修士学位論文

題目

ソフトウェア開発PBLにおけるタスクの作業時間の見積もり手法に関する調査と改善

指導教員
楠本 真二 教授

報告者
古田 雄基

平成 29 年 2 月 7 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻

平成 28 年度 修士学位論文

ソフトウェア開発 PBL におけるタスクの作業時間の見積もり手法に関する調査と改善

古田 雄基

内容梗概

実践的なソフトウェア開発の教育としてソフトウェア開発 PBL(Project-based Learning) が活発に行われている．PBL は企業が行うソフトウェア開発プロジェクトとは異なり，プロジェクトメンバはソフトウェア開発を学習しながら，プロジェクトに参加している．そのため，十分なソフトウェア開発技術を持っていないままソフトウェア開発を行っている可能性が高い．その影響で，PBL ではソフトウェア開発で発生し得るリスクが顕在化しやすいと考えられている．そのリスクの一つに作業時間の見積もりの甘さが挙げられる．一般にソフトウェア開発の見積もりは開発者の経験に基づく部分が多いため，PBL では見積もりの甘さが発生しやすいと考えられる．作業時間の見積もりが不正確であった場合，学習者たちが計画の不備に気づけず，本来習得・経験すべきであった実践的な開発技術が学習できないままプロジェクトが破綻してしまう可能性があるため，ソフトウェア開発 PBL においても作業時間を正確に見積もることは重要である．また，近年アジャイルソフトウェア開発が注目を集めており，PBL に於いても適用される事例が増えている．そこで，本稿ではアジャイルソフトウェア開発を行う PBL におけるタスクの作業時間の見積もりを自動的に行う手法について調査を行った．その結果，機械学習を用いた既存研究の手法に改良を加えることにより，開発者の見積もりよりも良い精度でタスクの作業時間を予測することが出来た．

主な用語

ソフトウェア工学
アジャイルソフトウェア開発
Project Based Learning
見積もり
機械学習

目 次

表 目 次

図 目 次

1　まえがき

ソフトウェア開発において，プロジェクトを円滑に遂行するためには，工数予測が重要である [1][2][3][4]．工数とは，開発期間と要員の積で算出される延べ作業時間を表す数値であり，プロジェクトを管理する上で重要な指標となる．工数を正確に予測し，必要な工数を把握することによって，スケジュールの適切な管理や開発資源の割り当てを行うことができる．結果として，納期遅れやコスト超過といったプロジェクトの失敗のリスクを抑えることが可能となる．

実践的なソフトウェア開発技術を教育する手法としてソフトウェア開発PBL(Project-based Learning) が活発に行われている．ソフトウェア開発 PBL[5] とは，学習者同士でグループを組ませて，ソフトウェア開発プロジェクトに取り組ませる教育手法である．ソフトウェア開発 PBL では，プロジェクトメンバがソフトウェア開発技術を学びながらプロジェクトに取り組んでいる．そのため，開発者が十分なソフトウェア開発技術を持っていない可能性があり，企業が行うソフトウェア開発プロジェクトよりも作業時間の見積もりが甘くなるというリスクが発生しやすい．一般にソフトウェア開発の見積もり手法の多くは開発者の経験に基づいたものであるため，ソフトウェア開発 PBL では見積もりが不正確になりやすいと考えられる．不正確な見積もりはスケジュールの適切な管理を困難にし，結果としてプロジェクト計画が破綻し，学習者たちが習得・経験する予定であった実践的な開発技術が学習できない状況が発生し得る．

また，近年アジャイルソフトウェア開発 [6][7][8] が注目を集めており，ソフトウェア開発 PBL に於いても適用される事例が増えている．アジャイルソフトウェア開発において作業時間などの見積もりは，多くの場合，人間の主観に基づいて集団で議論を行う見積もり手法が用いられる．人間は一般的に楽観的な予測を行う傾向があることが知られており，人間の主観に基づいた見積もりは過小見積もりになる傾向がある．そして人間は集団で予測を行う場合より楽観的な予測を行う傾向があるということが知られている [9]．つまり，人間の主観に基づいて集団で議論を行う見積もり手法には過小見積もりを導くという問題が存在する．また，人間の主観に基づいて集団で議論を行う見積もり手法は，開発者の経験にその精度が依存するという問題も存在する [10]．そのため，人間の主観に基づかない，自動的な見積もり手法が必要とされている．

そこで，本研究では，アジャイルソフトウェア開発を取り入れたソフトウェア開発 PBL のプロジェクトにおいて，開発者の経験に基づかずに作業時間の見積もりを行うために，タスクの作業時間の見積もりを自動的に行う手法に関して調査を行った．

具体的には，まず，アジャイルソフトウェア開発プロジェクトにおける自動的な見積もり手法の既存研究である Porru らの研究 [9] で用いられていた機械学習に基づく自動的な見積

もり手法に関する調査を行った．Porru らの研究ではタスクの内容について記述されている Issue Report から自然言語解析などを用いてタスクの特徴を抽出し，それらに機械学習の手法を適用することでアジャイルソフトウェア開発でソフトウェアの規模を示す際に用いられる Story Point の値を予測をするという研究であった．そして，機械学習に基づく手法の改善を目的とした調査として，さらに 2 つの調査を行った．1 つ目は対象とするプロジェクト内のデータだけでなく，過去 4 年間にソフトウェア開発 PBL で実施されたプロジェクトのデータを用いることで精度が向上するかの調査を行った．もう 1 つは Porru らの研究研究においても重要とされていたタスクの種類を用いてデータを分類することによって精度が向上するかの調査を行った．

調査では，機械学習に基づく自動的な見積もり手法が出力するタスクの作業時間の予測値と，開発者らがプロジェクト中に実際に予測したタスクの作業時間の予測値のそれぞれに対して，実際にかかったタスクの作業時間の実測値との誤差を求め，その結果を比較することで機械学習に基づく自動的な見積もり手法の有効性を評価する．

以降，2 章では研究の背景となる関連研究について述べる．3 章では本研究で行った調査について述べる．4 章では本研究の妥当性の脅威について述べる．最後に 5 章で本研究のまとめを述べる．

2　準備

2.1　ソフトウェア開発 PBL

ソフトウェア開発 PBL[11] とは，ソフトウェア開発教育で用いられる実践的な教育手法の 1 つである．PBL（Project-based Learning）とは，期限が定められており，新たな価値の創造を目的としたプロジェクトと呼ばれる形式で，受講生が主体的にプロジェクト運用に必要な多様なスキルを獲得することを目的とした学習手法である [12]．ソフトウェア開発 PBL の実例としては，沢田ら [13] の飛行船制御ソフトウェアを開発するプロジェクトや，松澤ら [14] の PM を目指す企業の技術者と大学生の共同によるプロジェクト等がある．

ソフトウェア開発 PBL では，学習者同士のグループでソフトウェア開発プロジェクトに取り組ませ，コーディングやテストといった開発の下流工程だけでなく，ソフトウェアの設計といった上流工程も経験させることによって，より実践的なソフトウェア開発技術を学習することが出来るという特徴がある．

また，ソフトウェア開発 PBL を行う上で考慮すべき問題として，十分な開発技術を持たない可能性のある開発者が開発技術を学びながらプロジェクトに取り組むため，ソフトウェア開発におけるリスクが顕在化しやすいと言われている [5]．そのリスクの中には見積もりの甘さによる納期遅れが発生しやすいということも含まれている．

ソフトウェア開発 PBL のプロジェクトでは，開発者の経験が不足しているためにプロジェクトで発生する作業にかかる時間の見積もりが不正確なものになりやすいと考えられる．そして，そのような不正確な見積もりはプロジェクトの計画の破綻を導く可能性がある．プロジェクトの計画が破綻してしまうと本来学習者が習得・経験すべきであった開発技術が学習できないという事態になりかねない．そのため，ソフトウェア開発 PBL のプロジェクトにおいても，正確な作業時間の見積もりは重要である．

また，近年はアジャイルソフトウェア開発をソフトウェア開発 PBL に取り入れた事例が増加している [15], [16], [17].

2.2　アジャイルソフトウェア開発

アジャイルソフトウェア開発とは，小さなイテレーションを繰り返しながらソフトウェア開発を行う開発手法である．図 1 はイテレーションの概略図である．

イテレーションは以下の 4 つの工程でひとまとまりになっている．

1. そのイテレーションで取り組む内容を決める計画
2. ソフトウェアの開発作業

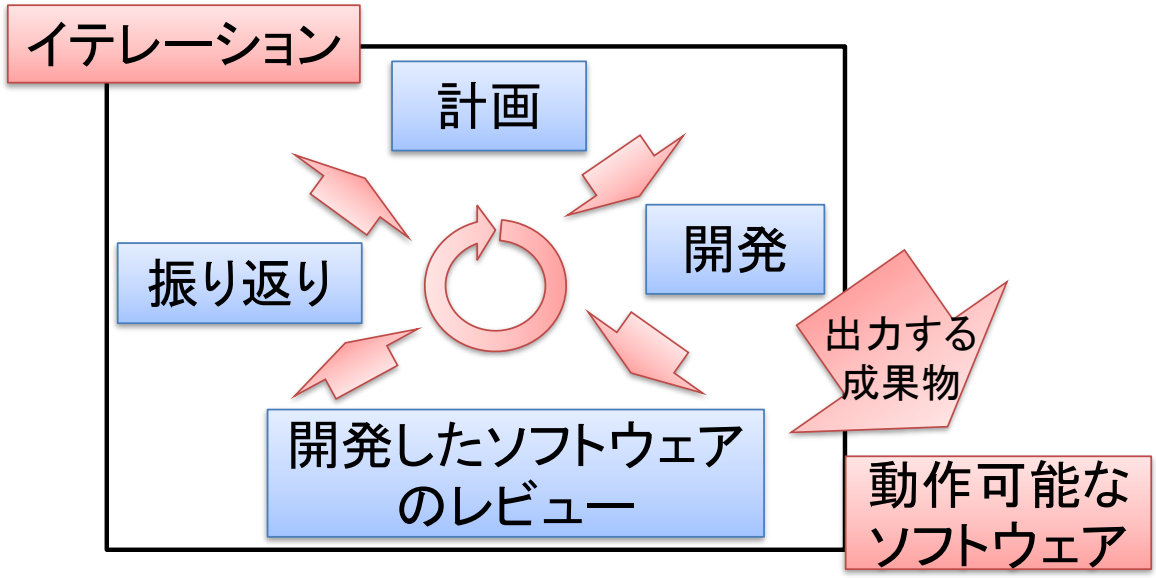

図 1: アジャイルソフトウェア開発のイテレーションの概略図

3. 開発したソフトウェアのレビュー

4. イテレーション全体の振り返り

アジャイルソフトウェア開発では，イテレーション毎のソフトウェアの開発作業が終わった段階で動作可能なソフトウェアが存在するという特徴がある．

また，開発者はイテレーションの初めの計画の段階で取り組む内容の作業時間の見積もりを行う．その見積もりには主観的な評価に基づく見積もり手法が最も用いられている [18]．その中で広く用いられている手法の 1 つに Planning Porker がある．

2.2.1 既存の見積もり手法: Planning Porker

アジャイルソフトウェア開発において広く用いられている見積もり手法に「Planning Poker」と呼ばれる手法がある [19]．Planning Poker は開発メンバー全員で予測対象のユーザーストーリーについて議論を行い，そのユーザーストーリーの Story Point を予測する手法である．

- ユーザーストーリー

  アジャイルソフトウェア開発で用いられる仕様書．ソフトウェアの機能について開発者が理解できる最低限の文書が書かれている．

- Story Point

  アジャイルソフトウェア開発で広く用いられるストーリーの規模を示す値 [18]．あるユーザーストーリーを基準として，他のユーザーストーリーが相対的に何倍程度の難しさであるかを表現した値である．ただし，Story Point は開発チームが考える難しさを示す主観的な値であるという点には注意が必要である．また，過剰に細かく見積もりを行うことを避けるために，Story Point は連続した値ではなく，フィボナッチ数の何れかのみを値として持たせる場合が多い．

具体的な手順は以下のとおりである．

1. 開発メンバー全員で予測対象のユーザーストーリーに関して議論を行う．
2. 開発メンバーそれぞれが予測対象のユーザーストーリーに割り当てる Story Point の見積もりを行う．
3. それぞれの見積もり結果を同時にお互いに提示する．
4. 最大値，最小値の Story Point の見積もりを行ったメンバーがその見積もりを行った理由についてメンバー全員に説明する．
5. ユーザーストーリーの見積もり値について開発チームの全員の合意が得られるまで上記の手順を繰り返す．

問題点　Planning Poker のように人間の主観に基づいて見積もりを行う場合，その予測は過小見積もりになる傾向があると言われており，その傾向は集団で議論を行って見積もりをする場合，その傾向はより顕著になると言われている [9]．また，Planning Poker による見積もりは，開発者の経験にその精度が依存するという報告もある [10]．そのため，人間の主観に基づかない，自動的な見積もり手法が必要とされている．

2.3　アジャイルソフトウェア開発プロジェクトにおける自動的な見積もりの既存研究

アジャイルソフトウェア開発プロジェクトにおける自動的な見積もりを行う手法の既存研究としては，Abrahamsson らの研究 [20] がある．Abrahamsson らの研究では, ユーザーストーリーの記述の中に出現する特定の単語 (キーワード) の情報等を用いて，そのユーザーストーリーの実装にかかる時間を見積もる手法を提案している．

また，他の既存研究として，Porru らの研究 [9] もある．Porru らの研究については 2.3.1 節で詳細を説明する．

2.3.1 Issue Report を用いたタスクの Story Point の見積もり手法

アジャイルソフトウェア開発では開発中に行うタスクの内容を Issue Report に記述して管理をする場合が多い．また，開発者はそのタスクについて見積もった Story Point の値もその Issue Report に対して記述して管理している．

- Issue Report

  課題追跡システム (Issue Tracking System) と呼ばれるシステム (例：JIRA[21], Redmine[22], Trac[23]) で課題を管理するために用いられるもの．Porru らの研究 [9] では JIRA が用いられている．

- タスク

  ユーザーストーリーをさらに細かい粒度の作業に分割したもの．分割することが可能な最小単位の粒度の作業を意味する．

- Story Point

  2.2.1 節の Story Point の項目を参照

Porru らの研究 [9] では，Issue Report の記述内容からそのタスクの Story Point の値を予測する手法を提案している．1 つの実企業開発プロジェクトと 8 つのオープンソース開発プロジェクトに手法を適用し，開発者と同程度の精度で予測可能であることが示されている．

図 2 は手法の概略図である．

具体的な手法の手順について説明する．

1. まず，Issue Report が蓄積されている課題追跡システム等のデータベースから Issue Report を取得する．
2. 取得した Issue Report に記述されているタスクの特徴を抽出し，抽出したタスクの特徴を示す特徴ベクトルに変換する
3. 特徴ベクトルと Issue Report に記述されている Story Point を学習データとして機械学習を適用し予測機を構築する．
4. 構築した予測機に予測対象になる新規 Issue Report を適用し Story Point を予測する．

また，タスクの特徴抽出に関して，Porru らの研究では JIRA の Issue Report の属性の内，「概要」「詳細」「コンポーネント」「種類」の 4 つの属性からタスクの特徴を抽出する．これらの属性の説明と属性からタスクの特徴を抽出する方法については表 1 に示す．

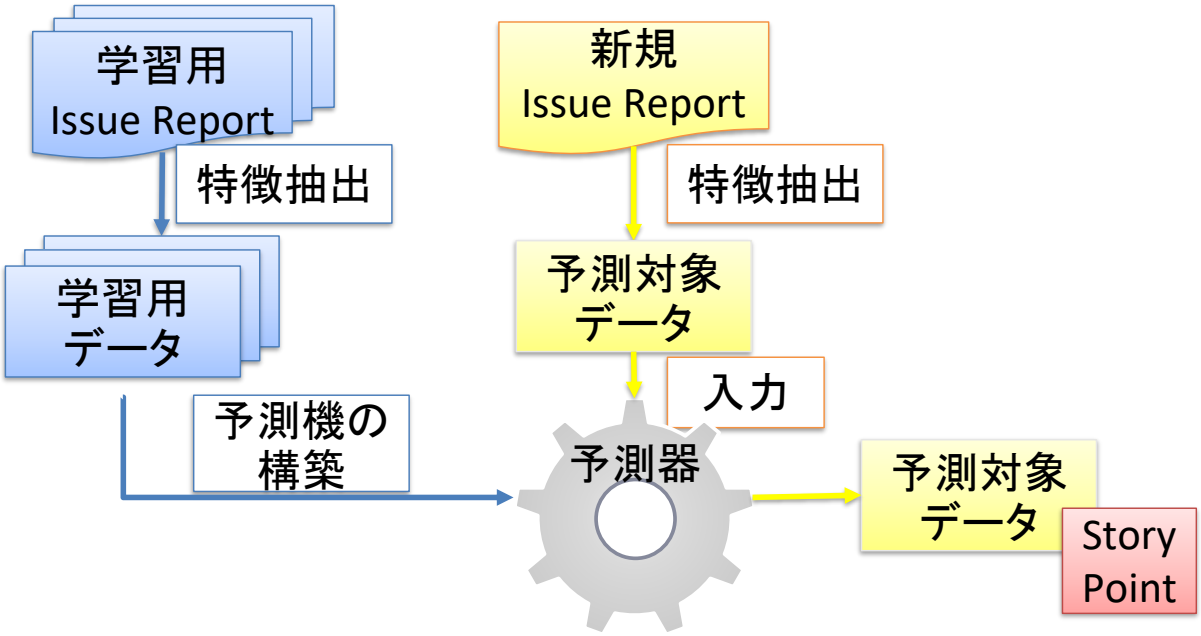

図 2: Porru らの手法の概略図

表 1: Porru らの研究で使用する Issue Report の属性

属性名	属性の説明	特徴抽出方法
概要	タスクの内容が記述された文章	自然言語解析
詳細	タスクの内容が記述された文章	自然言語解析
種類	タスクの作業の種類を示す値	ダミー変数化
コンポーネント	タスクに関連するファイル名	ダミー変数化

「概要」「詳細」等の文章情報については，自然言語解析により，文章に出現する単語を用いた特徴ベクトルに変換される．自然言語解析による特徴ベクトル化については 2.5 節で説明する．

「種類」「コンポーネント」等の名義尺度で表される属性については，ダミー変数化を行い，特徴ベクトルに変換する．ダミー変数化については 2.6 節で説明する．

2.4 機械学習

本研究において，機械学習はタスクの内容を記述した文書やタスクの属性から抽出したタスクの特徴を入力とし，そのタスクの作業時間を予測を行う手法として用いる．Porru らの研究 [9] では予測対象が離散値である「Story Point」であったため，教師あり学習で分類問題を解くためのアルゴリズムを用いていた．しかし，本研究の予測対象は連続値である「作業時間」を対象とするため，教師あり学習で分類問題を解くアルゴリズムを用いる．実際に本研究で使用している機械学習のアルゴリズムを以下の 2.4.1 節と 2.4.2 節で説明する．

2.4.1 Support Vector Regression

Support Vector Regression(SVR) は教師あり学習の分類問題に広く用いられるアルゴリズム Support Vector Machine(SVM) から派生したアルゴリズムである．SVM が分類問題を解くアルゴリズムであるのに対し，SVR は回帰問題を解くこともできる．

2.4.2 K 近傍法

K 近傍法 (KNN) は機械学習のアルゴリズムの 1 つであり，とても単純なアルゴリズムであるため，広く用いられている．KNN のアルゴリズムでは，まず，データを特徴ベクトルで表し，予測対象のデータと学習用のデータの距離を計算する．予測対象のデータ近くにある学習用のデータの上位 k 件を用いて，予測対象の分類を行う．回帰問題を解く場合は，上位 k 件のデータの値の平均値や，ベクトル同士の距離の近さに基づいた加重平均の値を予測値とする．この回帰問題を解く K 近傍法のアルゴリズムを基にした開発工数の見積もり手法として類似性に基づく手法 [24] 等が提案されている．

また，ベクトル同士の距離の計算は本研究ではユークリッド距離の逆数を用いて行う．例えばあるデータのベクトル p_a と p_b の距離をユークリッド距離で表すと式 1 になる．ここで，v_{aj} と v_{bj} はそれぞれベクトル p_a と p_b の特徴量を表す．本研究の例でいうと，ベクトル p_a がタスクの特徴を表すベクトルになり，v_{aj} がタスクの内容を記述した文書やタスクの属性から抽出したタスクの特徴の 1 つを表す．

$$dist\left(p_a, p_b\right) = \sqrt{\sum_{j=1}^{n}\left(v'_{aj} - v'_{bj}\right)^2} \tag{1}$$

また，このベクトル間の距離のことを「類似度」と呼ぶ．

2.5 自然言語解析

自然言語解析は人間が用いる自然言語をコンピュータで理解できるようにするために用いる技術である．本研究において，自然言語解析はタスクの内容について書かれた文章を出現する単語を用いて表される特徴ベクトルに変換するために用いる．自然言語解析は，初めに文章を意味を持つ最小単位の単語に分割する形態素解析を行う．形態素解析については 2.5.1 節で説明する．

この時得られたそれぞれの単語に重みづけをすることにより機械学習に用いる特徴ベクトルにする．単語に重みづけをする手法には TF-IDF 法を用いている．TF-IDF 法については 2.5.2 で説明する．

2.5.1 形態素解析

形態素解析とは，自然言語で記述された文章を意味を持つ最小単位に分割しそれぞれの品詞を判別する技術である．形態素解析ツールを用いることでドキュメント内に出現する固有表現の大半を判別することが可能となる [25]．

本研究においては，日本語で書かれた文書情報から 2.5.2 節の TFIDF 法を適用する際に，日本語の文書を単語に分割する目的で形態素解析を使用している．使用している形態素解析ツールは Kuromoji[26] である．

また，本研究が対象としたプロジェクトでは，Camel 形式で書かれたクラス名などが日本語の文書情報として出現する．Camel 形式で書かれた英語の文章は Kuromoji では単語に分割することが出来ない．そのため，英数字で構成される Camel 形式の単語が出現した場合は，Camel 形式の単位でさらに単語に分割する．

例えば，「SearchingForm の作成」という文書が出現した場合，自然言語解析の結果，図 3 のようになる．

因みに，Porru らの研究 [9] では，分割した単語を特徴ベクトルにする際に，1 単語だけでなく，連続する 2 つの単語の組み合わせも特徴として加えているが，本研究では 1 単語のみで特徴ベクトルを構成している．

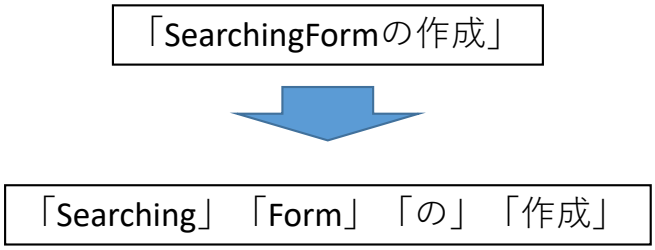

図 3: 自然言語解析による単語分割の例

2.5.2 TFIDF 法

TFIDF 法とはテキストマイニングの分野などで用いられる文書中の単語に関する重みづけを行う手法の一つである．TFIDF 法は文書中の出現頻度が高い単語の特徴量を大きくし，複数の文書に出現する単語の特徴量を下げる重みづけを行う．

重みの計算式は式 4 になる．

$$\text{単語}_i\text{の重み} = TF_i \times IDF_i \tag{2}$$

$$TF_i = \frac{\text{文書中のある単語}_i\text{の出現回数}}{\text{文書に出現する総単語数}} \tag{3}$$

$$IDF_i = \log \frac{\text{総文書数}}{\text{単語}_i\text{が出現する文書数}} \tag{4}$$

例えば，「Event 作成」という文書 A と「Form 作成」という文書 B の 2 つに対して TFIDF 法による重みづけをを行うとする．文書 A と文書 B は形態素解析の結果「”Event”，”作成”」と「”Form”，”作成”」という単語に分割される．文書 A の TF の計算について考えると，文書 A には 2 単語含まれており，それぞれの単語の出現回数は 1 回ずつである．そのため，文書 A の”Event”と”作成”の TF の値は $TF_i = \frac{1}{2} = 0.5$ となる．同様に文書 B の”Form”と”作成”の TF の値も $TF_i = \frac{1}{2} = 0.5$ となる．

次に，文書 A の IDF の計算についてであるが，まず，総文書数は文書 A と文書 B の 2 つである．”Event”という単語は文書 A にしか出現しないため，$IDF_{\text{”}Event\text{”}} = \log \frac{2}{1} = 0.68$ となる．同様に”Form”という単語も文書 B にしか出現しないため，$IDF_{\text{”}Form\text{”}} = \log \frac{2}{1} = 0.68$ となる．”作成”という単語は文書 A にも文書 B にも出現する．そのため，$IDF_{\text{”作成”}} = \log \frac{2}{2} = 0$ となる．

最終的に，文書 A と文書 B を TFIDF 法で重みづけすると図 4 のような変換が行われることになる．

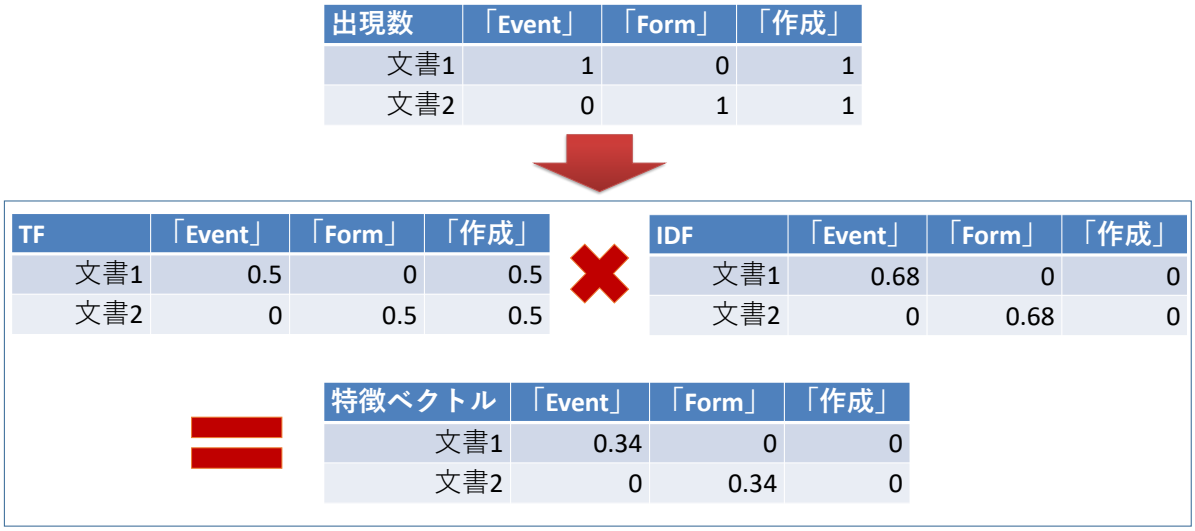

図 4: TFIDF 法による重みづけの例

2.6 ダミー変数化

ダミー変数化は名義尺度のメトリクスを他の比例尺度のメトリクスと同様に扱えるようにするため，カテゴリごとにダミー変数に置き換える手法である．プロジェクト p_i が名義尺度のメトリクス m_j においてカテゴリ c に属するかどうかを表すダミー変数 $d_{ij}(c)$ は式 (5) で定義される．

$$d_{ij}(c) = \begin{cases} 1 & \text{...カテゴリ } c \text{ に属する} \\ 0 & \text{...カテゴリ } c \text{ に属さない} \end{cases} \tag{5}$$

具体的には図 5 のように変換が行われる．図は”作成 (ソースコード)””作成 (単体テスト)””レビュー”の 3 種類のカテゴリーを持つ属性「種類」をダミー変数化した例である．ダミー変数化の結果，1 つの名義尺度の属性だった「種類」は 2 つの比例尺度の属性「種類＝‘作成 (ソースコード)’」「種類＝‘作成 (単体テスト)’」に変換される．

種類
作成(ソースコード)
作成(単体テスト)
レビュー
作成(ソースコード)

種類=‘作成(ソースコード)’	種類=‘作成(単体テスト)’
1	0
0	1
0	0
1	0

図 5: ダミー変数化の例

3　本研究で行った調査の報告

3.1　調査内容

本研究では，アジャイルソフトウェア開発を取り入れたソフトウェア開発 PBL のプロジェクトにおけるタスクの作業時間の自動的な見積もりを行う手法の調査を行う．そのために以下の項目について調査を行った．

RQ1: 機械学習に基づく手法は有効であるか？

RQ2: 過去に実施された **PBL** のプロジェクトデータは予測精度を向上させるか？

RQ3: タスクの種類であらかじめ層別したデータセットを構築することで予測の精度を向上させることが出来るか？

また，それぞれの調査では，主に以下の観点で調査を行う．

- 開発者の予測と同じ程度の精度で予測可能であるか
- 前の調査よりも手法の精度は向上はしているか？

3.2　調査に用いるソフトウェア開発 PBL プロジェクトの概要

大学院博士前期課程 1 年の学生を対象としたソフトウェア開発 PBL[27] で実施された過去 4 年間のプロジェクトのデータを用いる．

ソフトウェア開発 PBL で学生が取り組むプロジェクトの概要は以下のとおりである．

- 開発するソフトウェア：Web 上でチケットの購入予約をするシステム
- 開発手法：アジャイルソフトウェア開発

より具体的には，Scrum 開発手法を実践する．更に，チケット駆動開発という手法も取り入れている．Scrum 開発手法については 3.2.1 節で，チケット駆動開発については 3.2.2 節で説明する．

- 開発期間：5 日間

  1 日を 1 イテレーションと考えて開発に取り組む．

- 開発人数：5〜6 人
- 利用したサーバー：Tomcat[28]
- 利用したフレームワーク:Jersey[29]
- 利用したデータベース:mongoDB[30]

2.2 節で説明したように，ソフトウェア開発 PBL のプロジェクトにおいても，イテレーションの初めにはそのイテレーションで開発する作業の計画を行う．具体的には以下の手順でそのイテレーションで開発する作業の計画を行う．

1. そのイテレーションで取り組むユーザーストーリーを選択する
2. 選択したユーザーストーリーを更に粒度の細かいタスクへと細分化する
3. 細分化したタスクの内容を Issue Report に記入する
4. タスクの作業にかかる時間の予測も併せて記入する

また，ソフトウェア開発 PBL のプロジェクトに参加している開発者は，開発中に発生するあらゆるタスクを Issue Report に記入してそのタスクを管理することになっている．そのため，プロジェクト中に開発者が取り組んだタスクは全て Issue Report の形で課題追跡システムのデータベース上に蓄積されることになる．本研究では，その蓄積された Issue Report のデータを利用して調査を行う．

調査に用いる Issue Report には以下の表 2 に示した属性が含まれている．

Issue Report の属性に記入される内容の例として，図 6 のような内容が記入されている．

ちなみに，対象とするソフトウェア開発 PBL のプロジェクトでは課題追跡システムとして Trac[23] を用いている．

表 2: Issue Report の属性

属性名	説明
概要	タスクの内容が記述された文章
種類	タスクの作業の種類を示す値
コンポーネント	タスクに関連するファイル名
イテレーション	実施したイテレーションの名前
担当者	担当した開発者の名前
予測時間	タスクの作業時間の予測値
作業時間	実際にかかった作業時間

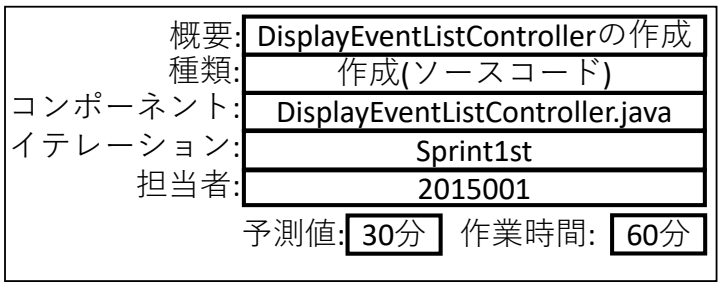

図 6: Issue Report の記入内容の例

3.2.1 Scrum 開発

Scrum 開発はアジャイルソフトウェア開発手法のフレームワークの 1 つである．Scrum は経験主義に基づき，反復的かつ漸進的な手法を用いて，予測可能性の最適化とリスク管理を行うことを目標としたプロセスフレームワークである [31]．そのフレームワークには，プロジェクトの見える化や見える化に基づく現状の検査および適応といったいくつかの重要なコンセプトがまとめられている [11]．

3.2.2 チケット駆動開発

チケット駆動開発はプログラム開発手法の 1 つである．チケット駆動開発ではソフトウェアの開発中に発生する全てのタスクを課題追跡システムの Issue Report に割り当てて管理する．また，ソースコード等の成果物に変更がある場合，それらの変更は Issue Report に関連付けされて管理される．チケット駆動開発の利点としては，まず，課題追跡システム上の Issue Report を確認することで，開発中に発生する全てのタスクを管理することが出来るため，プロジェクトの進捗状況が把握しやすいという点が挙げられる．

また，チケット駆動開発では版管理システムを用いてソースコードを管理している．そして，ソースコードの変更は必ず Issue Report に割り当てて管理されている．つまり，開発者がソースコードを変更した意図は Issue Report で確認することが出来るようになっている．そのため，開発者が行う版管理システムに対するコミットの単位が明確になる，つまり，複数の意図をはらんだコミットが行われにくいという利点もある．

3.3 RQ1：機械学習に基づく手法は有効であるか？

動機　ソフトウェア開発 PBL のプロジェクトは実世界で行われるソフトウェア開発プロジェクトと比べ，一般的に開発期間が短く，また，開発するソフトウェアの仕様も明確であるため，Issue Report の記述が非常に簡潔であるという特徴がある．そのため，実世界で行われるソフトウェア開発プロジェクトと比べ，自然言語解析を用いて抽出できる特徴が少ない．また，Porru らの研究 [9] では予測対象は離散値である「StoryPoint」であったが，本研究では連続値である「作業時間」が予測対象となる．それらの影響で，機械学習に基づく手法が Porru らの研究で報告されていた程上手くいかない可能性があると考えられる．

そこで，本研究が対象とするプロジェクトにおいて，機械学習に基づく手法がどの程度有効であるかを調査する．

アプローチ　Porru らの研究 [9] の手法 (2.3.1 節参照) をソフトウェア開発 PBL で行われた 31 のプロジェクトに対して適用する．使用する機械学習手法は表 3 に示す．SVR を採用す

る理由としては，Porru らの研究で最も良い精度を示した機械学習の手法だったからである．KNN を採用する理由としては，本研究が対象とするソフトウェア開発 PBL のプロジェクトの示す特徴に適した手法であると考えたからである．ソフトウェア開発 PBL のプロジェクトは Issue Report から抽出できる特徴が少ないと考えられる．また，予測対象も離散値よりも予測が難しい連続値である「作業時間」になるため，複雑なアルゴリズムでは上手く相関が見つけられない可能性が高いと思われる．さらに，ソフトウェア開発 PBL のプロジェクトでは Issue Report の記述内容が類似しているタスクは似たような「作業時間」の値を示す傾向がある．これらの要因より，ソフトウェア開発 PBL のプロジェクトの作業時間の見積もりに機械学習を用いるのであれば，特徴ベクトルが類似しているデータの加重平均の値を予測値とする，非常に単純なアルゴリズムである KNN に適していると考えた．

表 3: 採用した機械学習

機械学習手法	採用理由
SVR(Support Vector Regression)	Porru らの研究 [9] で最も精度が良かった
KNN(K 近傍法)	ソフトウェア開発 PBL のプロジェクトの示す特徴は K 近傍法に適していると考えられるため

また，Porru らの研究 [9] の手法を適用するにあたり，以下の 2 点の変更を行った．

- 機械学習手法に関する変更点

  Porru らの研究 [9] の手法では予測対象が離散値である「Story Point」であった．しかし，本研究の予測対象は離散値ではない「作業時間」に代わるため，機械学習の手法はクラス分類の手法ではなく，回帰分析の手法を用いる．尚，RQ1 の調査において機械学習手法の実装は weka[32] に実装されているアルゴリズムを使用している．weka はニュージーランドのワイカト大学で開発された機械学習ソフトウェアである．

- 自然言語解析手法に関する変更点

  Porru らの研究とは違い，本研究で対象とする文書は日本語で記述されているため，形態素解析の方法に幾つか変更を行った．本研究における自然言語解析手法の詳細は 2.5 節に記載する．

3.3.1 実験手順

実験は Porru らの研究 [9] と同様に 10 分割交差検証法を用いて行う．10 分割交差検証はデータセットを 10 個のサブセットに分割し，そのうちの 1 つをテストデータに，残りのサブセットを学習データに分割して予測手法を適用する実験方法である．具体的な手順は以下のとおりである．

1. Issue Report の内容からタスクの特徴を抽出してプロジェクトのデータセットを構築する．
2. データセットを 10 個のサブセットに分割する．
3. 10 個のサブセットからテストデータセットとして 1 つのサブセットを選ぶ
4. 残りの 9 個のサブセットを学習データとして機械学習手法を適用し，回帰式を導出する．
5. 回帰式からテストデータの予測値を算出する．
6. 算出した予測値と Issue Report に記載されている実測値から誤差を計測する．誤差の尺度は MRE(Magnitude of Relataive Error)[33] を用いる．
7. 10 個のサブセット全てをテストデータとするまで手順 3〜手順 6 を繰り返す．これでデータセットに含まれる全データ分の誤差が得られる．
8. 得られた全データ分の誤差から手法の予測精度を算出する．予測精度の尺度には MMRE を用いる．

実験に使用する Issue Report の条件　今回対象とする PBL のプロジェクトにおいて，実験に使用する Issue Report は幾つかの条件を満たすものに限定した．

- タスクの「種類」に関する制約

  PBL のプロジェクトではタスクの種類として「作成 (ソースコード)」「作成 (単体テスト)」「作成 (結合テスト)」「レビュー」「バグ修正 (ソースコード)」「バグ修正 (単体テスト)」「バグ修正 (結合テスト)」「結合テスト」「その他」のカテゴリーが存在する．しかし，今回の手法では，イテレーションの初めの計画段階で行う見積もりを想定しており，その時点で開発者が主に見積もりを行うことになるタスクは「作成 (ソースコード)」「作成 (単体テスト)」「レビュー」の 3 種類である．そのため，実験においては，この 3 種類のタスクで構成されたデータセットを使用する．

- Issue Report の記入漏れに関する制約

  Issue Report の中には，「作業時間」の記入漏れがあるものが存在する．これらに関しては，開発者が Issue Report のタスクに取り組み始めた時刻「開始時刻」と Issue Report のタスクを完了した時刻「終了時刻」を用いて修正する．「開始時刻」「終了時刻」は Issue Report の記入項目の中に含まれている．「作業時間」の記入漏れがあり，かつ「開始時刻」「終了時刻」に記入漏れや不備がある場合はその Issue Report はデータセットから除外する．「開始時刻」「終了時刻」の不備とは，プロジェクトが行われていた期間外の時刻が記入されているといった明らかな不備のある記入のことである．また，「作業時間」の記入漏れがあり，かつ「開始時刻」「終了時刻」がイテレーションを跨いでいる場合，その Issue Report も除外する．

  また，Issue Report の中には「種類」の項目の入力を明らかに間違えているものが存在する．例えば，「概要」の記述に”バグ修正”と書いているにも関わらず，「種類」のカテゴリーが”作成 (ソースコード)”である場合などである．そのような「種類」の項目の入力を明らかに間違えている場合には，「概要」の記述に適したカテゴリーに「種類」の項目を修正する．

- Issue Report のタスクの粒度に関する制約

  プロジェクトによっては，細分化されたタスクの粒度が極端に細かいプロジェクトが存在する．主に 2 種類の細かすぎるタスクの粒度の例がある．1 つ目は「あるクラスのソースコードを作成するタスク」が，「そのクラスの属性や関数の定義のみを行うタスク」と，「関数の実装を行うタスク」に分割されている場合である．そのようなタスクの分割方法が間違っているわけではないが，今回対象としたプロジェクトのほとんどは，「そのクラスの属性や関数の定義のみを行うタスク」で Issue Report を作成していないため，今回は「あるクラスのソースコードを作成するタスク」が「そのクラスの属性や関数の定義のみを行うタスク」と，「関数の実装を行うタスク」に分割されているような場合には，作業時間が短かった方の Issue Report をデータセットから除外した．

  2 つ目は Issue Report を「あるクラスのソースコードを作成するタスク」が，「そのクラスのある 1 つの関数の実装」単位で分割されている場合である．そのようなタスクの分割方法が間違っているわけではないが，今回対象としたプロジェクトのほとんどは，「そのクラスのある 1 つの関数の実装」単位で Issue Report を作成していないため，そのような分割を行っているプロジェクトは実験の対象から外した．

MRE(Magnitude of Relataive Error)[33]　MRE は実測値から見た相対誤差を表す誤差尺度である．MRE を計算する式は以下の式 6 になる．

$$MRE = \frac{\left|\text{実測値} - \text{予測値}\right|}{\text{実測値}} \tag{6}$$

MRE は，Porru らの研究 [9] の誤差尺度として用いられており，条件を揃えるために本実験でも同じ尺度を用いる．

また精度を評価する際は，MMRE を用いる．MMRE は全てのデータの誤差 MRE を平均したものである．MMRE の計算式は以下の式 7 になる．

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \tag{7}$$

3.3.2　結果

31 プロジェクトに対して適用した実験の結果が表 4 である．また，実験の結果を箱ひげ図で示したものが図 7 と図 8 である．箱ひげ図は箱ひげ図は一番上の横線から順に「最大値」「第 3 四分位数」「中央値」「第 1 四分位数」「最小値」を表している．横線の外にある丸は「外れ値」を表している．箱ひげ図は統計ソフト R[34][35] を用いて作成している．まず，SVR と KNN の比較について，表 4 より，31 プロジェクトの内，26 のプロジェクトにおいて，KNN の方が良い精度を示しており，また，図 7 を見ると，SVR は外れ値を示すプロジェクトが多いことが分かる．外れ値を除いた図 8 を見ると KNN の方が良い精度を示す傾向があることが分かる．次に，KNN と開発者の予測を比較すると，表 4 より，ほぼ全てのプロジェクトにおいて開発者の予測の方が精度が良いことが分かる．

結果として，機械学習の手法としては，SVR よりも KNN の方が有効であるが，開発者の予測と同程度の精度ではなかった．

3.3.3　考察

SVR より KNN が有効であった理由は，やはり KNN の方がソフトウェア開発 PBL のプロジェクトの示す特徴に適していたということが考えられる．SVR よりも KNN の方が有効であったため，今後の調査では機械学習手法として KNN の手法を用いることにする．また，開発者の予測よりも精度が悪い理由としては，やはり機械学習に基づく手法をそのまま使用するだけでは，ソフトウェア開発 PBL のプロジェクトで作業時間を見積もる場合には，良い精度は得られなかったということが考えられる．そのため，PBL のプロジェクトに対しても良い精度が得られるように機械学習に基づく手法を改善する必要がある．

表 4: 31 プロジェクトの MMRE の値の比較

プロジェクト名	SVR	KNN	開発者の予測
2016G1	152%	128%	170%
2016G2	442%	218%	109%
2016G3	276%	268%	82%
2016G8	387%	422%	126%
2016G9	257%	187%	68%
2015G1	166%	148%	133%
2015G2	250%	218%	176%
2015G3	235%	163%	104%
2015G4	192%	315%	276%
2015G5	148%	124%	70%
2015G6	256%	158%	79%
2015G7	277%	296%	171%
2015G8	268%	186%	155%
2014G1	237%	193%	86%
2014G2	287%	245%	205%
2014G3	219%	216%	128%
2014G4	186%	132%	75%
2014G5	435%	366%	60%
2014G6	142%	160%	141%
2014G7	212%	377%	149%
2014G8	230%	187%	84%
2014G9	992%	432%	81%
2013G1	283%	249%	127%
2013G2	768%	294%	77%
2013G3	148%	132%	112%
2013G4	170%	153%	78%
2013G5	167%	150%	79%
2013G6	186%	138%	84%
2013G7	1605%	938%	84%
2013G8	252%	251%	99%
2013G9	335%	250%	63%

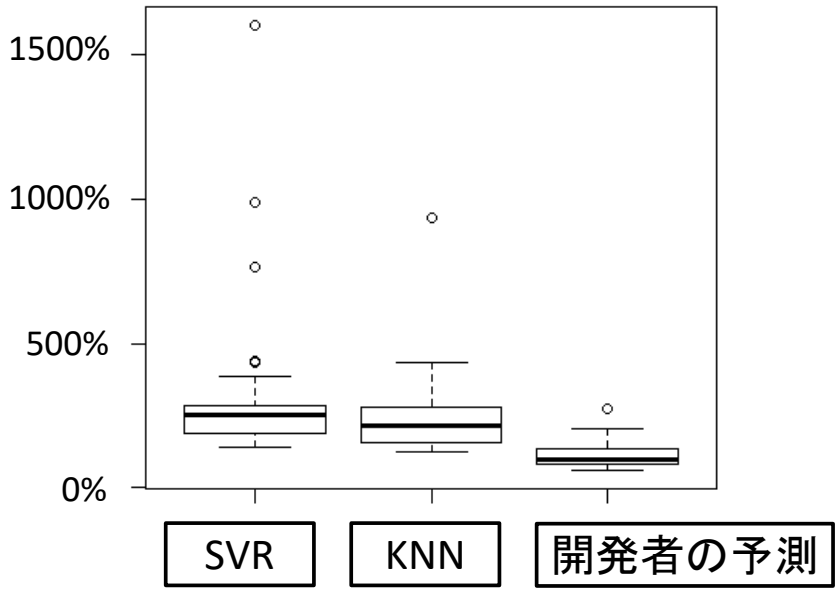

図 7: 31 プロジェクトの MMRE の値を比較した箱ひげ図

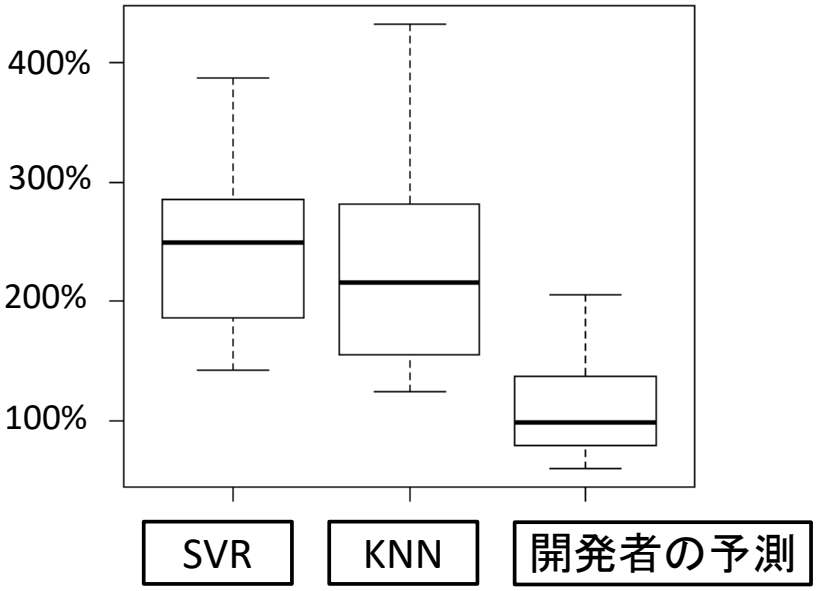

図 8: 31 プロジェクトの MMRE の値を比較した箱ひげ図 (外れ値なし)

3.4 RQ2：過去に実施された PBL のプロジェクトデータは予測精度を向上させるか？

動機 RQ1 で適用した手法では，機械学習に与える学習用のデータとして，同一プロジェクトのデータのみを利用しており，過去に実施された他のプロジェクトのデータは利用していない．対象としているソフトウェア開発 PBL はこれまでの過去 4 年間実施されており，その時に実施されたプロジェクトのデータが蓄積されている．

そこで，過去に実施されたソフトウェア開発 PBL のプロジェクトのデータを用いることにより，予測の精度を向上させることが出来るかを調査する．

アプローチ 同一プロジェクト内の Issue Report のみを用いた実験と過去に実施された PBL のプロジェクトの Issue Report を用いた実験の 2 種類をソフトウェア開発 PBL で行われた 22 プロジェクトに対して適用し，その結果を比較する．2 種類の実験は RQ1 で行った実験に基づいているが，テストデータと学習データの生成方法について変更した点があるため，改めて説明を行う．

3.4.1 実験 1：同一プロジェクト内の Issue Report のみを用いた実験

1. Issue Report の内容からタスクの特徴を抽出してプロジェクトのデータセットを構築する．
2. データセットを Issue Report が完了した時刻順にソートし，イテレーション単位でサブセットに分割する．
3. まず，データセットから 1 回目のイテレーションのデータを学習データにし，2 回目のイテレーションをテストデータにする．
4. テストデータの予測値として，学習データの中でテストデータに類似した上位 k 件のデータの類似度に基づいた加重平均を算出する．
5. 算出した予測値と Issue Report に記載されている実測値から誤差を計測する．誤差の尺度は MRE を用いる．
6. 次に，先ほどの学習データにテストデータとして使用したイテレーションのデータを加える．
7. 先ほどまでテストデータだったイテレーションの次のイテレーションのデータをテストデータにする．
8. テストデータの予測値として，学習データの中でテストデータに類似した上位 k 件のデータの類似度に基づいた加重平均を算出する．

9. 算出した予測値と Issue Report に記載されている実測値から誤差を計測する．

10. 手順 6〜手順 9 を最後のイテレーションをテストデータにするまで繰り返す．

11. 得られた誤差から手法の予測精度を算出する．予測精度の尺度には MMRE を用いる．

図 9 は実験の手順を示した概略図である．図 9 は 3 つ目のイテレーション 3rd をテストデータとしたときの例であるが，この時の学習データはそれまでのイテレーション，つまり，1 つ目と 2 つ目のイテレーション 1st,2nd になる．これらの学習データを用いてテストデータを予測することで，3rd のデータ関して誤差を計算することが出来る．そして，これを繰り返すことにより得られる誤差を用いることでこのプロジェクトに対する実験 1 の手法の予測精度を算出することが出来る．

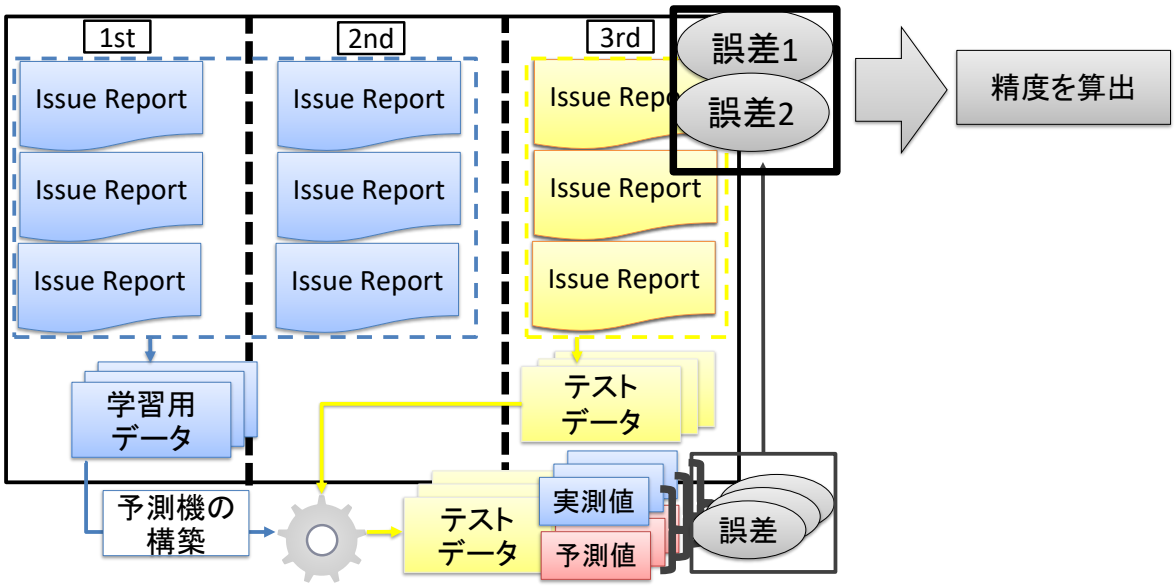

図 9: 実験 1 の概略図

変更点：テストデータと学習データの生成方法　RQ1 の実験では 10 分割交差検証を用いていたが，この方法は Issue Report が完了した時系列について考慮されていない．そこで本実験では，実際のプロジェクトの時系列を考慮し，データセットをイテレーション単位で分割する上記の生成方法を用いた実験を行った．

3.4.2 実験 2：過去に実施された PBL のプロジェクトの Issue Report を用いた実験

1. 予測対象となるプロジェクトとその前年度までに実施された全てのプロジェクトの Issue Report の内容からタスクの特徴を抽出してデータセットを構築する．
2. 予測対象となるプロジェクトに含まれていたデータをテストデータとし，その前年度までに実施された全てのプロジェクトのデータを学習データとする．
3. テストデータの予測値として，学習データの中でテストデータに類似した上位 k 件のデータの類似度に基づいた加重平均を算出する．
4. 算出した予測値と Issue Report に記載されている実測値から誤差を計測する．
5. 得られた誤差から手法の予測精度を算出する．予測精度の尺度には MMRE を用いる．

図 10 は実験の手順を示した概略図である．図 10 は 2015 年に実施されたプロジェクトをテストデータとしたときの例であるが，この時の学習データはその前年までに実施されたプロジェクト，つまり，2013 年〜2014 年に実施された全てのプロジェクトとなる．これらの学習データを用いてテストデータを予測することで，2015 年に実施されたプロジェクトのデータ関して誤差を計算することが出来る．これにより得られる誤差を用いることでこのプロジェクトに対する実験 2 の手法の予測精度を算出することが出来る．

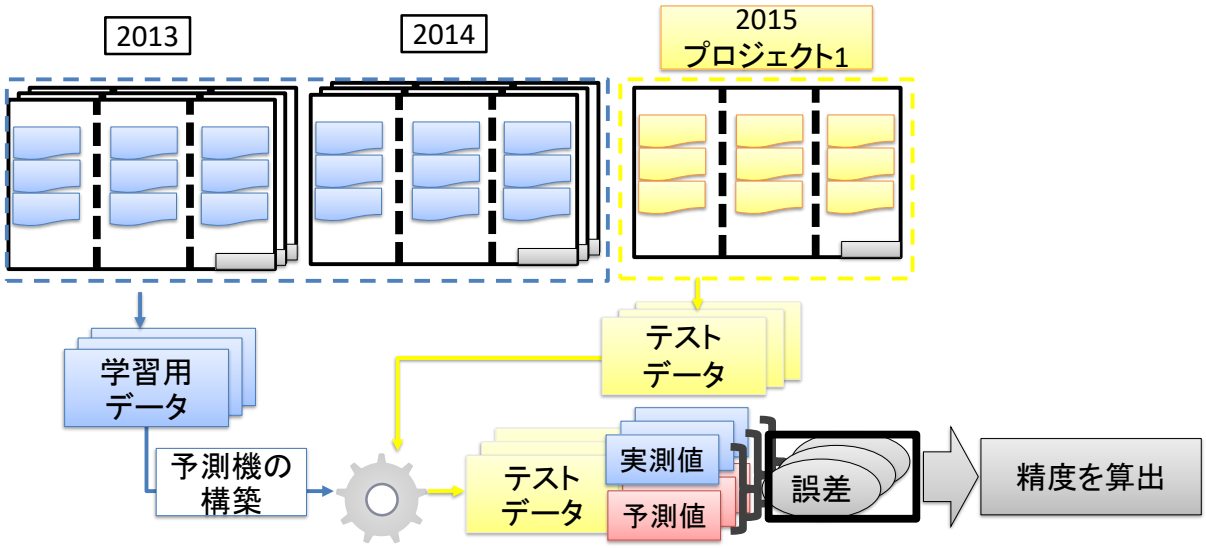

図 10: 実験 2 の概略図

テストデータと学習データの生成方法：実施年度で分割　本実験では，テストデータと学習データの生成方法として，ある年のプロジェクトのデータを予測対象のテストデータとした場合，その前年の PBL の全プロジェクトのデータを学習データとする生成方法を用いる．

3.4.3　結果

22 プロジェクトに対して適用した実験の結果が表 5 である．また，実験の結果を箱ひげ図で示したものが図 11 である．また，実験において類似している上位何件のデータを予測に使用するかを示す k の値についてであるが，表 5 及び図 11 では，k の値 1〜5 までの中で最も精度が良かった手法の結果を示している．

まず，実験 1 と実験 2 の比較について，表 5 より，22 プロジェクトの内，16 のプロジェクトにおいて，実験 2 の方が良い精度を示しており，また，図 11 を見ても，実験 2 の方が良い精度を示していることが分かる．次に，実験 2 と開発者の比較について，表 5 より，ほぼ全てのプロジェクトにおいて開発者の予測の方が精度が良いことが分かる．

結果として，過去に実施されたソフトウェア開発 PBL のプロジェクトのデータを用いる方が精度が向上するが，開発者の予測と同程度の精度ではなかった．

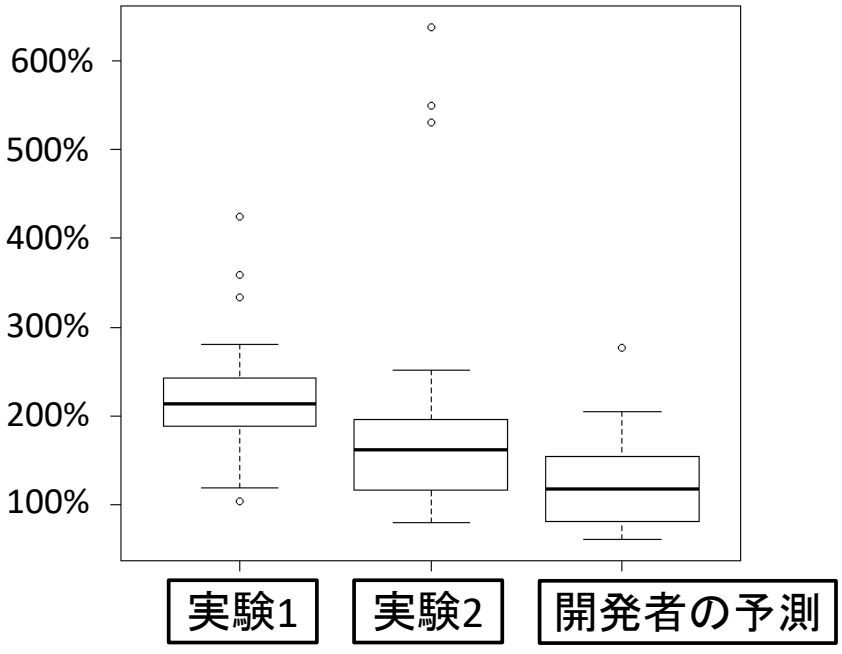

図 11: 22 プロジェクトの MMRE の値を比較した箱ひげ図

表 5: 22 プロジェクトの MMRE の値の比較

プロジェクト名	実験 1	実験 2	開発者の予測
2016G1	103%	150%	170%
2016G2	162%	550%	109%
2016G3	171%	128%	82%
2016G8	359%	639%	126%
2016G9	204%	173%	68%
2015G1	118%	175%	133%
2015G2	243%	179%	176%
2015G3	188%	114%	104%
2015G4	197%	146%	276%
2015G5	142%	95%	70%
2015G6	207%	79%	79%
2015G7	218%	251%	171%
2015G8	280%	196%	155%
2014G1	334%	193%	86%
2014G2	232%	178%	205%
2014G3	212%	90%	128%
2014G4	216%	205%	75%
2014G5	241%	86%	60%
2014G6	208%	135%	141%
2014G7	424%	531%	149%
2014G8	251%	131%	84%
2014G9	217%	116%	81%

3.4.4 考察

過去に実施したソフトウェア開発 PBL のプロジェクトの Issue Report を用いる実験 2 の方が同一プロジェクト内の Issue Report のみを用いる実験 1 よりも精度が良い理由としては，やはり，過去に実施されたソフトウェア開発 PBL のプロジェクトのデータは精度の向上に有効であったと考えられる．

次に，開発者の予測よりは精度が悪い理由の考察についてであるが，まず，過去に実施したソフトウェア開発 PBL のプロジェクトの Issue Report を用いる利点としては，過去に実施したソフトウェア開発 PBL のプロジェクトの中には，予測対象と類似したタスクが必ず存在している点であると考えられる．しかし，Issue Report の書き方はプロジェクトによって違いがあり，記述内容が似ていてもタスクの種類が異なる場合が見られた．そのため，KNN において，類似した Issue Report を適切に検索できていない可能性があると考えられる．

3.5 RQ3：タスクの作業種類によるデータの分類は予測精度を向上させるか？

動機 タスクの種類は予測において重要な特徴であることが Porru らの研究 [9] でも述べられている．

そこでタスクの種類であらかじめ層別したデータセットを構築することにより精度が向上するかどうかを調査する．

アプローチ 以下の表 6 のように層別したデータセットを構築して RQ2 の実験を再度適用する．また，本実験では誤差の尺度に関して今まで使用してきた MRE に加えて BRE(Balanced Relative Error)[36] の尺度も用いる．

表 6: タスクの種類で層別したデータセット

データセット	含むタスクの種類	採用理由
基本タスク	作成 (ソースコード) 作成 (単体テスト) レビュー	今までの実験で用いていたもの
コーディング	作成 (ソースコード) 作成 (単体テスト)	コーディング作業を含むタスク
source	作成 (ソースコード)	コーディングの最小単位
test	作成 (単体テスト)	コーディングの最小単位

変更点：誤差尺度に BRE を追加　Porru らの研究 [9] では誤差を MRE を用いて評価している．その理由はアジャイルソフトウェア開発の研究で広く用いられているためである．しかし，MRE には，その計算式 6 を見てわかる通り，予測値に比べて実測値がはるかに大きい場合，つまり，過小見積もりを行う場合に誤差が 100%に収束するという特徴がある．そのため，誤差は過大見積もりと過小見積もりをバランスよく評価する尺度である BRE[37] を用いて評価すべきであるといわれている [38]．そこで本実験では，誤差の尺度として BRE を用いた実験も同時に行う．

BRE の計算式 9 は以下の式になる．

$$BRE = \frac{\left|\text{実測値} - \text{予測値}\right|}{denominator} \tag{8}$$

$$denominator = \begin{cases} \text{実測値} & ...\text{実測値} <= \text{予測値} \\ \text{予測値} & ...\text{実測値} > \text{予測値} \end{cases}$$

また精度を評価する際は，MBRE を用いる．MBRE は全てのデータの誤差 BRE を平均したものである．MBRE の計算式は以下の式 9 になる．

$$MBRE = \frac{1}{n}\sum_{i=1}^{n} BRE_i \tag{9}$$

3.5.1　結果

実験の結果を図 12 図 13 図 14 表 7 図 15 図 16 図 17 図 18 表 8 に示す．図 12 図 13 図 14 図 15 図 16 図 17 図 18 は 22 プロジェクトに適用した実験 1 と実験 2 と開発者の予測の結果を箱ひげ図で示したものである．表 7 表 8 は 22 プロジェクトに対して実験 2 の結果と開発者の予測結果を比較して，どちらの精度が良かったかを示した表である．実験 1 と実験 2 の結果に関して比較すると，図 12 図 13 図 14 図 15 図 16 図 17 図 18 のほとんど全てにおいて，その傾向が変わらないことが分かる．つまり，「test」のデータセット以外では，実験 1 よりも実験 2 の方が良い精度を示しており，実験 1 と実験 2 のどちらにおいても，タスクの作業種類を限定して構築したデータセットの方が良い精度を示す傾向がある．

また，実験 2 と開発者の予測の結果に関して比較すると，図 12 図 13 図 14 表 7 より，MMRE で評価をした場合は，過去に実施されたソフトウェア開発 PBL のプロジェクトデータを用いる手法の精度は「基本タスク」と「test」以外は開発者の予測の精度と同程度の精度であった．しかし，図 15 図 16 図 17 図 18 表 8 より，MBRE で評価した場合，過去に実施されたソフトウェア開発 PBL のプロジェクトデータを用いる手法の精度は「基本タスク」以外は開発者の予測の精度よりも良い精度であった．

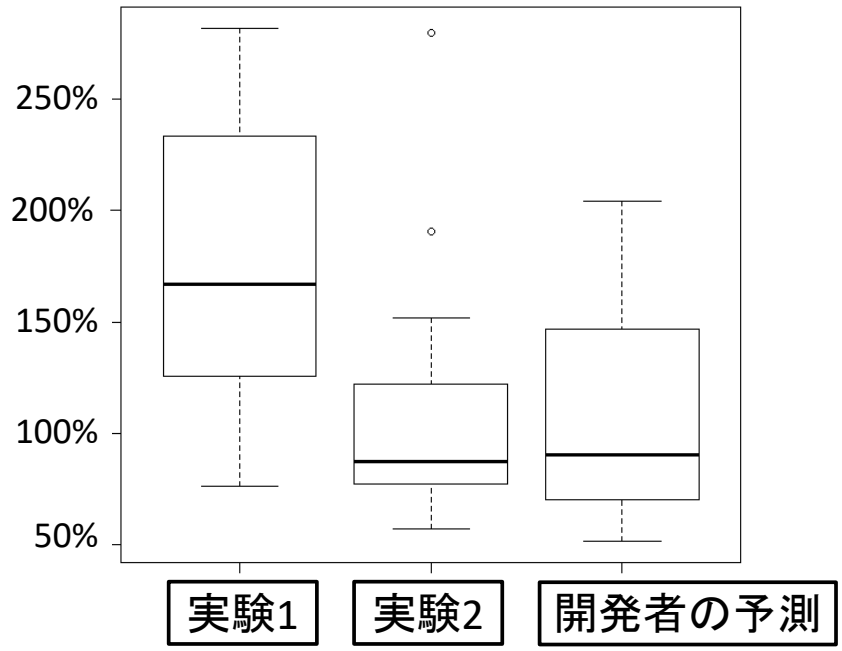

図 12: 22 プロジェクトの MMRE の値を比較した箱ひげ図 (コーディング)

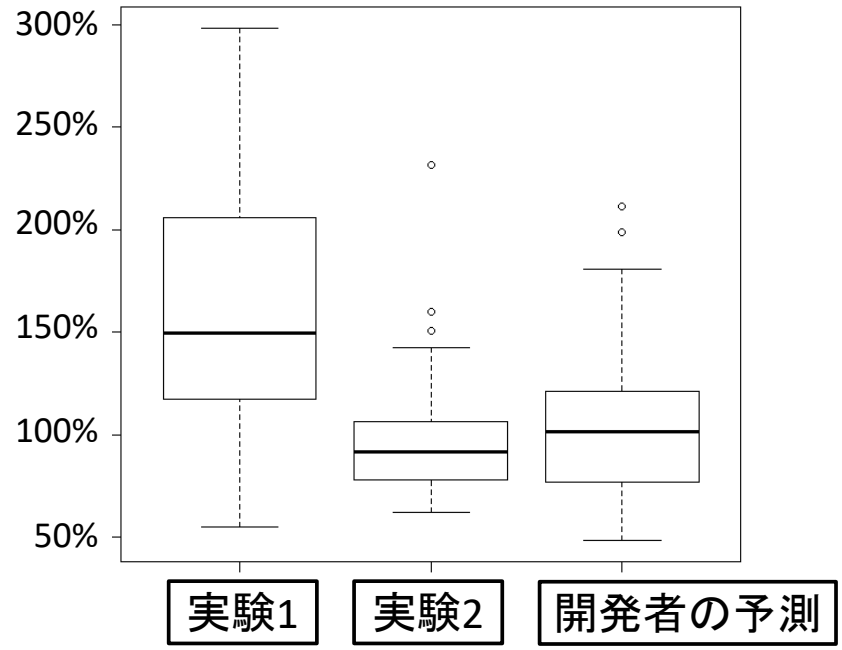

図 13: 22 プロジェクトの MMRE の値を比較した箱ひげ図 (source)

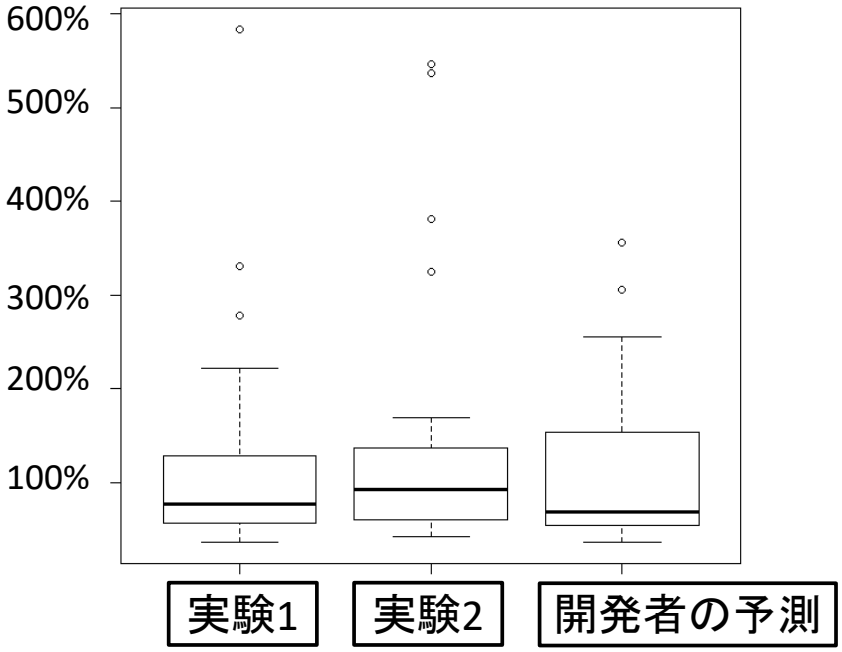

図 14: 22 プロジェクトの MMRE の値を比較した箱ひげ図 (test)

表 7: 精度 (MMRE) が良かったプロジェクトの数の比較

実験名	基本タスク	コーディング	source	test
実験 2	5	11	12	6
開発者の予測	17	11	10	16

表 8: 精度 (MBRE) が良かったプロジェクトの数の比較

実験名	基本タスク	コーディング	source	test
実験 2	9	17	16	14
開発者の予測	13	5	6	8

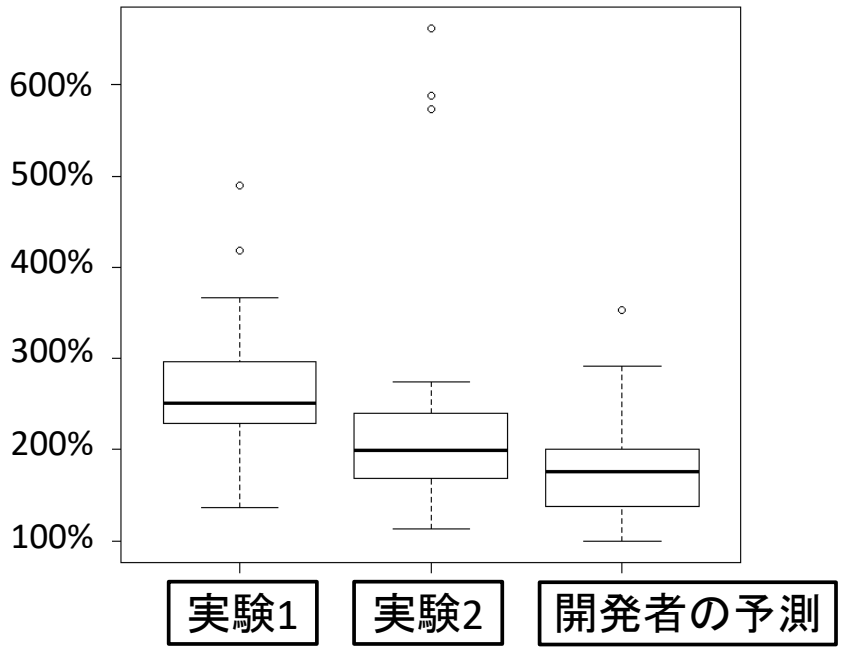

図 15: 22 プロジェクトの MBRE の値を比較した箱ひげ図 (基本タスク)

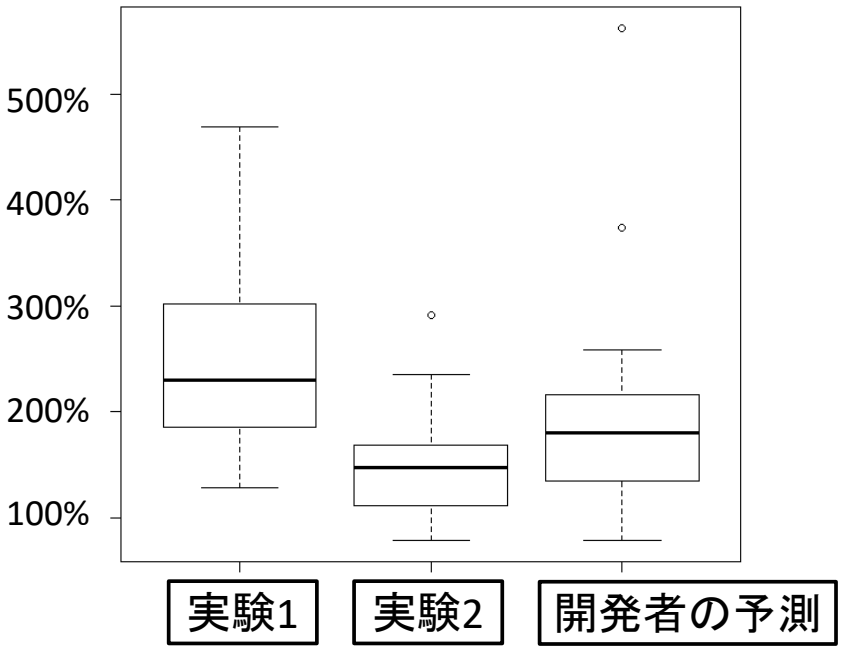

図 16: 22 プロジェクトの MBRE の値を比較した箱ひげ図 (コーディング)

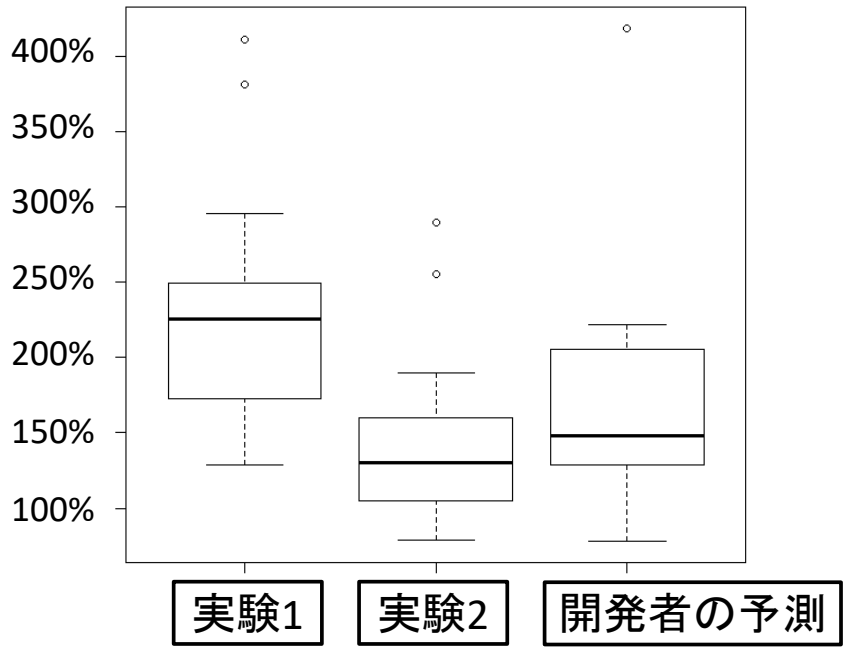

図 17: 22 プロジェクトの MBRE の値を比較した箱ひげ図 (source)

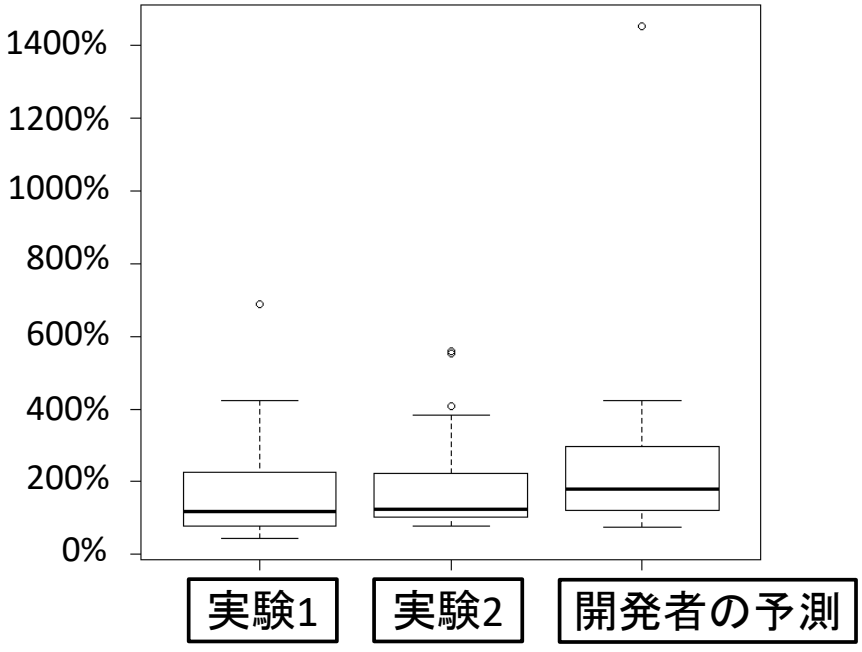

図 18: 22 プロジェクトの MBRE の値を比較した箱ひげ図 (test)

表 9 は過去に実施したソフトウェア開発 PBL のプロジェクトデータを用いた場合の実験の詳細な結果を示した表である．一番良い結果のプロジェクトだと 100%前後の精度を示していることがわかる．

表 9: 実験 2 の 22 プロジェクトに関する詳細な結果 (MBRE)

MBRE	基本タスク	コーディング	source	test
2016G1	166%	92%	104%	81%
2016G2	589%	145%	132%	121%
2016G3	175%	131%	132%	131%
2016G8	663%	154%	160%	104%
2016G9	200%	170%	190%	109%
2015G1	185%	132%	120%	122%
2015G2	197%	198%	131%	559%
2015G3	131%	96%	96%	104%
2015G4	170%	111%	121%	118%
2015G5	116%	94%	88%	101%
2015G6	113%	86%	92%	78%
2015G7	274%	291%	255%	409%
2015G8	212%	207%	160%	554%
2014G1	236%	111%	129%	93%
2014G2	217%	158%	116%	383%
2014G3	178%	150%	113%	174%
2014G4	239%	133%	105%	178%
2014G5	247%	235%	289%	169%
2014G6	168%	160%	175%	126%
2014G7	574%	164%	181%	221%
2014G8	161%	78%	79%	77%
2014G9	202%	169%	135%	302%

また，表 10 はソフトウェア開発 PBL で実際に開発者が予測した結果と過去に実施したソフトウェア開発 PBL のプロジェクトデータを用いた場合の実験の結果全てを比較した表である．+が付いている項目は開発者の予測の方が良かった項目で，-が付いている項目は過去に実施したソフトウェア開発 PBL のプロジェクトデータを用いた場合の実験の結果の方が

良かった項目である．開発者の予測よりも良い精度を示しているプロジェクトが複数存在していることがわかる．

表 10: 学生の予測と比較した結果

	基本タスク	コーディング	source	test
2016G1	-16%	-19%	-19%	-7%
2016G2	+402%	-60%	-72%	-83%
2016G3	+70%	+31%	+54%	-12%
2016G8	+481%	-34%	-57%	+8%
2016G9	+50%	+16%	+61%	-97%
205G1	+26%	-3%	-11%	-18%
205G2	-3%	-18%	-26%	+244%
205G3	-16%	-43%	-52%	-18%
205G4	-122%	-84%	-101%	-25%
205G5	+5%	-42%	-15%	-91%
205G6	-9%	-49%	-26%	-89%
205G7	+64%	+33%	+42%	+81%
205G8	+43%	+5%	+12%	+275%
2014G1	+118%	-12%	-8%	-4%
2014G2	-28%	-88%	-26%	-40%
2014G3	-15%	-95%	-92%	-140%
2014G4	+101%	-35%	-70%	+21%
2014G5	-21%	-139%	-129%	-127%
2014G6	+9%	-12%	-35%	+13%
2014G7	+390%	-25%	+11%	+7%
2014G8	+62%	+0%	-0%	+1%
2014G9	-152%	-394%	+6%	-1152%

3.5.2 考察

タスクの作業種類で層別した場合に「test」データセット以外で過去に実施したソフトウェア開発 PBL のプロジェクトのデータを用いた手法の精度が向上する理由としては，やはり，予測においてはタスクの作業種類は重要であったため，その分類を確実に行えるようにデー

タセットを層別したことで予測精度が向上したと考えられる．また，タスクの作業種類を限定したことにより，類似している Issue Report の検索精度が向上したために，予測精度が向上したという理由もあると思われる．今回はタスクの作業種類でデータセットを層別するという手段を用いたが，本来であれば，タスクの作業種類と「作業時間」の相関関係を自動的に計算して，予測値に反映させることにより，タスクの作業種類の分類による予測精度の向上は達成されるべきである．

また，MBRE で評価すると開発者の予測の精度が悪くなる理由としては，人間の判断に依存する予測は過小見積もりになる傾向があると Porru らの研究 [9] でも言われており，その傾向が結果にも表れたと考えられる．逆に，過去に実施したソフトウェア開発 PBL のプロジェクトのデータを用いた自動的な見積もり手法に関する精度の悪化は開発者の予測ほどではないということから，過去に実施したソフトウェア開発 PBL のプロジェクトのデータを用いた自動的な見積もり手法は人間の判断に依存しないため，予測が過小見積もりになる傾向を小さくすることが可能と考えることもできる．

次に，「test」のデータセットが他の層別されたデータセットとは違う傾向を示す場合があることに関する考察について述べる．「test」データセットに含まれる”作成 (単体テスト)”のタスクは，”作成 (ソースコード)”で実装したソフトウェアの機能をテストするためのコードを書くタスクである．そのため，”作成 (ソースコード)”と比べ，タスクの完了の定義が難しい．同じ”作成 (単体テスト)”のタスクでもプロジェクトによっては他のプロジェクトよりも多くのテストコードを書いている場合などがあり，「作業時間」の値にばらつきが出る場合が多く見受けられる．その影響があったために，「test」データセットは他のデータセットと比べると予測精度が悪くなる傾向があったのではないかと考えられる．

結論として，タスクの作業種類を正確に分類することが出来れば，過去に実施されたソフトウェア開発 PBL のプロジェクトのデータを用いた手法は開発者の予測よりも良い精度で予測可能であると言える．

3.6 RQ に対する回答のまとめ

ここでは，アジャイルソフトウェア開発を取り入れたソフトウェア開発 PBL のプロジェクトにおけるタスクの作業時間の自動的な見積もりを行う手法の調査として行った RQ に対する回答をまとめる．

RQ1: 機械学習に基づく手法は有効であるか？ **No**

RQ2: 過去に実施された **PBL** のプロジェクトデータは予測精度を向上させるか？ **Yes**

RQ3: タスクの種類であらかじめ層別したデータセットを構築することで予測の精度を向上させることが出来るか？ **Yes**

RQ1 について，3.3 節で行った実験では，開発者の予測と同じ程度の精度で予測可能とは言えなかったため，回答としては No となった．RQ2 について，3.4 節で行った実験では，開発者の予測と同じ程度の精度で予測可能とは言えなかった．しかし，RQ1 で行った結果よりも精度が向上することが確認できたため，過去に実施された PBL のプロジェクトのデータは有効であったと言える．そのため，回答としては Yes となった．RQ3 について，3.5 節で行った実験では，RQ2 の結果よりも精度は向上し，また条件によっては開発者の予測よりも良い精度で予測することが可能であった．そのため回答としては Yes となった．

調査の結論としては，過去に実施されたソフトウェア開発 PBL のプロジェクトのデータを用いることにより，開発者の予測よりも良い精度で「作業時間」を予測することが可能である．

4　妥当性の脅威について

4.1　内的妥当性

本研究では Issue Report の文書情報やタスクの作業種類とタスクの作業時間の見積もりの間に因果関係を仮定している．しかし，開発者の能力やプロジェクトの運営方針等のような見積もりに影響する交絡因子が存在していると考えられる．本研究ではこれらの交絡因子による影響を考慮せずに実験を行っているため，これは内的妥当性の脅威である．今後の課題として，開発者の能力や，プロジェクトの持つ特徴などを考慮した実験を行う必要があると考えている．

4.2　外的妥当性

本研究における実験はあるソフトウェア開発 PBL で行われたプロジェクトに限定されており，他のソフトウェア開発 PBL や実際のソフトウェア開発プロジェクトでは同じ結果が得られない可能性があるため，これは外的妥当性の脅威である．今後の課題として他のプロジェクトにおける有効性を検証する必要があると考えている．

4.3　構成概念妥当性

3.3.1 節で説明した本研究の実験に使用する Issue Report の記入漏れ等の修正は手作業で行ったため，漏れが生じる恐れは否定できない．しかし，修正が必要であった Issue Report の数はごく少数であり，仮に修正漏れがあったとしても結果に大きな影響を与えることはないと考えている．

5　あとがき

本研究では，ソフトウェア開発 PBL のプロジェクトにおけるタスクの作業時間の自動的な見積もりを行う手法の調査を行った．その結果，過去に実施したソフトウェア開発 PBL のプロジェクトのデータを用いて類似に基づく手法を適用することにより，Issue Report の内容からタスクの作業時間の自動的な見積もりを開発者の予測よりもよい精度で行うことが出来た．

今後の課題としては，過去に実施されたソフトウェア開発 PBL のプロジェクトのデータを用いた手法の更なる改良があげられる．具体的な方針としては，2 つの方針が考えている．1 つ目は，予測に用いる過去に実施されたソフトウェア開発 PBL のプロジェクトのデータを予測対象のプロジェクトと類似したプロジェクトに限定するというプロジェクトの持つ特徴を考慮する方針を考えている．もう 1 つは，過去に実施されたソフトウェア開発 PBL のプロジェクトの蓄積データから，開発者個人の能力を計測し予測結果に反映させるという開発者の能力を考慮する方針も考えている．

また，異なるコンテキストへの適用を通じた有効性の評価も今後の課題である．他のソフトウェア開発 PBL のプロジェクトや実際の企業でのソフトウェア開発プロジェクト等でも過去の実施されたプロジェクトのデータを用いることでより良い精度が得られるかを調査する必要があると考えている．

謝辞

本研究の全過程を通し，理解あるご指導を賜り，的確なご助言を頂きました 楠本 真二 教授に心より感謝申し上げます．

本研究に関して，適切なご指導を賜り，また日常の議論の中で多くのご助言を頂きました 肥後 芳樹 准教授に深く感謝申し上げます．

本研究に関して，親切なご指導を賜り，また日常の中で多大なるご助力を頂きました 柗本 真佑 助教に深く感謝申し上げます．

本研究に関して，データを提供していただくとともに，ミーティングにおける議論の中で貴重なご助言を頂きました，大阪工業大学情報科学部情報システム学科の 井垣 宏 准教授に深く感謝申し上げます．

研究室生活の中で，事務作業を行う際に多大なるご支援を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻楠本研究室事務員の 神谷 智子 氏に深く感謝申し上げます．

研究室生活の中で相談に乗って頂き，また励まして頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 2 年の小倉 直徒 氏，同 佐飛 祐介 氏，同 鷲見 創一 氏，同 幸 佑亮 氏，同 横山 晴樹 氏に深く感謝申し上げます．

研究室生活を大変豊かにして頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 1 年の下仲 健斗 氏，同 中島 弘貴 氏，同 山田 悠斗 氏，同 山本 将弘 氏に深く感謝申し上げます．

研究室の環境維持に多くのご助力を頂きました，大阪大学基礎工学部情報科学科 4 年の有馬 諒 氏，同 佐々木 美和 氏，同 谷門 照斗 氏，同 松尾 裕幸 氏，同 山田 涼太 氏に深く感謝申し上げます．

最後に，本研究に至るまでに，講義，演習，実験等でお世話になりました大阪大学大学院情報科学研究科の諸先生方に，この場を借りて心から御礼申し上げます．

参考文献

[1] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター. ソフトウェア開発見積りガイドブック～IT ユーザとベンダにおける定量的見積りの実現～. オーム社, 2006.

[2] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター. IT ユーザとベンダのための定量的見積りの勧め～見積り精度を向上する重要ポイント～. オーム社, 2005.

[3] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター. ソフトウェア改良開発見積りガイドブック～既存システムがある場合の開発～. オーム社, 2007.

[4] B.W.Boehm, C.Abts, A.W.Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with Cocomo Ⅱ.* Prentice Hall, 2000.

[5] 齋藤尊, 新美礼彦, 伊藤恵. 過去の pbl の情報を用いた工数見積り支援ツールの開発. 日本ソフトウェア科学会大会論文集, Vol. 31, pp. 434–442, 2014.

[6] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[7] 和島史典 栗林健太郎 柴田博志 家永英治貝瀬岳志. スクラム実践入門: 成果を生み出すアジャイルな開発プロセス.

[8] 平岡嗣晃 前川祐介英繁雄. ハイブリッドアジャイルの実践. リックテレコム, 2013.

[9] Simone Porru, Alessandro Murgia, Serge Demeyer, Michele Marchesi, and Roberto Tonelli. Estimating story points from issue reports. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2016, pp. 2:1–2:10, New York, NY, USA, 2016. ACM.

[10] Viljan Mahnič and Tomaž Hovelja. On using planning poker for estimating user stories. *Journal of Systems and Software*, Vol. 85, No. 9, pp. 2086–2095, 2012.

[11] 井垣宏, 福安直樹, 佐伯幸郎, 柗本真佑, 楠本真二. アジャイルソフトウェア開発教育のためのチケットシステムを用いたプロジェクト定量的評価手法の提案. 情報処理学会論文誌, Vol. 56, No. 2, pp. 701–713, feb 2015.

[12] Hadj Batatia, Alain Ayache, and Hannu Markkanen. Netpro: An innovative approach to network project based learning. In *Proceedings of the International Conference on Computers in Education*, ICCE '02, pp. 382–, Washington, DC, USA, 2002. IEEE Computer Society.

[13] 沢田篤史, 小林隆志, 金子伸幸, 中道上, 大久保弘崇, 山本晋一郎ほか. 飛行船制御を題材としたプロジェクト型ソフトウェア開発実習. 情報処理学会論文誌, Vol. 50, No. 11, pp. 2677–2689, 2009.

[14] 松澤芳昭, 杉浦学, 大岩元ほか. 産学協同の pbl における顧客と開発者の協創環境の構築と人材育成効果. 情報処理学会論文誌, Vol. 49, No. 2, pp. 944–957, 2008.

[15] Viljan Mahnic. A capstone course on agile software development using scrum. *IEEE Transactions on Education*, Vol. 55, No. 1, pp. 99–106, 2012.

[16] Maria Paasivaara, Casper Lassenius, Daniela Damian, Petteri Räty, and Adrian Schröter. Teaching students global software engineering skills using distributed scrum. In *Software Engineering (ICSE), 2013 35th International Conference on*, pp. 1128–1137. IEEE, 2013.

[17] Christelle Scharff. Guiding global software development projects using scrum and agile with quality assurance. In *Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on*, pp. 274–283. IEEE, 2011.

[18] Muhammad Usman, Emilia Mendes, Francila Weidt, and Ricardo Britto. Effort estimation in agile software development: a systematic literature review. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, pp. 82–91. ACM, 2014.

[19] James Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, Vol. 3, , 2002.

[20] Pekka Abrahamsson, Ilenia Fronza, Raimund Moser, Jelena Vlasenko, and Witold Pedrycz. Predicting development effort from user stories. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ESEM '11, pp. 400–403, Washington, DC, USA, 2011. IEEE Computer Society.

[21] Jira software - ソフトウェアチーム向け課題管理、プロジェクト管理ソフトウェア — atlassian. https://ja.atlassian.com/software/jira.

[22] Overview - redmine. http://www.redmine.org/.

[23] The trac project. https://trac.edgewall.org/.

[24] 角田雅照, 大杉直樹, 門田暁人, 松本健一, 佐藤慎一ほか. 協調フィルタリングを用いたソフトウェア開発工数予測方法. 情報処理学会論文誌, Vol. 46, No. 5, pp. 1155–1164, 2005.

[25] 江川翔太, 岡野浩三, 楠本真二. ソフトウェア文書匿名化ツールの試作. 電子情報通信学会論文誌 D, Vol. J98-D, No. 11, pp. 1419–1422, 11 2015.

[26] Kuromoji の github リポジトリ. https://github.com/atilika/kuromoji.

[27] Cloud spiral. http://cloud-spiral.enpit.jp/.

[28] Apache tomcat - welcome! http://tomcat.apache.org/.

[29] Jersey. https://jersey.java.net/.

[30] Mongodb for giant ideas — mongodb. https://www.mongodb.com/.

[31] Ken Schwaber and Jeff Sutherland. The scrum guide (2013). *http://www. scrum. org/Scrum-Guides¿. Acessado em*, Vol. 16, p. 18, 2013.

[32] Weka 3 - data mining with open source machine learning software in java. http://www.cs.waikato.ac.nz/ml/weka/.

[33] Tron Foss, Erik Stensrud, Barbara Kitchenham, and Ingunn Myrtveit. A simulation study of the model evaluation criterion mmre. *IEEE Trans. Softw. Eng.*, Vol. 29, No. 11, pp. 985–995, November 2003.

[34] The comprehensive r archive network. https://cran.r-project.org/.

[35] R 言語逆引きハンドブック: 2.14.1 に対応! シーアンドアール研究所, 2012.

[36] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki. Robust regression for developing software estimation models. *Journal of Systems and Software*, Vol. 27, No. 1, pp. 3 – 16, 1994.

[37] Kjetil Molokken-Ostvold and Magne Jorgensen. A comparison of software project overruns-flexible versus sequential development models. *IEEE Trans. Softw. Eng.*, Vol. 31, No. 9, pp. 754–766, September 2005.

[38] 小野健一, 角田雅照. ソフトウェア開発工数見積もりにおける外れ値の影響評価. 研究報告ソフトウェア工学（SE）, Vol. 2015, No. 7, pp. 1–7, mar 2015.