


図 4.9　Git フローの調査

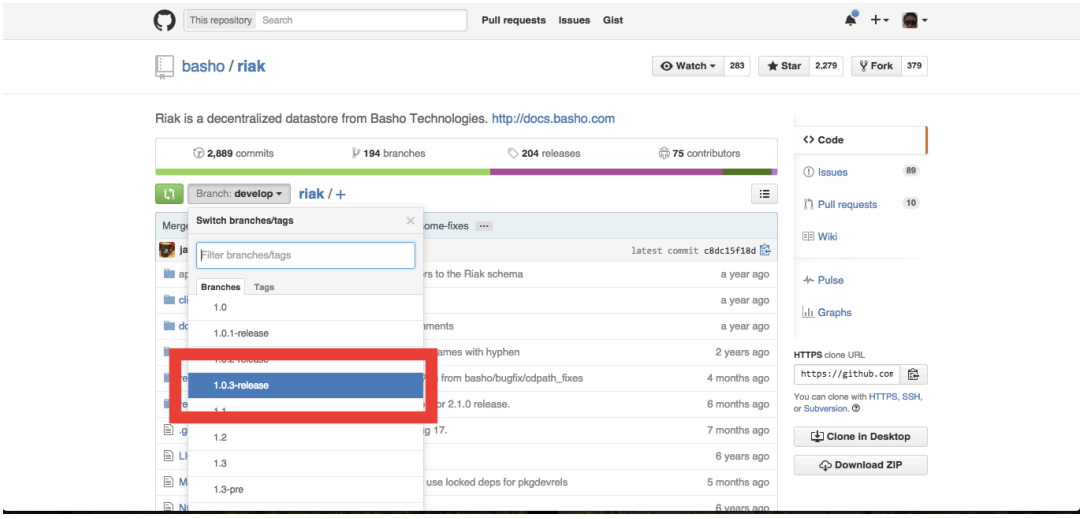


図 4.10　Git フローの調査

LINE フロー

バージョンごとに branch が作られている場合は，LINE フローである．

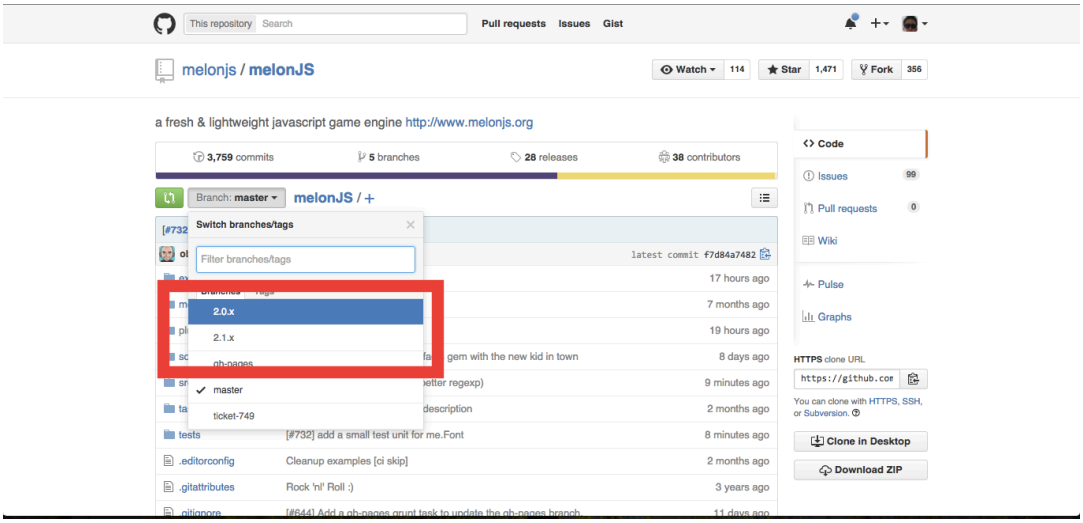


図 4.11　LINE フローの調査

日本 CAW フロー

Pull Request に [WIP] がある場合は，日本 CAW フローである．

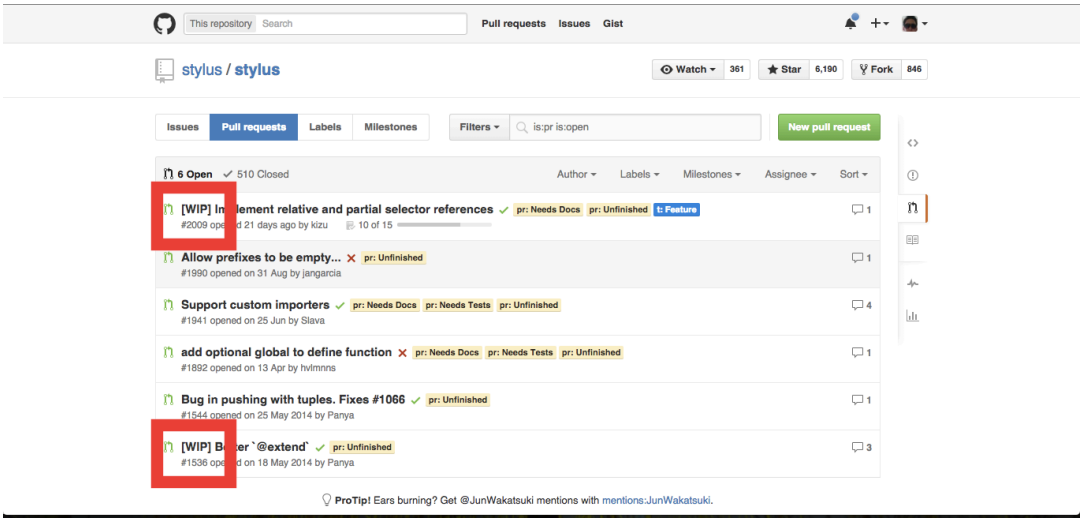


図 4.12　日本 CAW フローの調査

GitLab フロー

branch に stable がある場合は，GitLab フローである．

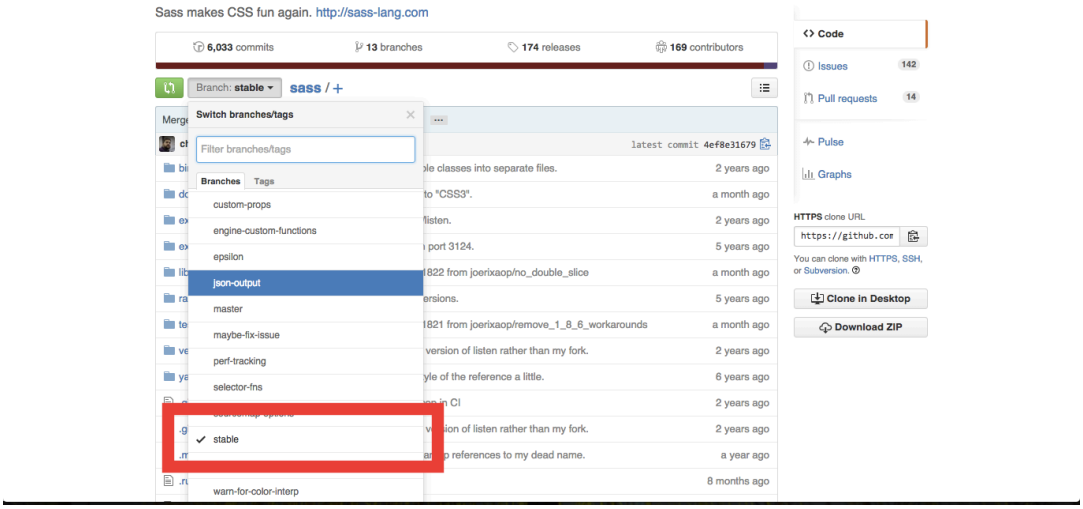


図 4.13　GitLab フローの調査

4.3.16　コマンドの説明

調査に用いたコマンドの説明を，以下に記述する．

requests

人が使いやすいように設計されていて，Python で書かれている Apache2 Licensed ベースの HTTP ライブラリである．

sudo

UNIX および，UNIX 系オペレーティングシステムのプログラムの 1 つで，ユーザが別のユーザの権限レベルでプログラムを実行するためのコマンドである．

echo

文字列を出力するコマンド

apt-get

パッケージを取得してインストール/アップデートする Debian プロジェクトが開発したパッケージ管理システムである．

chmod

UNIX および，UNIX 系オペレーティングシステムにおけるシェルコマンドの一種である．ファイルやディレクトリのファイルモードを変更するのに使われる．

grep

ファイルの中の文字列を検索する．

-r

各ディレクトリ下のすべてのファイルを再帰的に読み取る．

-I

バイナリファイルを無視する．

|

コマンドの標準出力を次のコマンドに渡す処理を行う．

wc

テキスト・ファイルの行数，単語数，バイト数を表示する．

-l

行数のみ集計し表示する．

4.4 分析ツール

分析に使うツールは，Excel と R である．Excel は，集めたデータを分析用のデータに分別したり，保存しておくために使う．R は，集めたデータを分析するために使う．

R とは，統計計算をしたり，その結果をグラフにまとめるための言語・環境で，ボランティアによって作られた，S-PLUS の主要な関数をほぼ備えた GNU 版（コピーフリー版）

です．1991 年，ニュージーランド，オークランド大学統計学科の 2 人の講師，Ross Ihaka と Robert Gentlemant によって開発が始まり，現在も開発が進められています [12].

4.4.1　R パッケージのインストール

決定木分析を，R で行う方法を記述する．
3 種類のパッケージをインストールする

```
options(repos=c(CRAN="http://cran.ism.ac.jp"))
```

"rattle"をインストールする．"rattle"とは，

```
install.packages("rattle")
```

成功すると以下の画面になる．

```
> install.packages("rattle")
Installing package into 'C:/Users/[redacted]/Documents/R/win-library/3.1'
(as 'lib' is unspecified)
 URL 'http://cran.ism.ac.jp/bin/windows/contrib/3.1/rattle_4.0.5.zip' を試しています
Content type 'application/zip' length unknown
 開かれた URL
downloaded 3.6 Mb

 パッケージ 'rattle' は無事に展開され、MD5 サムもチェックされました
```

図 4.14　rattle インストール成功画面

"RColorBrewer"をインストールする．"RColorBrewer"とは，様々な美しいカラーパレットが含まれているパッケージである．

```
install.packages("RColorBrewer")
```

成功すると以下の画面になる．

```
> install.packages("RColorBrewer")
Installing package into 'C:/Users/[redacted]/Documents/R/win-library/3.1'
(as 'lib' is unspecified)
 URL 'http://cran.ism.ac.jp/bin/windows/contrib/3.1/RColorBrewer_1.1-2.zip' を試しています
Content type 'application/zip' length unknown
 開かれた URL
downloaded 26 Kb

 パッケージ 'RColorBrewer' は無事に展開され、MD5 サムもチェックされました
```

図 4.15　RColorBrewer インストール成功画面

"rpart.plot"をインストールする．"rpart.plot"とは，視覚的に表示するためのパッケージである．

```
install.packages("rpart.plot")
```

成功すると以下の画面になる．

```
> install.packages("rpart.plot")
Installing package into 'C:/Users/[redacted]/Documents/R/win-library/3.1'
(as 'lib' is unspecified)
 URL 'http://cran.ism.ac.jp/bin/windows/contrib/3.1/rpart.plot_1.5.3.zip' を試しています
Content type 'application/zip' length unknown
 開かれた URL
downloaded 510 Kb

パッケージ 'rpart.plot' は無事に展開され、MD5 サムもチェックされました
```

図 4.16　rpartPlot インストール成功画面

4.5　分析データ準備

4.5.1　全てのデータをまとめる

Excel の csv ファイルに調査した全てのデータを保存する．実際に保存したデータが図 4.17 である．

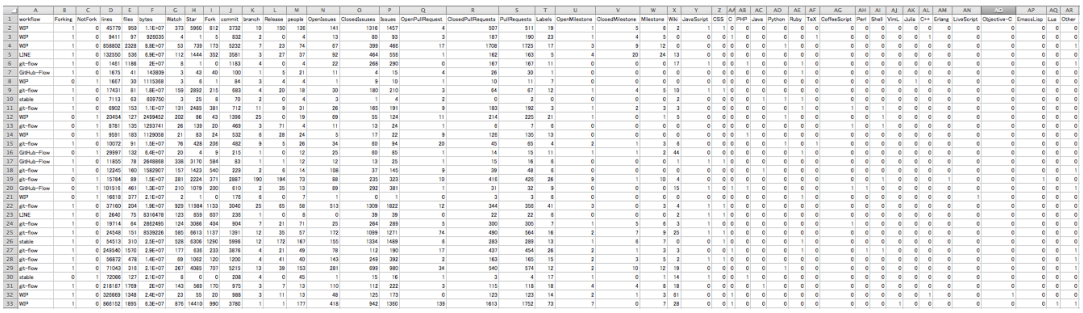


図 4.17　すべてのデータ一覧

Y 列からは，言語になっている．その言語があった場合は 1，ない場合は 0 を入れている．

4.5.2　全ての 2 つにわける

データをランダムに並び替え，2 つにわける．ランダムに分ける方法は，Excel の RAND 関数を使用する．わけられたデータで，それぞれ決定木分析を行う．

分けられたデータが図 4.18 と図 4.19 である．

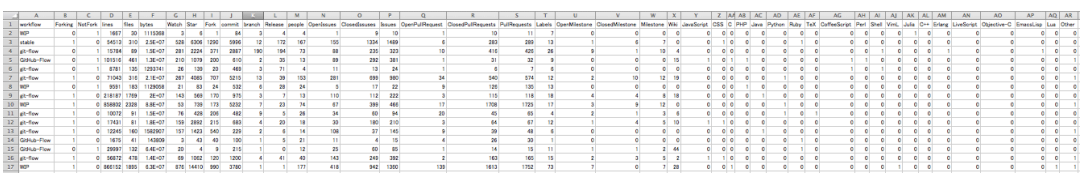
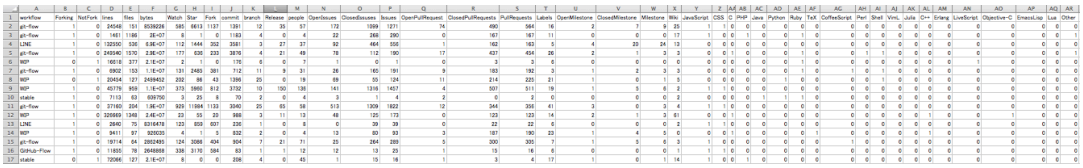


図 4.18　半分のデータ-1



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR
1	workflow	Forking	NotFork	lines	files	bytes	Watch	Star	Fork	commit	branch	Release	people	OpenIssues	ClosedIssues	Issues	OpenPullRequest	ClosedPullRequests	PullRequests	Labels	OpenMilestone	ClosedMilestone	Milestone	Wiki	JavaScript	CSS	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Other
2	git-flow	1	0	24548	151	6539226	585	6613	1137	1391	12	35	57	172	1099	1271	74	490	564	16	2	7	9	25	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	git-flow	1	0	1481	1186	3E+07	8	1	0	1182	4	0	4	22	266	290	0	167	167	11	0	0	0	17	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	LINE	1	0	132550	536	6.9E+07	112	1444	352	3581	3	27	37	92	464	556	1	162	163	5	4	20	24	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	git-flow	1	0	240340	1570	2.9E+07	177	630	233	3876	4	21	49	78	112	190	17	437	454	26	2	1	3	3	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1
6	WIP	0	1	16018	377	2.1E+07	2	1	0	175	6	0	7	1	0	1	0	3	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
7	git-flow	1	0	6902	153	1.1E+07	131	2483	381	712	11	9	31	26	165	191	9	183	192	3	1	2	3	3	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
8	WIP	0	1	20454	127	2409452	232	56	43	1396	25	0	19	69	55	124	11	214	225	21	1	0	1	3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	WIP	1	0	45179	958	1.1E+07	373	1668	812	3732	19	150	138	141	1316	1457	4	501	511	19	1	5	6	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
10	stable	1	0	7113	63	609790	3	25	8	70	2	0	4	3	1	4	2	0	2	0	0	0	0	2	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	git-flow	1	0	37160	204	1.9E+07	929	11994	1133	3040	25	69	58	513	1309	1822	12	344	356	41	3	0	3	4	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	WIP	1	0	326869	1348	2.4E+07	23	55	20	988	3	11	13	48	125	173	0	123	123	14	2	1	3	81	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
13	LINE	1	0	2640	75	6316478	123	859	607	238	1	0	8	0	39	39	0	22	22	6	0	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	WIP	1	0	3411	57	320535	4	1	5	532	2	0	4	13	80	93	3	187	190	23	1	1	4	5	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
15	git-flow	1	0	19114	64	2892495	124	3086	404	904	7	21	71	25	264	289	5	300	305	7	1	5	6	3	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
16	Github-Flow	1	0	11855	78	2648868	338	3170	584	83	1	1	12	12	13	25	1	15	16	6	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	stable	0	1	72096	127	2.1E+07	8	0	0	286	4	0	43	1	15	16	1	3	4	17	1	0	1	14	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 4.19　半分のデータ-2

4.5.3　コントロールできる要因かつ，成功プロジェクトの場合

プロジェクトが適切な開発フローを選択していることを実証する．プロジェクトが成功している場合，適切な開発フローを選択していることにする．プロジェクトが成功している場合の定義は，リリース数が 1 以上の時である．

また，選択基準を，プロジェクトが始められる時に使えるように，コントロールできる要因のみにする．コントロールできる要因とは，行数，プロジェクト経過日数，一日あたりの行数，一人日あたりの行数人数，言語である．

プロジェクトが成功している場合のデータをまとめたデータが図 4.20 である．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
GitHub フロー	11359	597	67	2632298	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	66245519	0.756	146	193	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1
日本CAW フロー	1157	472	9	113258	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
GitHub フロー	1170	1781	18	128651	0.03	0.66	22	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	9568	1882	76	17264851	0.18	5.1	28	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	217687	1220	2401	16870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	6398	1518	152	1102780	0.13	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24265378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0
Git フロー	23618	485	226	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	250803	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1
日本CAW フロー	863309	806	2424	8885272	1.2	1100	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LINE フロー	161332	1640	600	7885044	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	622	459	13359619	11	160	14	1	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	24372	1933	150	8878431	0.21	13	61	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10466517	0.194	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	15297	2210	80	14745686	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0

図 4.20　コントロールできる要因かつ，成功プロジェクト

4.6　階層的クラスター分析方法

"データ一覧.csv"のデータを読み込み，データセット"myData"に保存する．1 行目は，行ラベルということにして，データを読み込む．

```
myData <- read.csv("データ一覧.csv", row.names=1)
```

標準化してから距離を求める．標準化とは，各列の平均を 0，分散を 1 にすることである．

```
(myDistanceMatrix <- dist(scale(myData)))
```

階層的クラスター分析を実行する．

```
myClusters <- hclust(myDistanceMatrix)
```

分析結果を表示する

```
plot(myClusters, hang = -1)
```

4.7　決定木分析方法

"データ一覧.csv"のデータを読み込み，データセット"myData"に保存する．

```
myData <- read.csv("データ一覧.csv")
```

データセット"myData"を読み込む．

```
head(myData)
```

library の"mypart"と"maptools"を読み込む

```
library(mypart)
library(maptools)
```

"myData"のデータをデータセット"myTree"に保存する

```
myTree <- rpart(work フロー~ .,data = myData)
myTree
```

決定木を描画する．

```
fancyRpartPlot(myTree)
```

成功すると以下の画面になる．

```
> myData <- read.csv("決定木1.csv")
> head(myData)
      workflow   行数 プロジェクト経過日数 ファイル数 バイト数 Fork数 Commit数 Watch数 Star数 X1人日あたりのCommit数 X1日あたりのCommit数
1 GitHub フロー   1170                1781       18   128651     40      100       3     44                0.0026               0.0562
2 GitHub フロー  11359                 597       67  2632298    633       86     359   3439                0.0096               0.1400
3 GitHub フロー 101062                 622      459 13359619    213      614     225   1153                0.0710               0.9870
4 GitHub フロー  29697                 410      142 64966290     13      292      19      6                0.0400               0.7120
5    Git フロー   8276                 792      134  1285551     24      469      28    160                0.1000               0.5920
6    Git フロー  11740                1099      167  1551457    637      241     169   1952                0.0120               0.2190
  X1人日あたりの行数 X1日あたりの行数 branch数 release数 人数 OpenIssues ClosedIssuses Issues OpenPullRequest数 ClosedPullRequest数 PullRequest数
1               0.03               0.66         1          5    22           10             5     15                  4                  26            30
2               1.30              19.00         1          1    15           18            14     32                  3                  17            20
3              11.00             160.00         3         35    14           97           294    391                  1                  32            33
4               4.00              72.00         2          0    18           23            64     87                  0                  29            29
5               2.00              10.00         2         71     5           12            16     28                  1                   6             7
6               0.58              11.00         2          6    19          123            37    160                 12                  43            55
  Label数 OpenMilestone数 ClosedMilestone数 Milestone数 Wiki数 HTML JavaScript CSS Makefile C PHP Java Python Ruby TeX CoffeeScript Perl Shell
1       1               0                 0           0       0    0          0   0        0 0   0    0      0    1   0            0    0     0
2       6               0                 0           0       1    1          1   1        0 0   0    0      0    0   0            0    0     0
3       9               0                 0           0      15    1          1   0        0 1   1    0      0    0   0            0    1     0
4      11               1                 1           2      44    1          1   1        0 0   0    0      0    0   0            0    0     0
5       6               0                 0           0       0    0          0   0        0 0   0    0      0    0   0            1    0     1
6       6               1                 0           1       0    0          0   0        0 1   0    1      1    0   0            0    0     0
  VimL Julia C.. Erlang LiveScript Objective.C EmacsLisp Lua Cmake Inno.Setup IDL NSIS Other
1    0     0   0      0          0           0         0   0     0          0   0    0     0
2    0     0   0      0          0           0         0   0     0          0   0    0     0
3    0     0   0      0          0           0         0   0     0          0   0    0     1
4    0     0   0      0          0           0         0   0     0          0   0    0     0
5    0     0   0      0          0           0         0   0     0          0   0    0     0
6    0     0   0      0          0           0         0   0     0          0   0    0     0
>
> library(rpart)
> myTree <- rpart(workflow ~ ., data = myData, control = rpart.control(minsplit = 5))
>
> #木の描画
> library(rattle)
> library(rpart.plot)
> fancyRpartPlot(myTree)
```

図 4.21　分析成功画面

4.8　決定木性能測定方法

決定木性能測定方法を記述する．

4.8.1　訓練データとテストデータの準備

データをランダムに 2 種類に分ける．一方は，16 件の訓練データ，もう一方は，8 件のテストデータに分ける．

ランダムに分ける方法は，Excel の RAND 関数を使用する．

16 件の訓練データで，決定木を作る．8 件のテストデータで，精度と再現率を求める．

これを 10 回行い，10 通りのデータを用意する．

4.8.2　平均と信頼区間を求める

平均は，Excel の AVERAGE 関数を用いる．標準偏差は，STDEVP 関数を用いる．信頼区間は，CONFIDENCE 関数を用いる．

第 5 章

結果

5.1 階層的クラスター分析結果

2 種類のデータを用いて，R で決定決定木分析を行った．すべてのデータで行った場合，コントロールできる要因のみかつ，成功しているプロジェクトの場合である．

次に，それぞれの結果を記述する．

5.1.1 すべてのデータで行った場合

図 5.1 は，すべてのデータで分析を行った結果である．

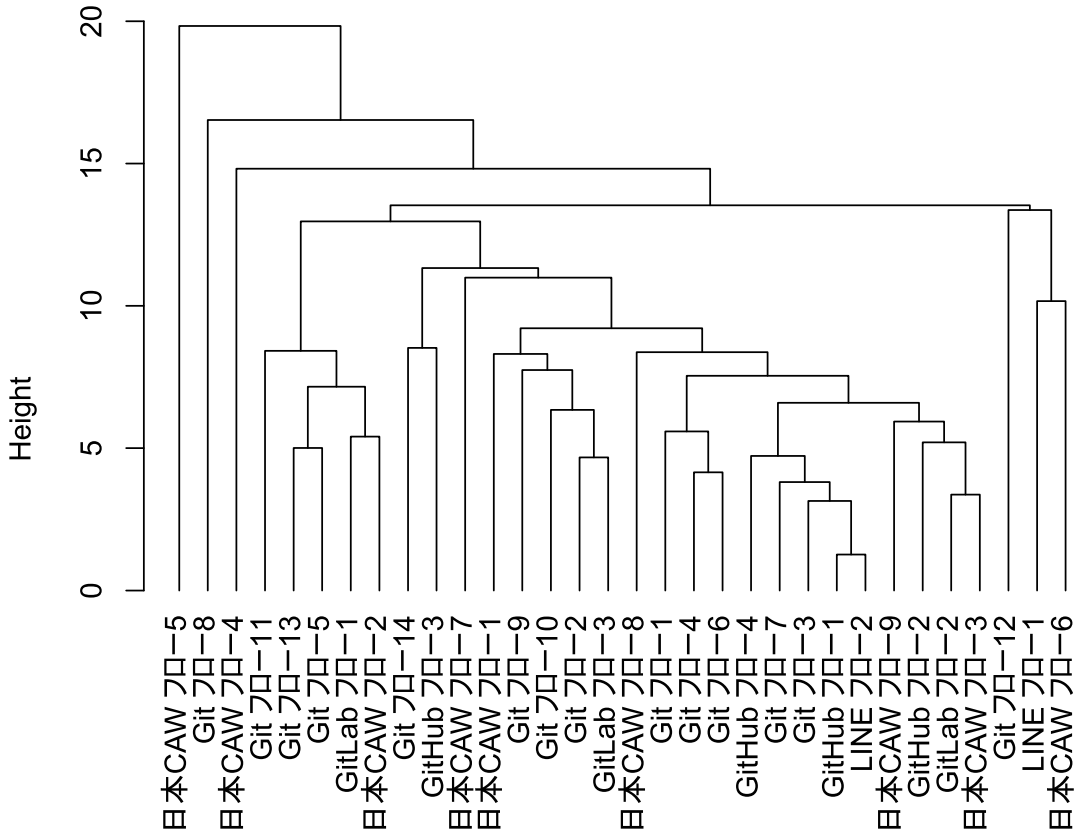


図 5.1　全てのデータで作った場合

5.1.2　コントロールできる要因のみかつ，成功しているプロジェクトの場合

図 5.2 は図 4.20 を用いて分析を行った結果である．

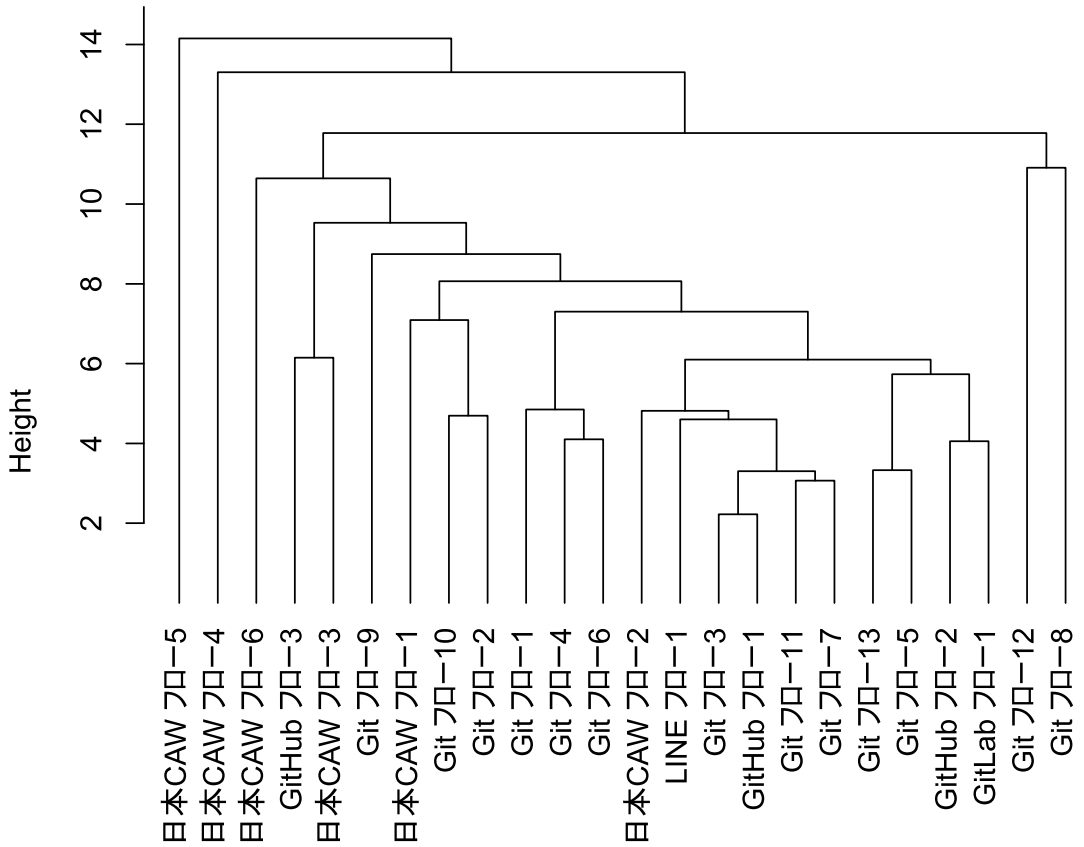


図 5.2　コントロールできる要因かつ，成功しているプロジェクトの場合

5.2　決定木分析結果

3 種類のデータを用いて，R で決定決定木分析を行った．すべてのデータで行った場合，データをランダムに並び替え，2 つに分けた場合，コントロールできる要因のみかつ，成功しているプロジェクトの場合である．

次に，それぞれの結果を記述する．

5.2.1　すべてのデータで行った場合

図 5.3 は，すべてのデータで分析を行った結果である．

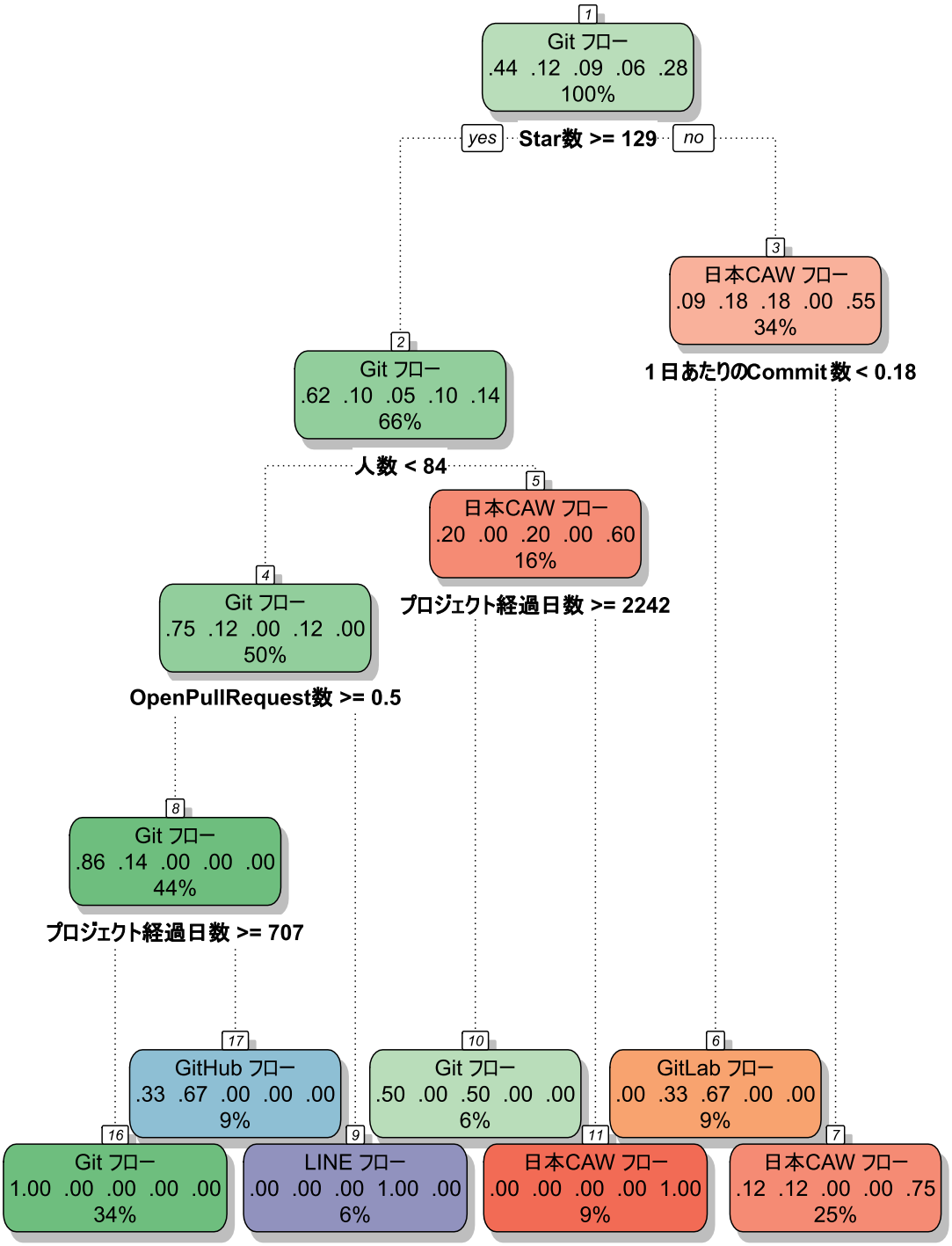


図 5.3　全てのデータで作った決定木

すべてのデータで分析した場合，Star 数，人数，Open Pull Request 数，Watch 数，ファイル数，branch 数，JavaScript で別れた．

5.2.2　データをランダムに並び替え，2 つに分けた場合

図 5.4 は，図 4.18 を用いて分析を行った結果である．図 5.5 は，図 4.19 を用いて分析を行った結果である．

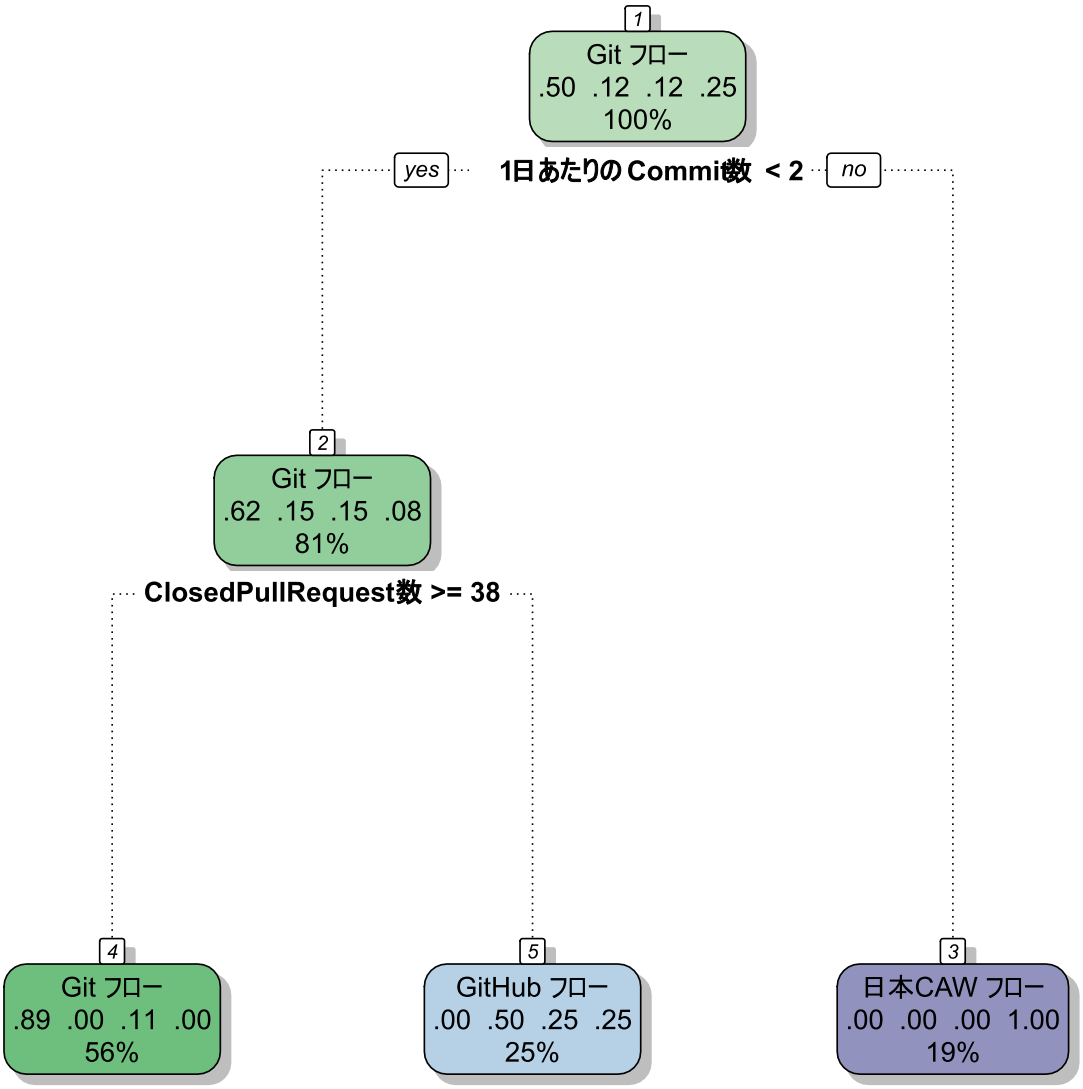

1
Git フロー
.50 .12 .12 .25
100%
yes
1日あたりの Commit数 < 2
no
2
Git フロー
.62 .15 .15 .08
81%
ClosedPullRequest数 >= 38
4
Git フロー
.89 .00 .11 .00
56%
5
GitHub フロー
.00 .50 .25 .25
25%
3
日本CAW フロー
.00 .00 .00 1.00
19%

図 5.4　決定木-1

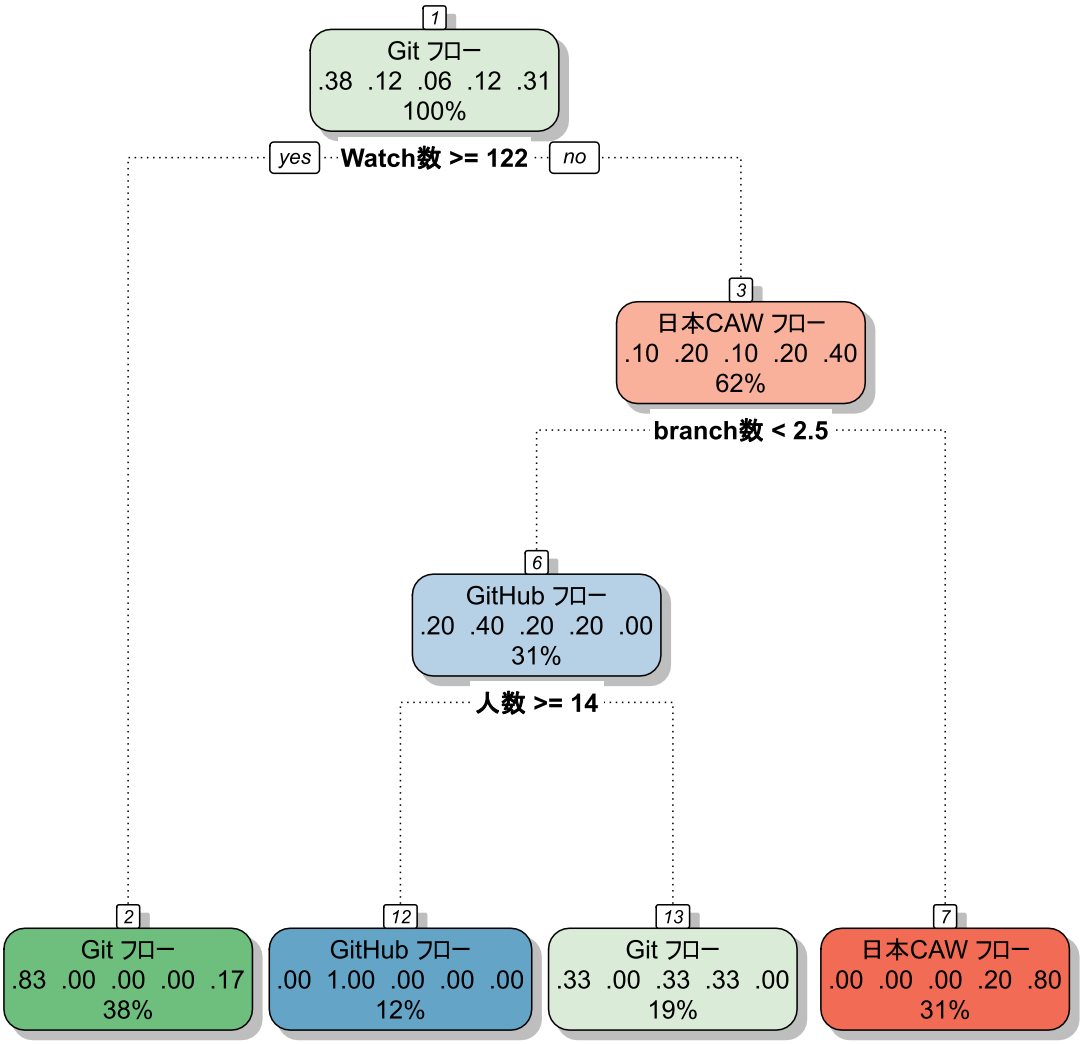


図 5.5　決定木-2

データをランダムに 2 つに分けた場合，異なった結果がでた．図 5.4 は，Issue 数，ファイル数で別れた．図 5.5 は，Ruby，行数，バイト数で別れた．

最も重要なファクターが全ての決定木で異なることがわかる．図 5.4 では，Issue 数である．図 5.5 では，Ruby である．また，別の決定木では，これらファクターは出てきていない．

5.2.3　コントロールできる要因のみかつ，成功しているプロジェクトの場合

図 5.6 は図 4.20 を用いて分析を行った結果である．

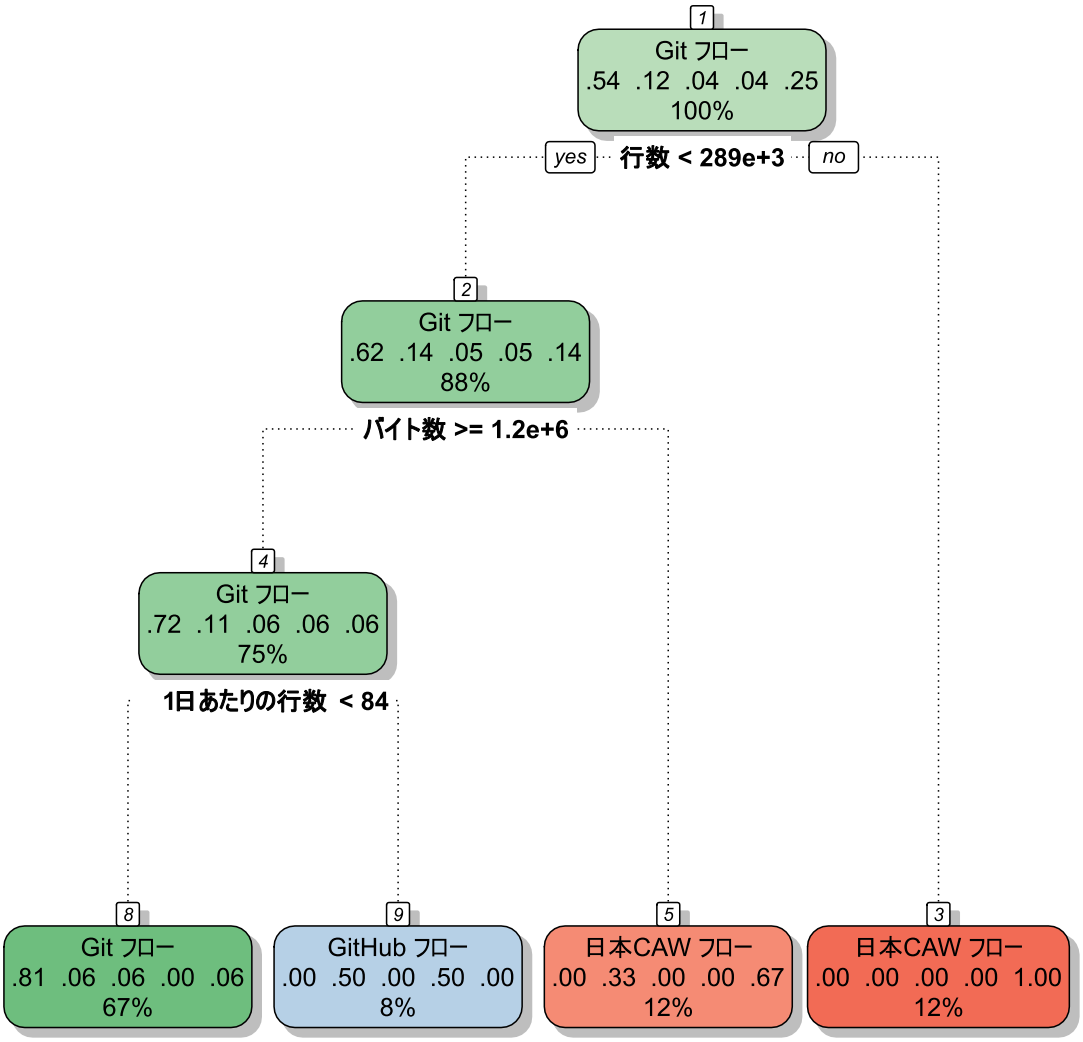


図 5.6　コントロールできる要因かつ，成功しているプロジェクトの場合

5.3　決定木性能測定結果

図 5.6 の性能を測定する．性能を測定した 10 回分の結果を，次に記す．

5.3.1　1 回目

1 回目に，決定木分析に使ったデータは，図 5.7 である．図 5.7 の決定木分析結果は，図 5.9 である．図 5.9 の性能を測定するのに使ったテストデータは，図 5.8 である．

1 回目の性能測定の結果，精度は，37.5 %だった．再現率は，50.0 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人1日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632298	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	15297	2210	80	14745686	0.091	69	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
Git フロー	19256	1662	45	2903816	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	622	459	13359619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	250603	3659	1610	29134100	1.2	69	57	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1
日本CAW フロー	911769	626	1985	66246519	0.756	146	193	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1
Git フロー	9568	1882	76	11264851	0.18	51	28	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	24372	1933	150	3879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	863309	806	2424	8885731	1.2	1100	91	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	6398	1518	152	11027830	0.13	42	32	1	1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10495817	0.194	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.7　訓練データ-1

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人1日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632298	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	15297	2210	80	14745686	0.091	69	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Git フロー	19256	1662	45	2903816	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	622	459	13359619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	250603	3659	1610	29134100	1.2	69	57	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1	1
日本CAW フロー	911769	626	1985	66246519	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1	1

図 5.8　テストデータ-1

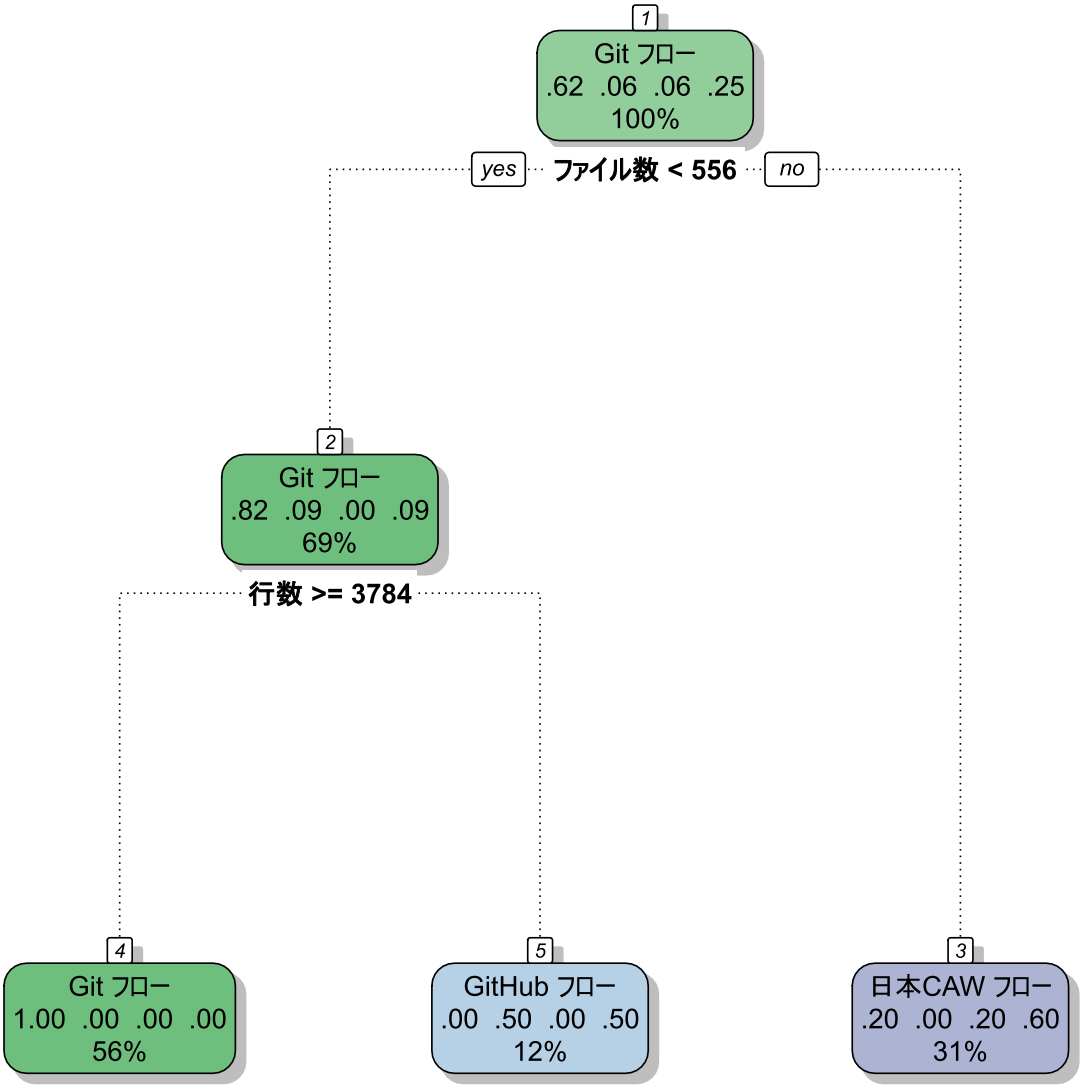


図 5.9　性能測定用決定木-1

5.3.2　2 回目

2 回目に，決定木分析に使ったデータは，図 5.10 である．図 5.10 の決定木分析結果は，図 5.12 である．図 5.12 の性能を測定するのに使ったテストデータは，図 5.11 である．

2 回目の性能測定の結果，精度は，50.0 %だった．再現率は，57.1 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1 人日あたりの行数	1 日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LINE フロー	161332	1640	600	7365044	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	250603	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1
Git フロー	15297	2210	80	14745696	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10495517	0.194	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	66246519	0.756	146	193	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1
Git フロー	9568	1882	76	17264851	0.18	5.1	28	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	622	459	13359619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	24372	1933	150	8879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
GitHub フロー	11359	597	67	2832396	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.10　訓練データ-2

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1 人日あたりの行数	1 日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
GitHub フロー	1170	1781	18	128651	0.03	0.65	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	6388	1516	152	11027830	0.13	4.2	32	1	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	217687	1220	2401	19870364	1.3	18	14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0
日本CAW フロー	863309	806	2424	88652721	12	1100	91	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.11　テストデータ-2

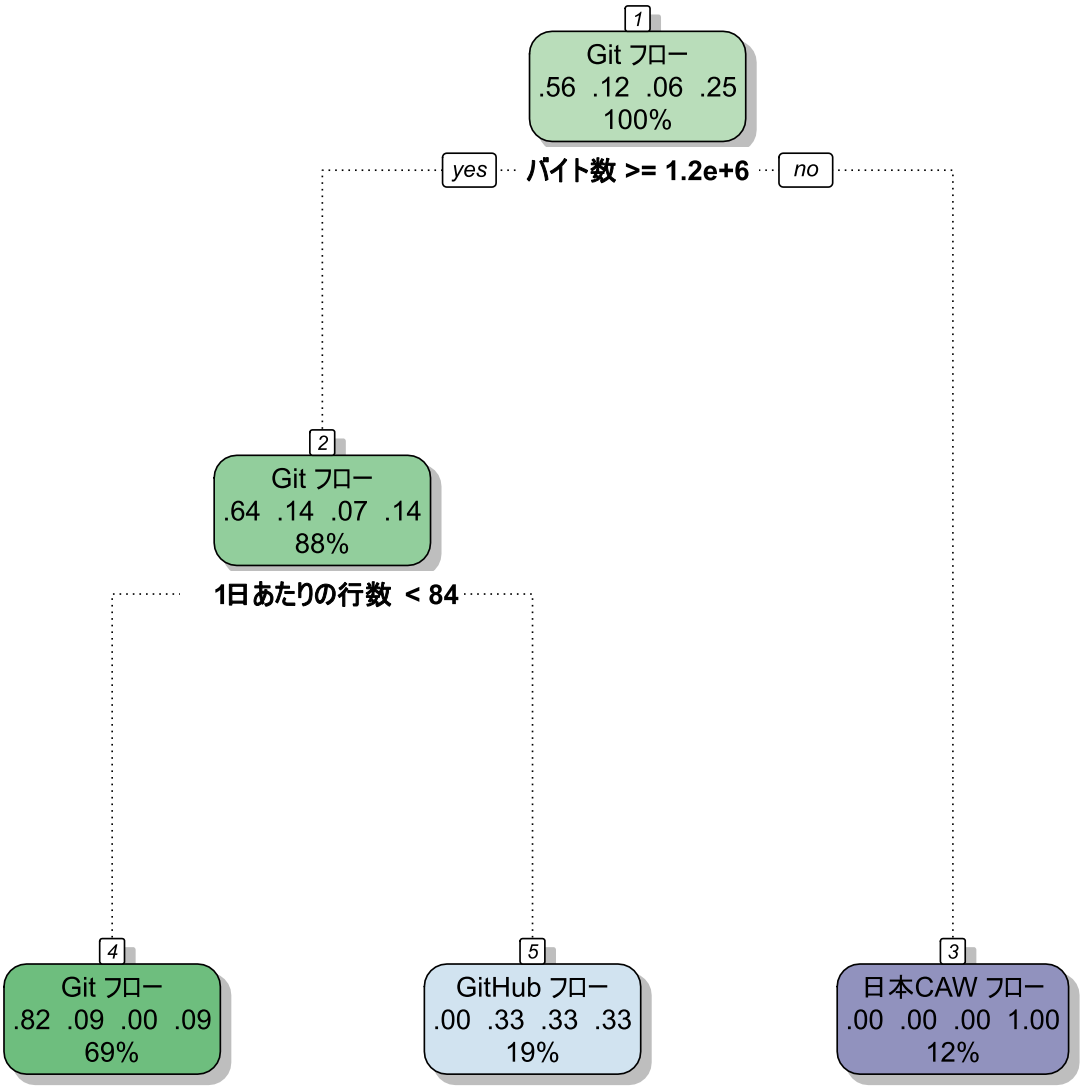


図 5.12　性能測定用決定木-2

5.3.3　3 回目

3 回目に，決定木分析に使ったデータは，図 5.13 である．図 5.13 の決定木分析結果は，図 5.15 である．図 5.15 の性能を測定するのに使ったテストデータは，図 5.14 である．

3 回目の性能測定の結果，精度は，37.5 %だった．再現率は，50.0 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	911769	626	1985	66246519	0.755	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1
GitHub フロー	1170	1781	18	128651	0.03	0.66	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10495517	0.194	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	2566408	0.095	16	169	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	250603	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1
Git フロー	24372	1933	150	887943	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	11740	1089	167	1551457	0.58	11	19	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	15297	2210	80	14745686	0.691	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0
Git フロー	217687	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	9137	814	214	1153424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	822	459	13359619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.13　訓練データ-3

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1 人日あたりの行数	1 日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	3DL	NSIS	Other
Git フロー	6398	1518	152	11027830	0.13	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	9568	1892	76	17264851	0.18	5.1	28	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632396	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LINE フロー	161332	1640	600	73850444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	863309	806	2424	88852721	12	1100	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.14　テストデータ-3

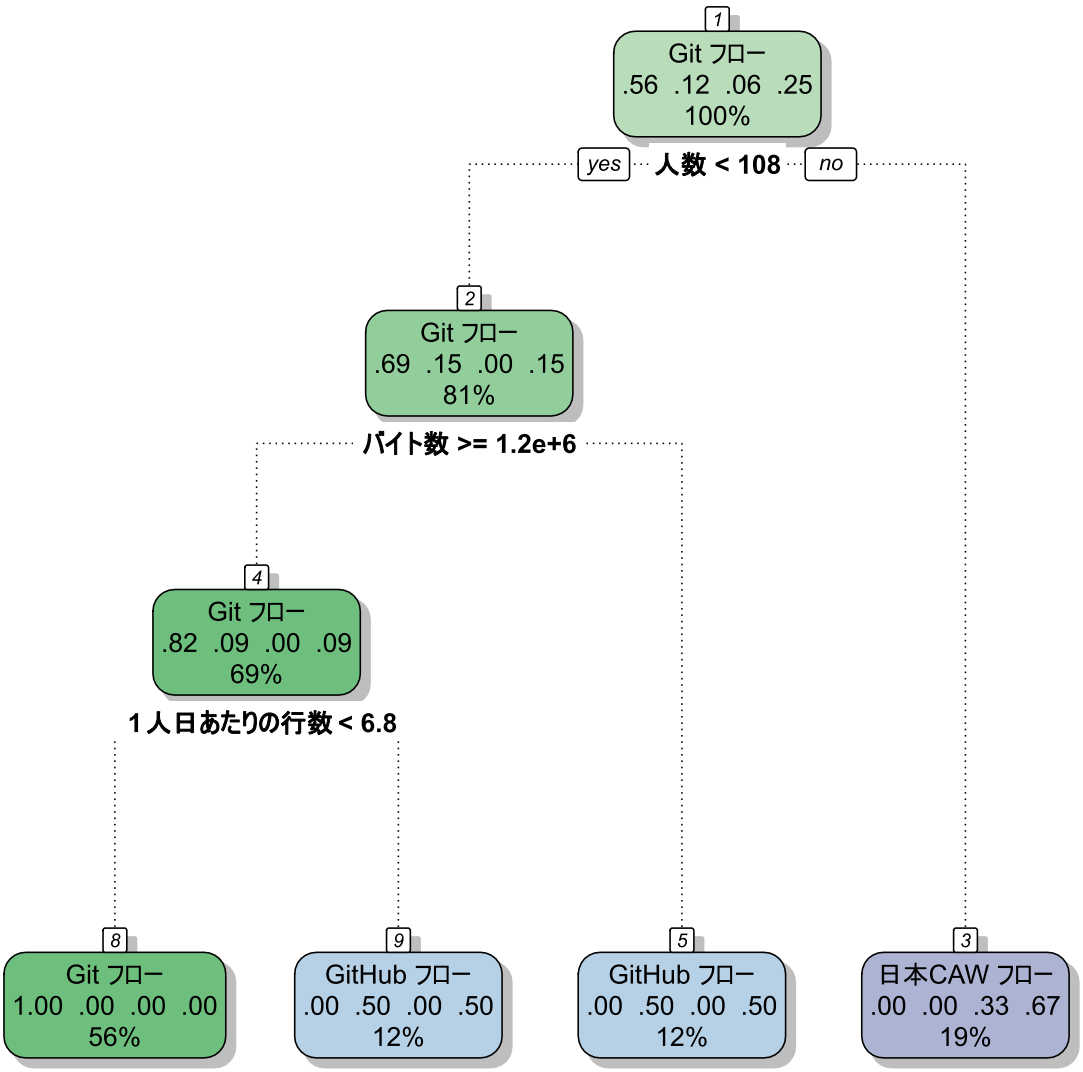


図 5.15　性能測定用決定木-3

5.3.4　4 回目

4 回目に，決定木分析に使ったデータは，図 5.16 である．図 5.16 の決定木分析結果は，図 5.18 である．図 5.18 の性能を測定するのに使ったテストデータは，図 5.17 である．

4 回目の性能測定の結果，精度は，37.5 %だった．再現率は，37.5 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
GitHub フロー	1170	1781	18	129851	0.03	0.66	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	6398	1518	152	11027830	0.13	4.2	32	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	24372	1933	150	8870431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10495517	0.184	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	622	459	1335961.9	11	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LINE フロー	161332	1640	600	73850444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	9568	1882	76	17264851	0.18	5.1	28	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0	0
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	863309	806	2424	88852721	1.2	1100	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	250803	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1

図 5.16　訓練データ-4

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	15297	2210	80	14745666	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	66246519	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1	0	1
Git フロー	217667	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632396	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	154	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.17　テストデータ-4

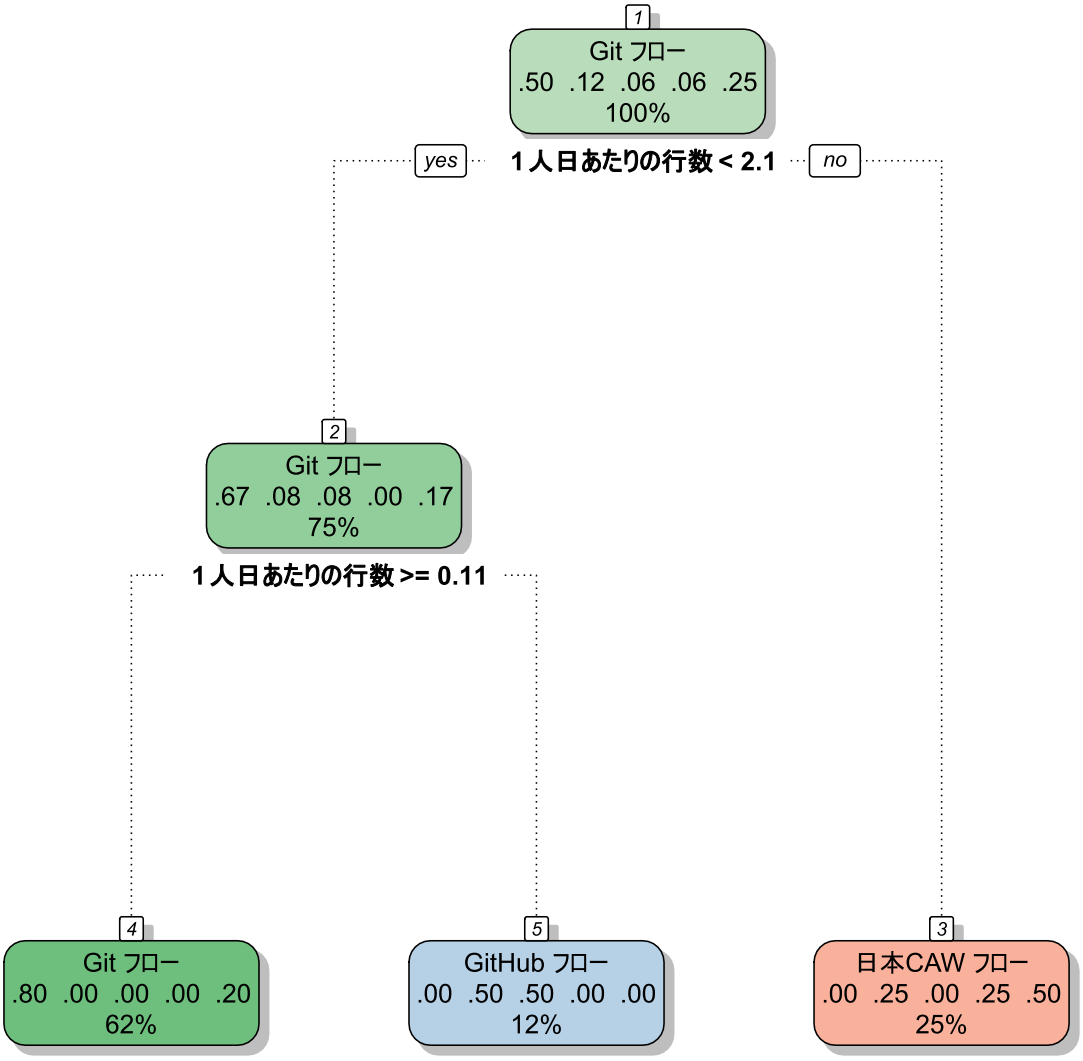


図 5.18　性能測定用決定木-4

5.3.5　5 回目

5 回目に，決定木分析に使ったデータは，図 5.19 である．図 5.19 の決定木分析結果は，図 5.21 である．図 5.21 の性能を測定するのに使ったテストデータは，図 5.20 である．

5 回目の性能測定の結果，精度は，37.5 %だった．再現率は，42.9 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10495517	0.184	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	2566928	0.095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	6398	1518	152	11027830	0.13	4.2	32	1	1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	24372	1933	150	6879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	66245519	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1
GitHub フロー	101062	622	459	1335619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	9568	1882	76	1726851	0.18	5.1	28	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2932298	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	250803	3659	1810	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1
日本CAW フロー	863309	806	2424	8885721	1.2	1100	91	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.19　訓練データ-5

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Git フロー	21767	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0
GitHub フロー	1170	1781	18	128951	0.03	0.66	22	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LINE フロー	161332	1640	600	73850444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	15297	2210	80	14745686	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0.158	26.5	170	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.20　テストデータ-5

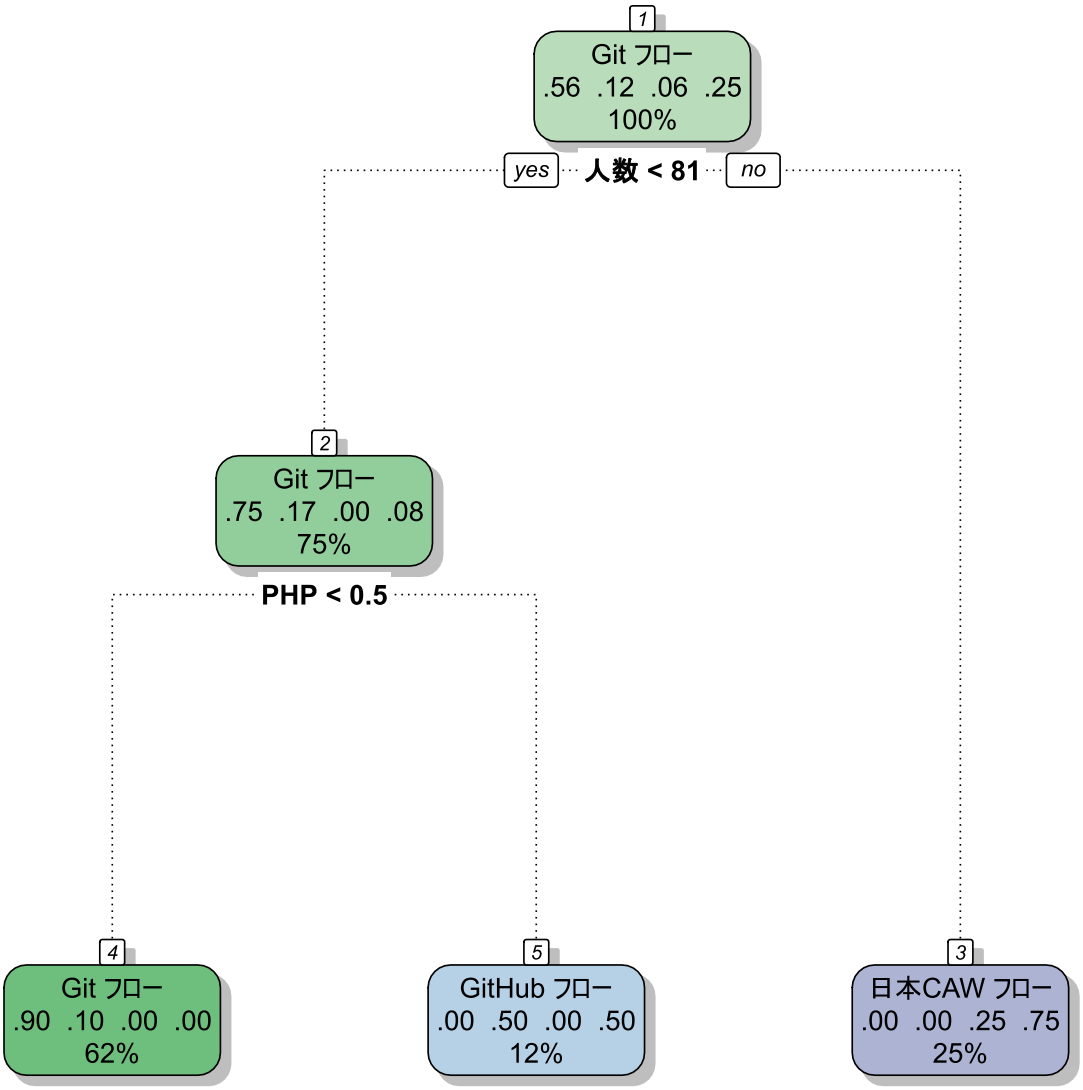


図 5.21　性能測定用決定木-5

5.3.6　6 回目

6 回目に，決定木分析に使ったデータは，図 5.22 である．図 5.22 の決定木分析結果は，図 5.24 である．図 5.24 の性能を測定するのに使ったテストデータは，図 5.23 である．

6 回目の性能測定の結果，精精度は，37.5 %だった．再現率は，42.9 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人1日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	15287	2210	80	14745686	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	6398	1518	152	1102789	0.13	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	2428378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
日本CAW フロー	863309	806	2424	6885272	1.2	1100	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	911769	626	1985	6624651	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1
Git フロー	250803	3659	1610	2913410	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1
GitHub フロー	101062	622	459	1335961	1.1	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	19256	1662	45	290281	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	42205	909	231	1979468	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	2566402	0.095	16	169	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	217687	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	1170	1781	18	129851	0.03	0.66	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.22　訓練データ-6

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1 人あたりの行数	1 日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632298	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	24372	1933	150	8879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	45673	1771	1077	10495517	0.184	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	9648	1882	76	17264851	0.18	5.1	28	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LINE フロー	161332	1640	600	73850444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.23　テストデータ-6

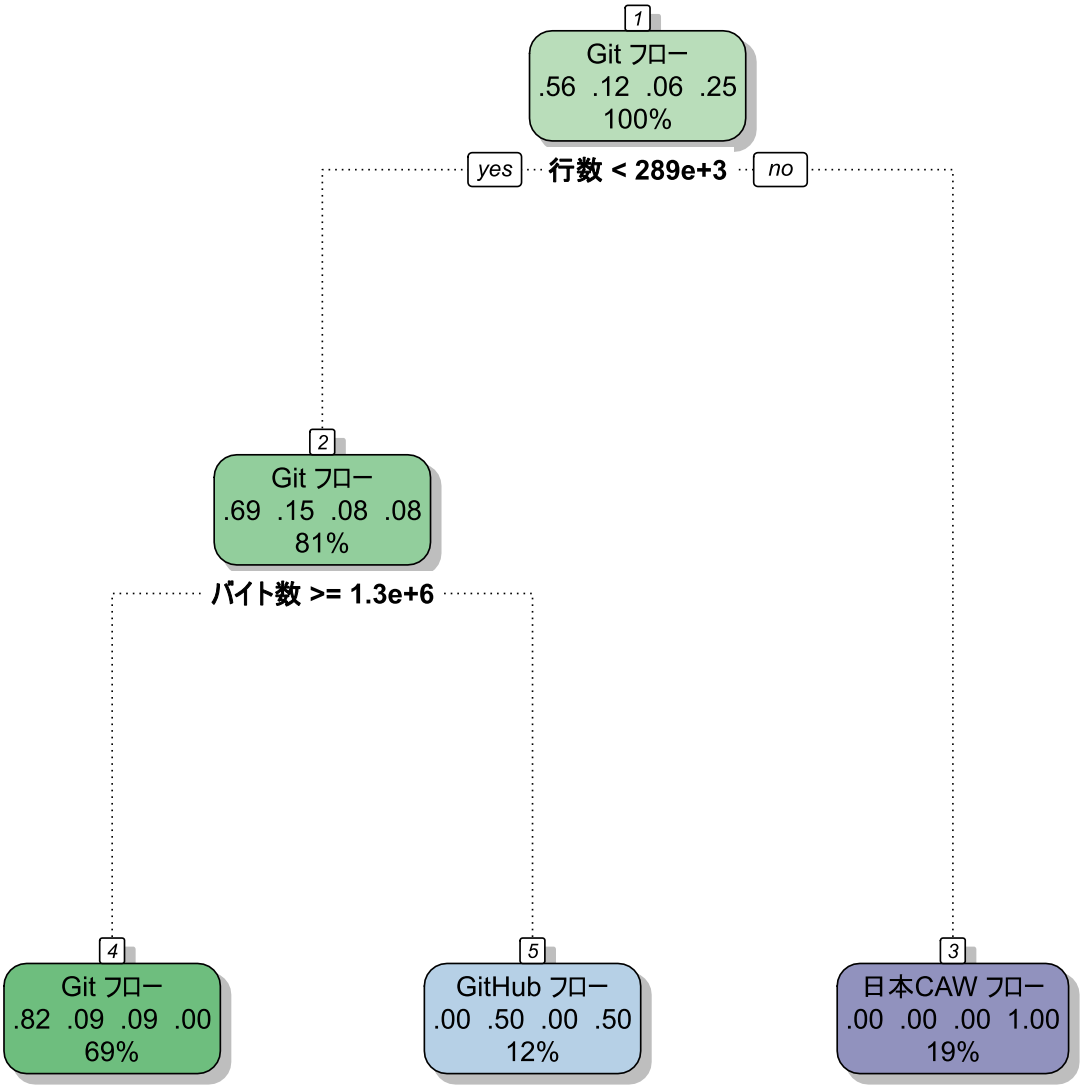


図 5.24　性能測定用決定木-6

5.3.7　7 回目

7 回目に，決定木分析に使ったデータは，図 5.25 である．図 5.25 の決定木分析結果は，図 5.27 である．図 5.27 の性能を測定するのに使ったテストデータは，図 5.26 である．

7 回目の性能測定の結果，精精度は，25.0 %だった．再現率は，50.0 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	3DL	NSIS	Other
GitHub フロー	101062	622	459	13359618	11	160	14	1	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	15297	2210	80	14745686	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0
Git フロー	6398	1518	152	11027830	0.13	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	9568	1882	76	17264851	0.18	5.1	28	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	56567	802	512	14245767	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0
GitHub フロー	11359	597	67	2632296	1.3	19	15	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	1170	1781	18	128651	0.03	0.66	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	217667	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.25　訓練データ-7

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	3DL	NSIS	Other
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	24372	1933	150	8879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	45673	1771	1077	10495517	0.184	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	250609	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1
日本CAW フロー	911769	626	1985	66246519	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1
LINE フロー	161332	1640	600	23850444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	863309	806	2424	8865272	1.2	1100	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.26　テストデータ-7

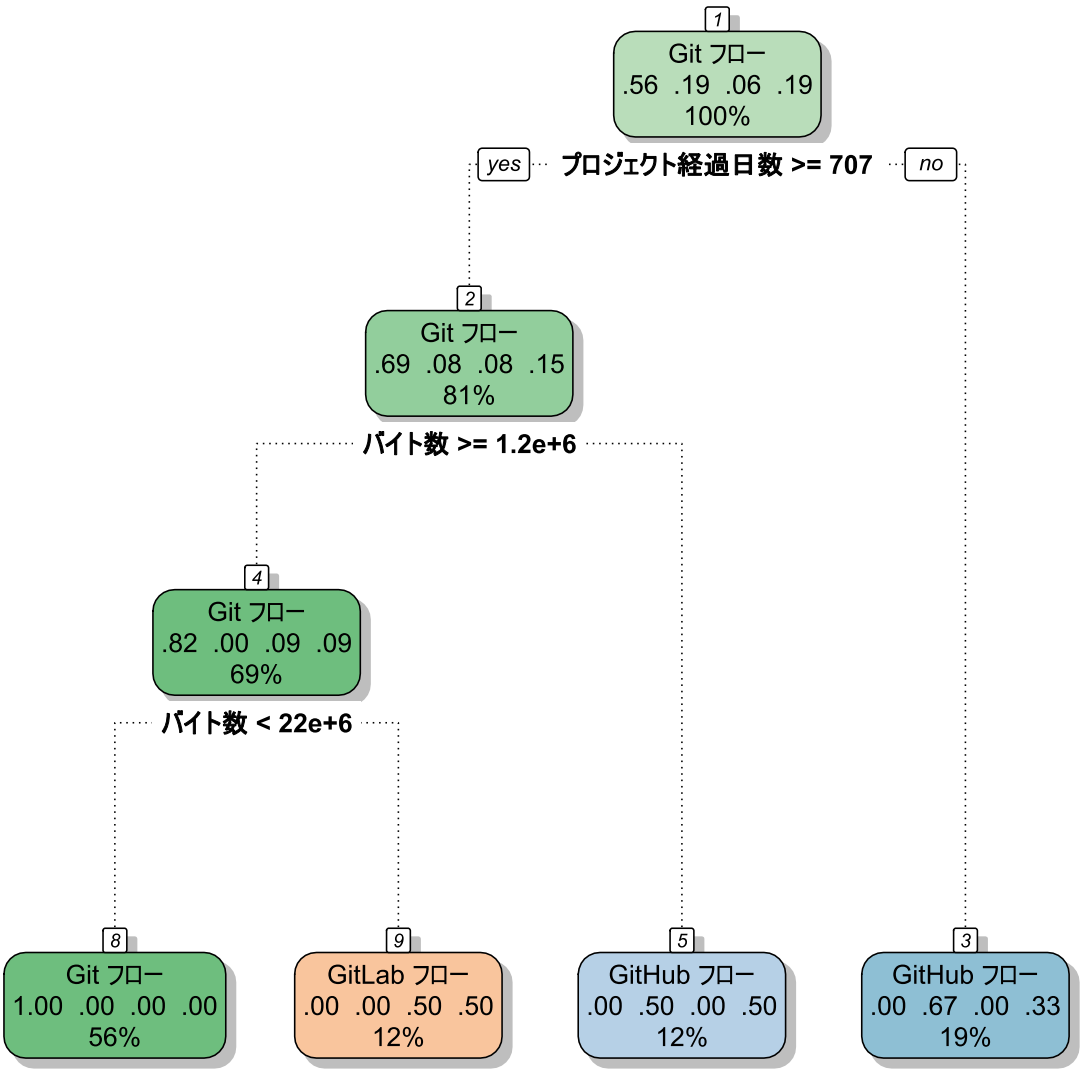


図 5.27　性能測定用決定木-7

5.3.8　8 回目

8 回目に，決定木分析に使ったデータは，図 5.28 である．図 5.28 の決定木分析結果は，図 5.30 である．図 5.30 の性能を測定するのに使ったテストデータは，図 5.29 である．

8 回目の性能測定の結果，精精度は，37.5 %だった．再現率は，42.9 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	6398	1518	152	11027830	0.13	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0
GitLab フロー	54218	3395	323	25664028	0.095	16	169	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	9137	814	214	1123404	0.41	11	27	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632298	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	15297	2210	80	14745686	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
日本CAW フロー	863309	806	2424	8885721	1.2	1100	91	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	24372	1933	150	8879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	6624551	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	1
Git フロー	9568	1882	76	17264851	0.18	5.1	28	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	1170	1781	18	139851	0.03	0.66	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

図 5.28　訓練データ-8

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	485	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
GitHub フロー	101062	622	459	13359619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LINE フロー	161332	1640	600	73850444	2.5	98	38	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	45673	1771	1077	10495517	0.184	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	250803	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1
Git フロー	21787	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.29　テストデータ-8

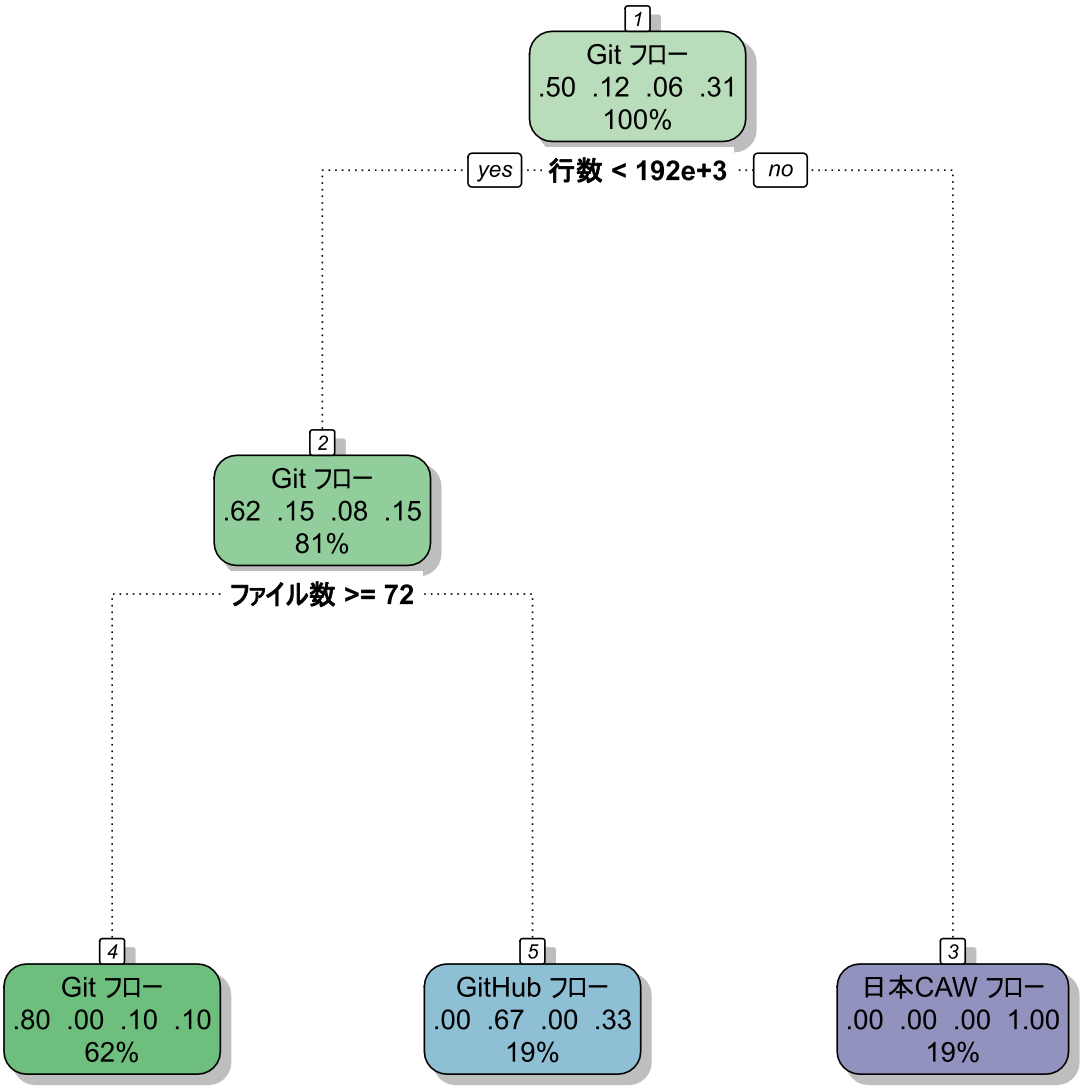


図 5.30　性能測定用決定木-8

5.3.9　9 回目

9 回目に，決定木分析に使ったデータは，図 5.31 である．図 5.31 の決定木分析結果は，図 5.33 である．図 5.33 の性能を測定するのに使ったテストデータは，図 5.32 である．

9 回目の性能測定の結果，精精度は，50.0 %だった．再現率は，57.1 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人当あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	217687	1220	2401	18870964	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	8276	792	134	1365551	2	10	5	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	56567	802	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	24372	1933	150	8879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	23618	485	226	2161679	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	45673	1771	1077	10495517	0.194	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	1170	1781	18	128651	0.03	0.66	22	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	66240519	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0
Git フロー	9568	1892	76	1726485	0.18	5.1	28	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LINE フロー	161332	1640	600	73650444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
GitHub フロー	11359	597	67	2632286	1.3	19	15	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.31　訓練データ-9

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1 人あたりの行数	1 日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	18256	1662	45	2903916	017	12	71	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	6398	1518	152	11027830	013	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	101062	622	459	13359619	11	160	14	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	15297	2210	80	14745696	0091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
Git フロー	71955	2714	319	21424103	0156	26.5	170	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
GitLab フロー	54218	3395	323	25664208	0095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	250803	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1
日本CAW フロー	863309	806	2424	88852721	12	1100	91	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

図 5.32　テストデータ-9

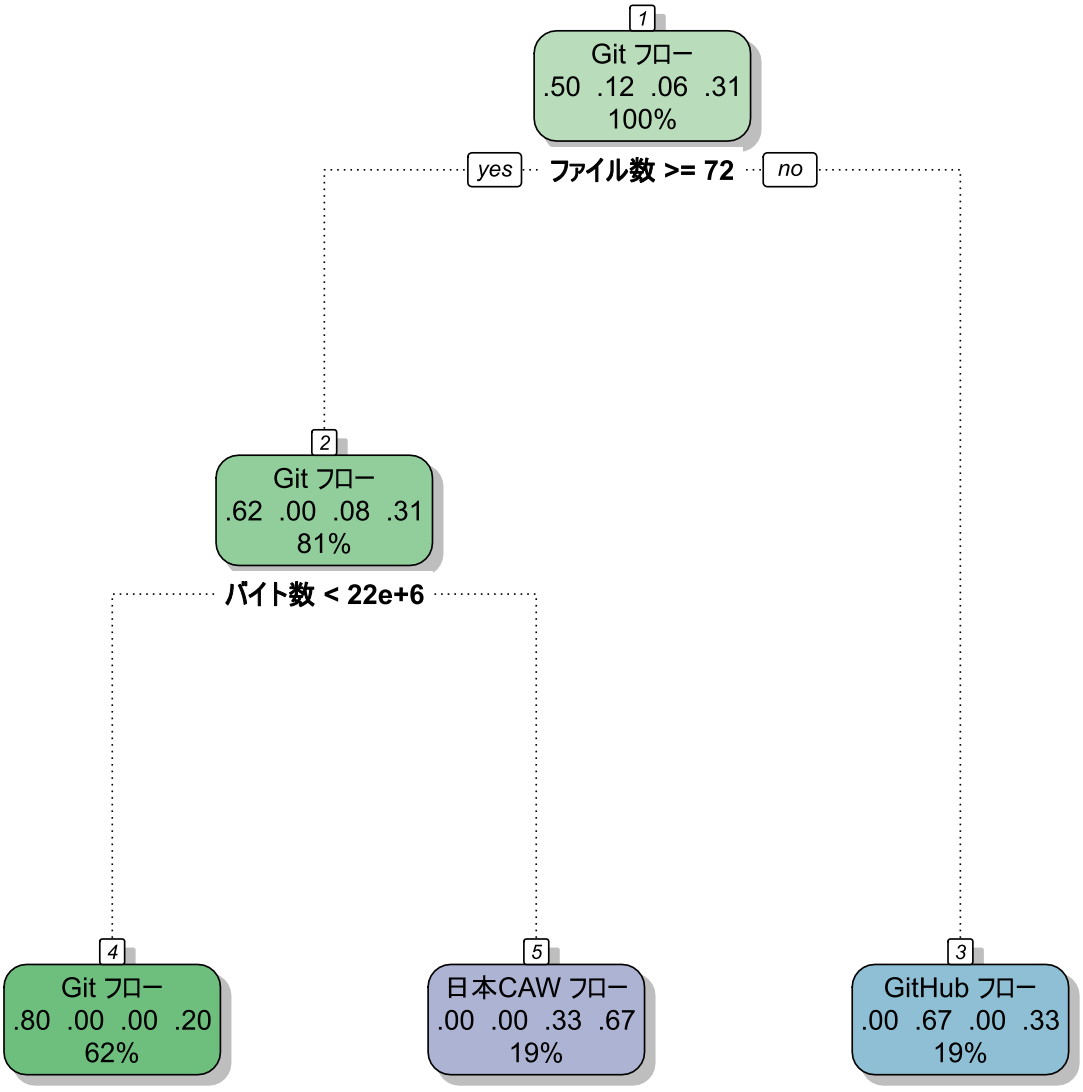


図 5.33　性能測定用決定木-9

5.3.10　10 回目

10 回目に，決定木分析に使ったデータは，図 5.34 である．図 5.34 の決定木分析結果は，図 5.36 である．図 5.36 の性能を測定するのに使ったテストデータは，図 5.35 である．

10 回目の性能測定の結果，精精度は，25.0 %だった．再現率は，28.6 %だった．

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人1日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
Git フロー	6398	1518	152	11027830	0.13	4.2	32	1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	19256	1662	45	2903916	0.17	12	71	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	15297	2210	80	14745696	0.091	6.9	76	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
Git フロー	24372	1933	150	8879431	0.21	13	61	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	42205	909	231	19794687	0.7	46	66	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	911769	626	1985	66246519	0.756	146	193	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1	
GitHub フロー	101062	622	459	1335961.9	1.1	160	14	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	217687	1220	2401	18870364	1.3	18	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	863309	806	2404	8885731	1.2	1100	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
日本CAW フロー	45673	1771	1077	10495517	0.184	25.8	140	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitLab フロー	54218	3395	323	25564028	0.095	16	169	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	11359	597	67	2632298	1.3	19	15	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	9568	1882	76	17264651	0.18	5.1	28	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	326788	1113	1370	24285378	21	290	14	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0
Git フロー	71955	2714	319	21424103	0.156	26.5	170	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	8276	792	134	1285551	2	10	5	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

図 5.34　訓練データ-10

workflow	行数	プロジェクト経過日数	ファイル数	バイト数	1人1日あたりの行数	1日あたりの行数	人数	HTML	JavaScript	CSS	Makefile	C	PHP	Java	Python	Ruby	TeX	CoffeeScript	Perl	Shell	VimL	Julia	C++	Erlang	LiveScript	Objective-C	EmacsLisp	Lua	Cmake	Inno Setup	IDL	NSIS	Other
日本CAW フロー	1157	472	9	1132534	0.3	2	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
GitHub フロー	1170	1781	18	128851	0.03	0.66	22	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	11740	1099	167	1551457	0.58	11	19	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
日本CAW フロー	9137	814	214	1123424	0.41	11	27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	23618	465	228	2161879	2.6	49	19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Git フロー	56567	902	512	14245757	1.8	71	40	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LINE フロー	161332	1640	600	73850444	2.5	98	39	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Git フロー	250803	3659	1610	29134100	1.2	69	57	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1

図 5.35　テストデータ-10

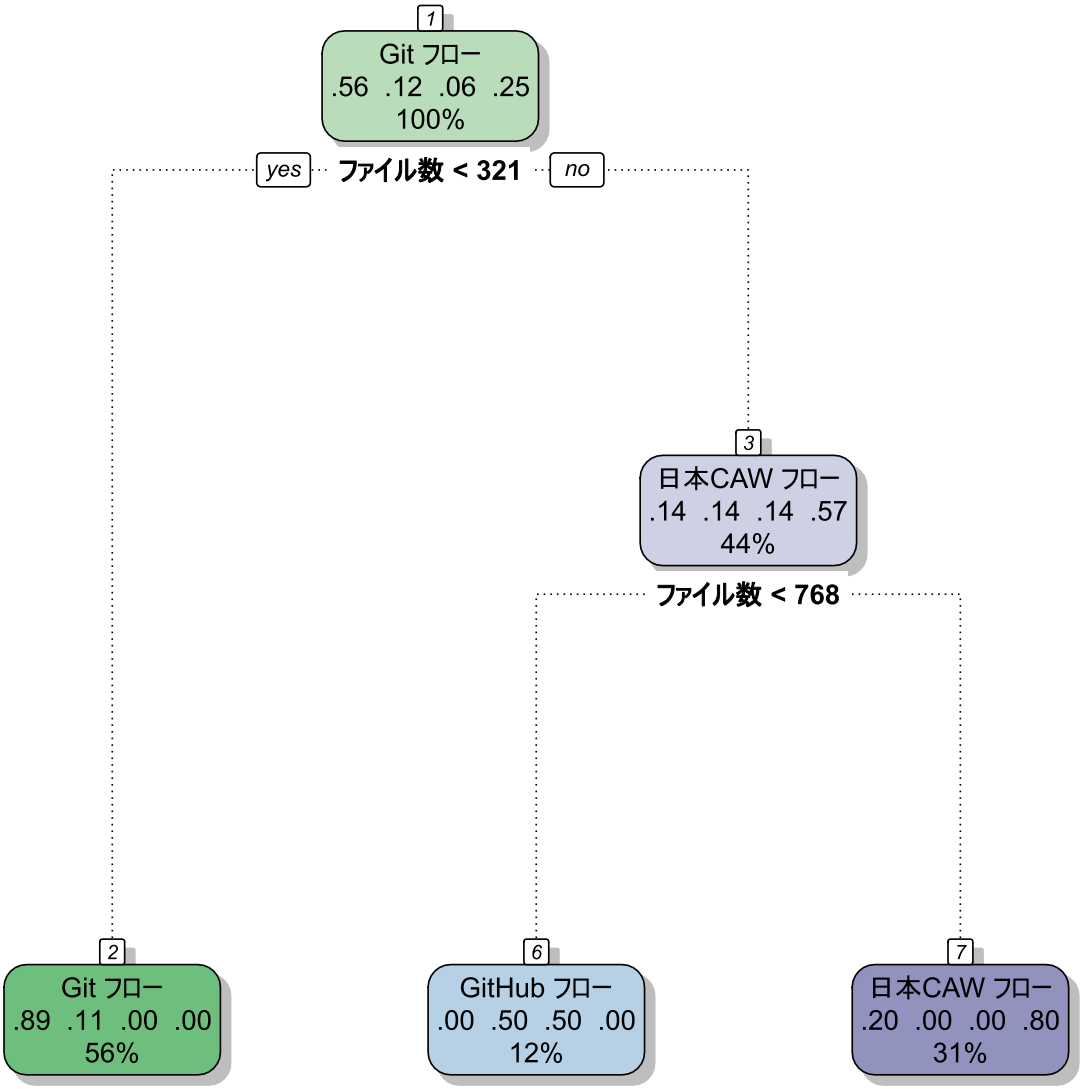


図 5.36　性能測定用決定木-10

5.3.11　平均と信頼区間

精度の平均は，37.5% であった．精度の信頼区間は，32.6 42.4% であった．再現率の平均は，45.9% であった．再現率の信頼区間は，40.7 51.1% であった．

第 6 章

考察

6.1 すべてのデータで行った場合

図 5.3 を考察する．図 5.3 は，図 4.17 を用いて分析を行った結果である．

図 5.3 より，Star 数，人数，OpenPullRequest 数，Watch 数，ファイル数，branch 数，JavaScript で開発フローを再現できることがわかった．また，リーフは，Git フローと GitHub フローと LINE フローと日本 CAW フローと GitLab フローである．

それぞれの分岐要因を考察していく．

Star 数により，選択するフローが変わっている．ここから，開発に直接かかわる人だけでフローが決まらないことが言える．これは，開発に関心がある人，注目している人も含めて，フローを選択しなければならないことが，想定される．

人数により，選択するフローが変わっている．変わっているのは，Git フローと日本 CAW フローである．Git フローと日本 CAW フローの大きな違いは，複雑度である．人数が 84 人以上の場合，複雑すぎると開発しづらいことから，より複雑度が低い日本 CAW フローを選択していることが，想定される．

1 日あたりの commit 数により，選択するフローが変わっている．ここからプロジェクトの生産性により，適正な開発フローが異なることが言える．3 番目に重要な指標となっていることから，ほかの生産性の指標を用いることで，より正確なフローを選択できるようになるだろう．

6.2 データをランダムに並び替え，2 つに分けた場合

図 5.9 と図 5.12 を比較して，考察する．図 5.9 は，図 4.18 を用いて分析を行った結果である．図 5.12 は，図 4.19 を用いて分析を行った結果である．

図 5.9 は，一日あたりの commit 数と ClosedPullRequest 数で，分岐している．また，リーフは，Git フローと GitHub フローと日本 CAW フローである．

図 5.12 は，Watch 数と branch 数と人数で，分岐している．また，リーフは，Git フローと GitHub フローと日本 CAW フローである．

2 つの決定木を見比べると，分岐が大きく異なっていることがわかる．

大きく異なっていることから，16 件に分けられたデータのばらつきが大きいことがわか

る．ここから，精度と再現率が低い可能性があることがわかる．

そのため，決定木の性能測定を行う必要がある．

以上までの考察を含めて，以下の改善を行う．

1 つ目は，指標を絞る．ばらつきが大きいことがわかったため，指標を絞ることにする．指標を絞ることで，ばらつきを減らし，精度と再現率を上げられると想定する．

指標は，プロジェクト開始時にわかっているものにした．プロジェクト開始時にわかっているものは，行数と人数と言語と生産性と規模である．

2 つ目は，成功しているプロジェクトに絞る．成功していないプロジェクトを含めると，最適なフローを選択できていない可能性がある．そのため，データに矛盾が生じている可能性が考えられる．データに矛盾があると，決定木がうまく再現できない．

成功しているプロジェクトの基準は，リリースを 1 以上していることにする．

6.3　コントロールできる要因のみかつ，成功しているプロジェクトの場合

図 5.6 を考察する．図 5.6 は，図 4.20 を用いて分析を行った結果である．

図 5.6 は，行数とバイト数と 1 日あたりの行数で，分岐している．行数とバイト数は，プロジェクトの規模に関わる．1 日あたりの行数は，プロジェクトの生産性に関わる．

分岐より，プロジェクトの規模と生産性が，重要な指標であることがわかった．

プロジェクトが大きくなるにつれて，管理が難しくなる．その管理を助けるためにフローを変える必要があるのだろう．

プロジェクトの生産性は，作業の多さに関わる．テストが多かったり，難しい開発であれば，生産性は大きく変わる．プロジェクトが進むスピードに比べて，複雑すぎるフローだと，開発を送らせてしまう．簡単すぎるフローだと，開発を進ませすぎてしまい，バグが多く入ってしまう可能性が考えられる．

6.4　精度と再現率

精度と再現率は，あまりよくない結果になった．

よくない結果になった要因は，人手で選んだ開発フローが間違えている，プロジェクトの重要な指標を調査できていない等があげられる．

人手で選んだ開発フローが間違えていることの対処は，第三者に判定してもらう方法がある．一人が選択したより，二人が選択したほうが精度は高くなるだろう．

プロジェクトの重要な指標を調査できていないことの対処は，プロジェクトの生産性に注目する．現在では，日数で割った指標しか用いていない．ここまでの分析で，プロジェクトの生産性が，要因になることがわかった．そのため，人数の増減傾向や，特定の曜日に開発が進む等，より詳細なプロジェクトの生産性を表していない．人数の増減傾向は，人がどのタイミングで増減したのかわかれば，指標として用いることが可能だろう．また，ソフトウェア開発プロジェクトは，開発が進むにつれて人数の変化がある．開発段階は人数が増加する．納入段階に近づくにつれて，減少が起こる．特定な曜日に開発が進み，特

定の曜日にテストを行うプロジェクトもある.

より詳細な指標を用いれば，新しい要因を作ることが出来，精度と再現率を上げることができるだろう.

第 7 章

結論

本研究により，開発フローを再現手法を提案した．GitHub 上のプロジェクトから性質と開発フローを調査し，決定木分析を行うことで，再現した．

この再現手法は，精度と再現率が高いとはいえない．精度は，37.5% だった．再現率は，45.9% だった．

しかし，精度と再現率を高めることができれば，開発経験の浅いプロジェクトでも，最適なフローを選択できるようになるだろう．

参考文献

[1] 小野寺航己. バージョン管理システムを活用するソフトウェア開発の開発フロー. 卒業論文, 千葉工業大学, 2015.

[2] 池田尚史, 藤倉和明, 井上史彰. チーム開発実践入門〜共同作業を円滑に行うツール・メソッド. 技術評論社, 2014.

[3] 松田航. システム開発の質を高めるバージョン管理ツールとは？ http://www.linuxacademy.ne.jp/lablog/programmer/319/（2016.01.22 閲覧）.

[4] Ken Nishimura. Gitlab flow から学ぶワークフローの実践. http://postd.cc/gitlab-flow/（2016.01.22 閲覧）.

[5] 可知豊. アップルが swift の oss 化で狙うもの–github で企業とコミュニティーの連携が進む. http://japan.zdnet.com/article/35075308/（2016.01.22 閲覧）.

[6] 野々下裕子. Github 共同創業者が和歌山県庁を表敬訪問し知事と対談. http://japan.cnet.com/news/business/35075525/（2016.01.22 閲覧）.

[7] 野々下裕子. 初心者から参加できる github 習得イベント「github patchwork」が神戸で開催. http://japan.cnet.com/news/business/35075265/（2016.01.22 閲覧）.

[8] Charlie Osborne. Github のサポート体制にユーザーが苦情–「完全に無視されている」. http://japan.zdnet.com/article/35076469/（2016.01.22 閲覧）.

[9] 大塚弘記. GitHub 実践入門 Pull Request による開発の変革. 技術評論社, 2014.

[10] harada4atsushi. Github flow で pull request ベースな開発フローの進め方. http://qiita.com/harada4atsushi/items/527d5f98320d993b3072（2015.10.03 閲覧）.

[11] hayaishi. Line ios アプリ開発についてのご紹介. http://developers.linecorp.com/blog/?p=2921（2015.10.03 閲覧）.

[12] 竹内俊彦. はじめての S-PLUS/R 言語プログラミング– 例題で学ぶ S-PLUS/R 言語の基本. オーム社, 2005.