筑波大学大学院博士課程

システム情報工学研究科特定課題研究報告書

リポジトリデータを利用した
ソフトウェア開発者評価支援システムの開発
―メトリクスデータ生成モジュールの開発―

邹一民

修士（工学）

(コンピュータサイエンス専攻)

指導教員　田中二郎

２０１６年　３月

概要

本プロジェクトは，ソフトウェア開発プロジェクトの管理者による，客観的に個人評価を見たいという要望に応じる個人評価支援システムを開発するものである．プロジェクト管理者は本システムが提供する個人のパフォーマンスデータを参照することで，プロジェクト発足時のメンバ編成や最終的な個人評価の参考にすることができる．主観的な個人レポートやアンケートとは異なる，客観的かつ定量的な個人のデータを提供することで，客観的な個人評価を可能にする．筆者は，反復型開発手法に沿ったプロジェクトタスク管理を担当する．システム開発に関するタスクをまとめ，WBS ファイルの作成，開発者を評価するメトリクスの洗出しをした．システム開発に関しては，あるプロジェクト内の特定開発者の「主体性メトリクス」,「発言力メトリクス」,「作業効率メトリクス」と「役割担当メトリクス」の内部処理部分の実装を担当した．そして三つのメトリクスを一つの画面で表示させるため，メトリクス一覧画面の設計と javascript 部分の実装を担当した．この部分を実現するための準備として，GitHub API， RedMine API から取得できるデータのスコープを調査し，D3.js と言う可視化ライブラリの使用方法を学習した[1][2].

目次

図目次

表目次

第1章 序論

1.1 本プロジェクトについて

本プロジェクトは顧客を日立製作所様とし，課題担当教員を三末和男教授，学生 4 人のメンバから構成されるチーム ViBi が開発を担当する自由企画実現型研究開発プロジェクトである．今回，筆者達のチーム ViBi は顧客に対し，リポジトリデータを利用したソフトウェア開発プロジェクト管理者向け個人評価支援システムを提案する．

1.2 背景と目的

ソフトウェア開発プロジェクトの多くは複数のソフトウェア開発者はチームを組み，共同作業である．しかしプロジェクトの管理者にとって，管理しているプロジェクトに対する各ソフトウェア開発者のプロジェクト内のパフォーマンス，プロジェクトへの貢献度(能力や経験)を定量的に測れないという問題がある．プロジェクト管理者の主観的かつ曖昧な評価基準に沿って，不適切な評価結果，個人イメージが生成する可能性があるため，ソフトウェア開発者のモチベーションの喪失する場合がある．また，ソフトウェア開発者がプロジェクトの関心度が下がり，自分の作業に影響があり，最後にプロジェクト運営の失敗につながる場合も少なからず存在する．この問題の解決のため，プロジェクト管理者が複数のソフトウェア開発者に対して客観的，定量的な判断材料による，個人評価をできるような問題解決策が望まれている．

株式会社日立製作所により提示された問題を解決するため，本プロジェクトでは MITEMIRU の提案し，開発をおこなう．MITEMIRU は，評価対象者が直接関連するソフトウェア開発の時利用した構成管理ツール，チケット管理ツールのリポジトリから取得したデータを集計・分析・加工する．加工データに対する，可視化モジュールを作り，プロジェクト管理者に表現する．リポジトリデータを利用し生成したグラフにより，ソフトウェア開発者のパフォーマンスに関連する定量的な評価材料としてプロジェクト管理者に提供し，日立に提示された問題を解決する．

MITEMIRU の設計から実装まで流れとして，まずはシステムの設計段階で，リポジトリマイニングやプロジェクト管理等に関連する文献，類似システムの調査により，リポジトリ上に存在するデータを利用し，ソフトウェア開発者のパフォーマンスを評価するアプローチを明確にする．次に，このアプローチに基づき，ソフトウェア開発者のパフォーマンス評価の判断材料となるメトリクスの洗い出し，と各メトリクスが利用できるメトリクスデータ，およびメトリクスデータの表現に必要となる実際のリポジトリ上のデータの取得手法を検討する．最後に，提供するメトリクスデータと具体的な可視化方法を決定する．

なお本プロジェクトでは，多数のソフトウェア開発者が共同作業するソフトウェア開発プロジェクトを対象として，客観的な各ソフトウェア開発者の評価資料に基づき，評価支援を提供する目標とする．

1.3　　報告書の構成

本報告書は七章で構成されている．各章の主要内容を表 1-1 に示す.

表 1-1：報告書の構成

章番号	主要内容
第一章	本研究開発プロジェクトの開発目的，顧客の抱えている問題，提案を紹介する.
第二章	システムの存在理由，要件，概要図，画面遷移図を紹介する.
第三章	類似システムと文献の調査，システムの提案手法の決定を説明する.
第四章	開発管理, 利用したツール, 各メンバの役割と四つのスプリントの実績を説明する.
第五章	エンドユーザ評価テストの概要，結果と改善案を紹介する.
第六章	筆者が四つのスプリント内の担当部分と実装した機能を紹介する.
第七章	本プロジェクト今後の期待を説明する.

第2章 システム概要

2.1　　システムの存在理由

プロジェクト管理者の多くがメンバのパフォーマンスを客観的に把握したいと考えている．このような情報は，プロジェクト発足時のメンバ編成や個人評価の判断材料となる． アンケートや個人レポートなどの主観的あるいは定性的な判断材料とは異なり，客観的かつ定量的な判断材料を視覚的に提供することで，データに基づく高効率な個人評価が可能になる．

本システムはプロジェクト管理ツール Redmine を利用することでプロジェクトリスク予測等のメトリクスを定量化するシステムである EPM-X と違い，開発者を中心とし，各開発者の動きをメトリクスで定量的に表示できる[4].

2.1　　システム利用者

本システムの利用対象者は，ソフトウェア開発プロジェクトの管理者である．例として，企業におけるプロジェクト責任者や，課題解決型学習(PBL: Project Based Learning)の指導教員等が挙げられる．

2.2　　利用者に対する問題

利用者に対する問題として，あるプロジェクトの開発チームの中の開発者の貢献度，個人スキルする時，客観的な判断材料が少ない．そのため，ソフトウェア開発者の不適切な評価が良く発生する．ソフトウェアの開発は複数の開発者が担当している．各開発者の能力が違うため，担当している役割分担も違う．複数の役割分担で，プロジェクトを推進している．プロジェクトが推進しつつ，各開発者の役割分担や能力が大きく影響を与える．そのため，プロジェクトの発足段階で，プロジェクトに参加できるソフトウェア開発者を選択する時，開発者の適性，スキルをきちんと考えなければならない．そして，企業での人事評価，大学で先生がPBLプロジェクトにおける学生の成績判定等，プロジェクト始めてから，定期的なチーム内の開発者のスキル判定，貢献度を評価しなければならない．

以上の二つの場合，いずれでも適切な評価が必要である．しかしプロジェクトの管理者として，開発者の経験，開発能力を評価する時，参照できる客観的かつ定量的な判断材料が少ないため，実際の評価が非常に困難である．

本システム, MITEMIRU はこの問題を解消するため, 実際のリポジトリデータを利用し，プロジェクト管理者にソフトウェア開発者評価のための定量的な判断材料を提供する．

2.3　　問題解決の手順

利用者は，チーム・プロジェクト編成時に候補となる個人に関して，本システムを利用してリポジトリ上から候補者のデータを取得する．システムは，取得したデータに応じて，グラフ描画等によりデータを可視化する．利用者は可視化されたデータを閲覧することで候補者の過去のプロジェクトにおける振る舞いを把握でき，チーム編成時や個人評価の判断材料として活用できる．

2.4 顧客に対する価値

本システムで, システムユーザが所有するプロジェクトの Redmine リポジトリの URL と GitHub リポジトリの URL を登録し, かつ各リポジトリに対する認証情報を本システムに登録するだけで，登録したプロジェクト内のソフトウェア開発者全員の情報が閲覧できる．本システムはリポジトリから開発者履歴に関するデータを取得，加工し，グラフを生成し，システムユーザに示す．システムユーザは特定のプロジェクトを選択し，システムが生成したプロジェクト開発者全員に関するグラフが判断材料として, システムユーザが参考する同時, 考察対象となる特定のソフトウェア開発者の能力と傾向が判断できる．各機能の価値は章 2.11 システムが提供する機能で述べる．

2.5 システム機能要件

本プロジェクトの機能要件としては，開発しながら，ミーティング等の手段で顧客と相談し，洗い出した．その後，各要件に優先順位を設定し，顧客と合意を取ってから，開発作業を実施する．優先度は三段階で定義されている，定義は以下の通りである．

- 高: 利用者の最低限の要件を満たすために，システムの動作上，必須となる機能
- 中: 利用者の要件を満たすため必要ではあるが，不足してもシステムを動作させられる機能
- 低: 利用者の要件として求められてはいるが，必要ではない機能

表 2-1：システム機能要件

主機能	機能要件	概要	優先度
ログイン･ログアウト機能	システムユーザの新規登録	システムの利用者のアカウントを作成し，データベースに保存する	高
	ログイン	既にデータベースに保存されたユーザ情報で，システムにログインできる	高
	ログアウト	システムからログアウトできる	高
	システムユーザ情報管理	データベースに保存されたシステムユーザ情報に対し，更新および削除できる(ログインしているユーザの情報のみ)	中
リポジトリ連携機能	リポジトリ認証情報の保存	リポジトリの問い合わせに必要となる認証情報をデータベースに登録できる	高
	開発者情報の登録	リポジトリ登録時点で，関連する全ての開発者情報をデータベースに保存できる	中
メトリクスデータ閲覧機能	プロジェクトの選択	メトリクスデータを閲覧する前に，目標プロジェクトを選択できる	高
	メトリクスデータの生成と可視化(同一プロジェクト)	プロジェクトを選択し，リポジトリからデータを取得した後，各開発者別で，メトリクスデータを集計・分析結果をグラフとして利用者に提供できる.	高
	メトリクスデータの生成と可視化(複数プロジェクト)	考察したい開発者を複数選択し，リポジトリからデータを取得してから，集計・分析結果を生成し，グラフとして可視化し，各開発者のパフォーマンスを比較する形式で利用者に提供する.	中
	メトリクスデータの生成と可視化(単一開発者)	考察したい開発者を選択し，該当開発者が所属している複数のリポジトリのデータを取得してから，集計・分析結果のグラフを生成して利用者に提供する.	中
	レポートファイルの出力	メトリクスデータの結果とグラフを外部ファイルとして出力する.	低
各種管理機能	プロジェクト情報管理	データベースに保存されたリポジトリの認証情報に対し，更新および削除できる	高
	開発者情報管理	データベースに保存された開発者情報に対し，更新，削除できる	中
	システムユーザ情報管理	データベースに保存されたシステムユーザ情報に対し，更新および削除できる(スーパユーザのみ利用可能，全てのシステムユーザの情報が管理できる).	中

2.6　　システム非機能要件

非能要件は主に顧客が本システムを利用する時，リポジトリからのデータ所得とシステム移行性である．

現時点は Redmine リポジトリと GitHub リポジトリを主要リポジトリとして，開発者に関するデータを取得する．将来，この二つのリポジトリ以外のリポジトリも拡張でき，データを所得する可能性があるため，システムの拡張性を非機能要件に考えた．

開発の時,本システムは Heroku というウェブサービスを利用し,そこでサーバーを立て,実装した機能をそこで顧客が利用し，フィードバックをする．今後，MITEMIRU はウェブサービスではなく，個人パソコン環境で，.exe ファイルを利用し，インストールし，利用する可能性がある，かつ個人利用の場合，実行環境の自動構築を考える必要があるため，本システムの移行方式と移行性等の非機能要件を以下の表のように考えた．

表 2-2：システム非機能要件

項目	特徴	概要	優先度
性能・拡張性	リソース 拡張性	システムでのリポジトリデータの取得先の登録が簡単であること.	高
移行性	移行方式	Web Serviceだけでなく，.exeファイルでのインストールが可能であること.	高
	移行対象 (機器)	汎用型コンピュータへ簡便に移行可能であること．汎用型コンピュータで利用可能な範囲のデータベースに移行可能であること.	高
	移行対象 (データ)	汎用型コンピュータで利用可能な範囲のデータベースに移行可能であること.	高

2.7　　システム概要説明

ユーザが本システムを利用する時，まずブラウザでプロジェクトを作成する必要がある．プロジェクトの入力情報としては，Redmine と GitHub のリポジトリの情報とそれに対する認証情報を登録する必要がある．あるリポジトリを登録した時，裏側で，API 経由で，該当リポジトリに含めている開発者全員が自動的に本システムのデータベースに保存される．

次は，ユーザが登録したプロジェクトのメトリクスを閲覧したい時，「メトリクスを見る」画面にアクセスし，プロジェクトを選択してから，裏側で該当プロジェクトの Redmine と GitHub のリポジトリの URL とその URL に対する認証情報を本システムのデータベースから呼び出し，リポジトリの URL と認証情報を引数として，各 API に伝送し，認証成功してから，各リポジトリから開発者全員のデータを取得し，本システムの裏側で，データを分析，集計してから，ユーザのブラウザ上にグラフと言う形式で表現させる．

最後はユーザが「各種管理」機能で，登録したプロジェクトのリポジトリの URL，リポジトリ認証情報，開発者情報を編集，削除できる．

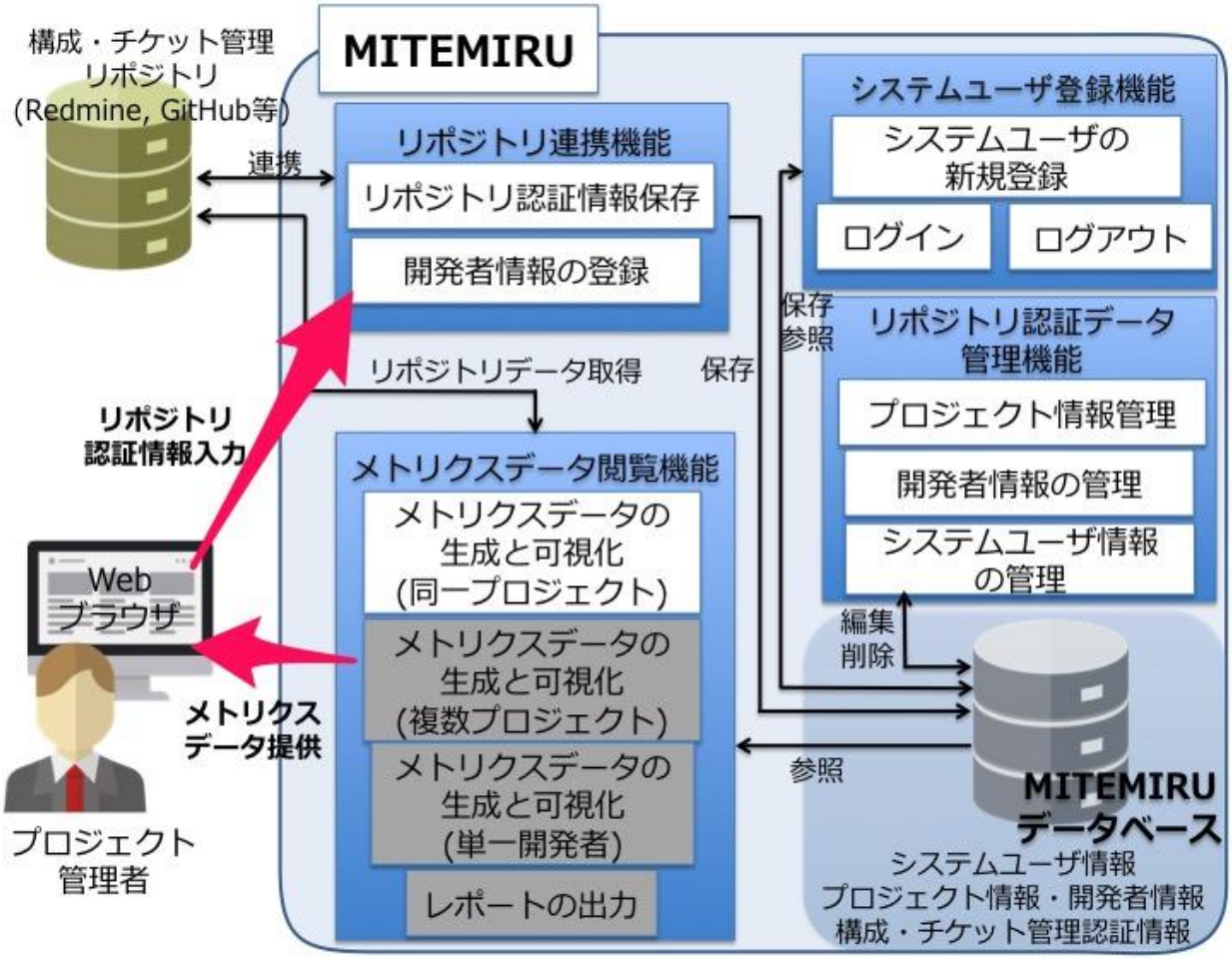


図 2-1：システム概要図

2.8　　システムの画面遷移

利用者は本プロジェクトにアクセスし，最初の画面は「システムユーザ登録画面」である，その画面で新規システムユーザあるいはログイン機能が利用でき，システムに登録してから，本システムは裏側で該当ユーザに対する本システムに登録したプロジェクトが存在しているかどうか処理を行い，既に登録したプロジェクトが存在している場合，システムは自動的に「メトリクスを見る」画面に遷移する．登録したプロジェクトが存在してない（初期利用）の場合，システムは「プロジェクト登録画面」に遷移し，「プロジェクト登録画面」でプロジェクトの新規登録操作を行う．登録処理完了してから，自動的に各種管理機能の中「プロジェクト管理画面」に遷移し，登録したプロジェクトが表示される．この画面で登録したプロジェクトに対する名前，リポジトリの認証情報更新等の機能が利用できる．

ユーザが画面の一番上のメニュー画面を利用し，各画面に遷移できる．「メトリクスを見る」ボタンをクリックすると，「メトリクスデータ閲覧画面」に遷移し，画面上のプルダウンリストから考察したいプロジェクトの名前を選択し，「選択」ボタンを押すと，該当プロジェクトに対するメトリクスグラフが表示できる．

本システムからログアウトしたい場合，画面右上の「ログアウト」ボタンを押すと，システムがログアウト処理を行い，システムが「システムユーザ登録画面」に遷移する．

本システムの画面遷移図を図 2-2 に示す．

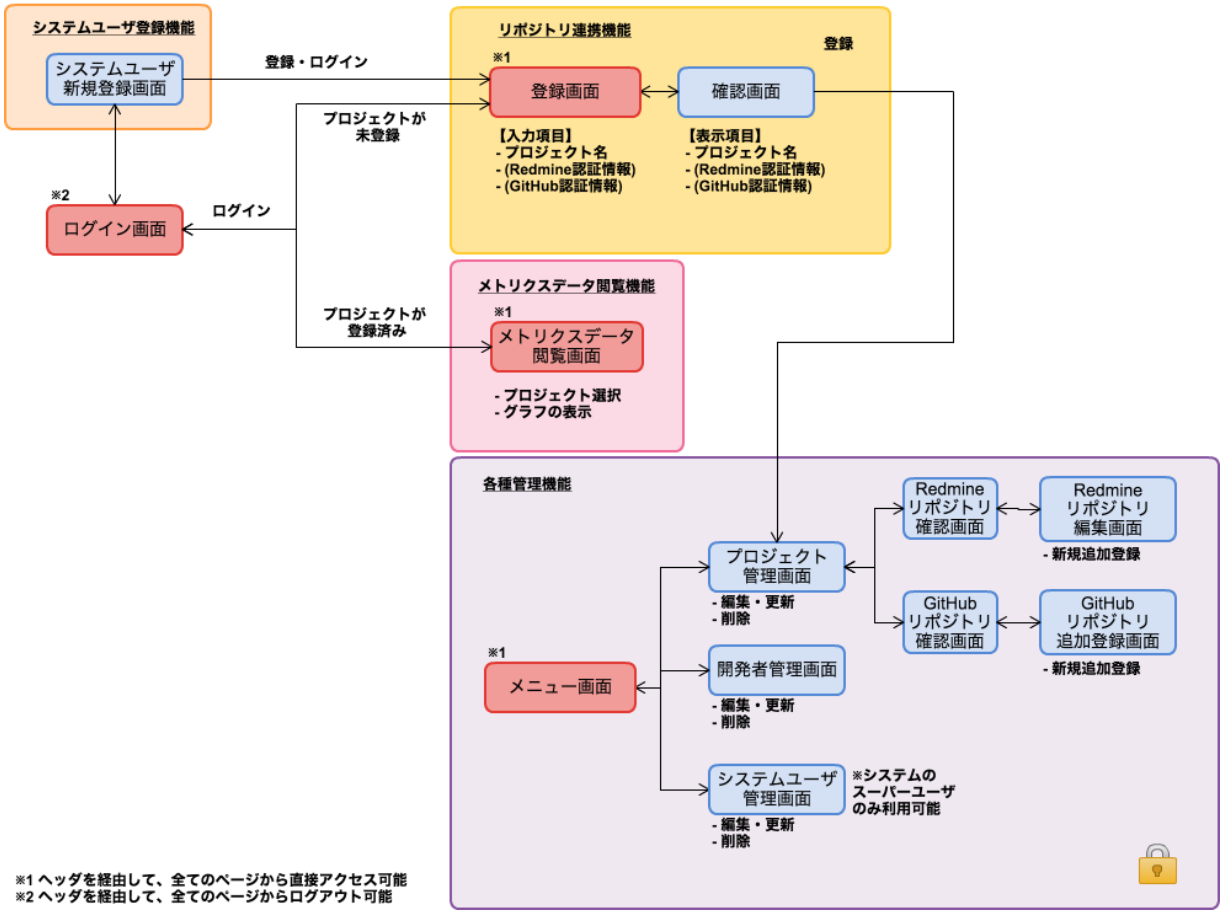


図 2-2：システム画面遷移図

2.9　　システムデータベース構成

各テーブルで保存するデータの説明を表 2-3 に示す.

表 2-3：システムデータベース説明

テーブル名	保存するデータ
roles	システム上の権限を記録するテーブル
roles_users	システム上のユーザと権限の関係を記録するテーブル
users	システム上のユーザを記録するテーブル
redmine_keys_users	Redmine リポジトリ認証情報とユーザ関係を記録するテーブル
github_keys_users	GitHub リポジトリ認証情報とユーザ関係を記録するテーブル
redmine_keys	Redmine リポジトリ認証情報を記録するテーブル
projects	システム上で登録したプロジェクト情報を記録するテーブル
github_keys	GitHub リポジトリ認証情報を記録するテーブル
ticket_repositories	Redmine リポジトリの URL を記録するテーブル
assign_logs	リポジトリと開発者の関係を記録するテーブル
version_repositories	GitHub リポジトリの URL を記録するテーブル
developers	開発者を記録するテーブル

本システムの ER 図を図 2-3 に示す.

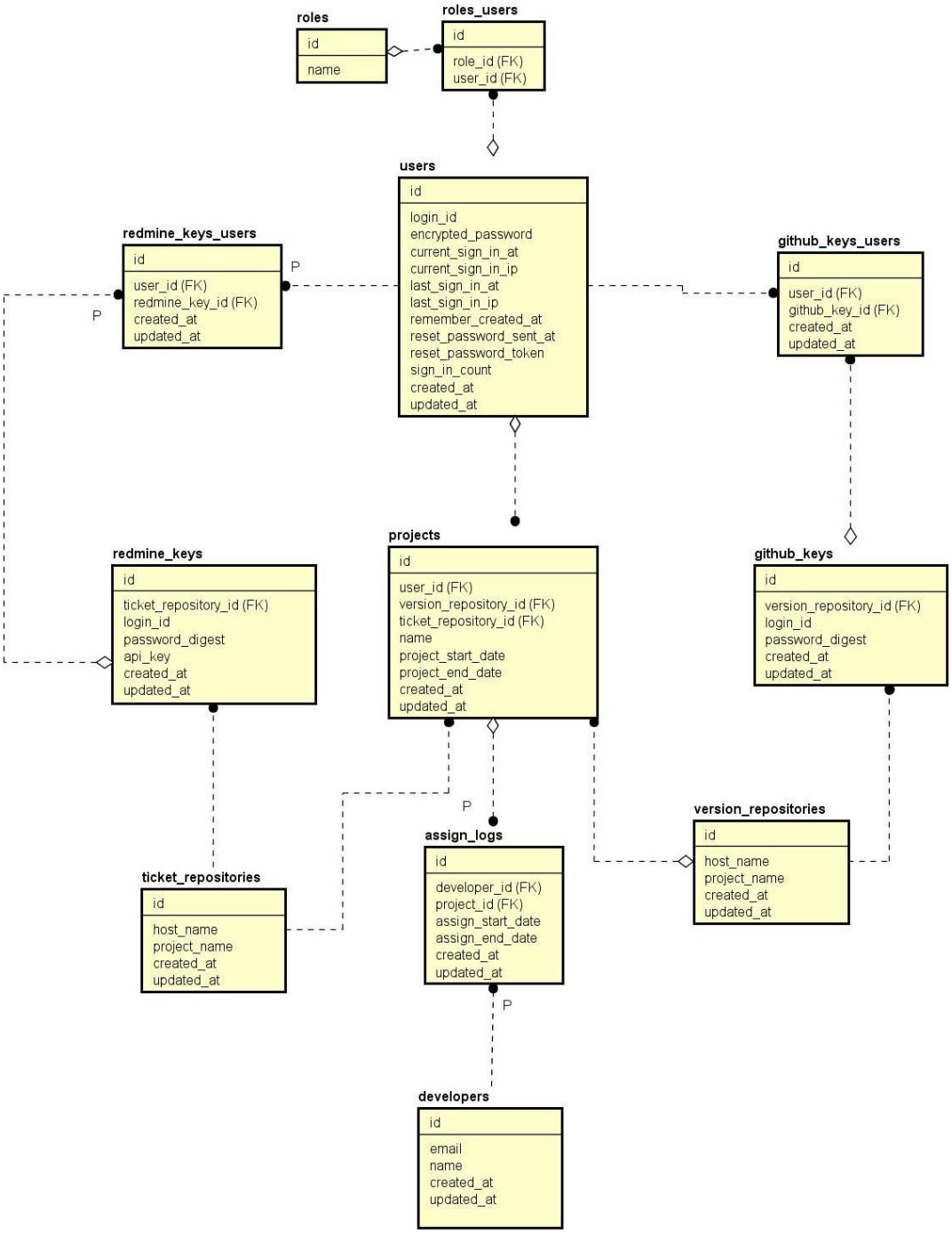

roles
id
name
roles_users
id
role_id (FK)
user_id (FK)
users
id
login_id
encrypted_password
current_sign_in_at
current_sign_in_ip
last_sign_in_at
last_sign_in_ip
remember_created_at
reset_password_sent_at
reset_password_token
sign_in_count
created_at
updated_at
redmine_keys_users
id
user_id (FK)
redmine_key_id (FK)
created_at
updated_at
P
P
github_keys_users
id
user_id (FK)
github_key_id (FK)
created_at
updated_at
redmine_keys
id
ticket_repository_id (FK)
login_id
password_digest
api_key
created_at
updated_at
projects
id
user_id (FK)
version_repository_id (FK)
ticket_repository_id (FK)
name
project_start_date
project_end_date
created_at
updated_at
github_keys
id
version_repository_id (FK)
login_id
password_digest
created_at
updated_at
P
version_repositories
id
host_name
project_name
created_at
updated_at
assign_logs
id
developer_id (FK)
project_id (FK)
assign_start_date
assign_end_date
created_at
updated_at
ticket_repositories
id
host_name
project_name
created_at
updated_at
P
developers
id
email
name
created_at
updated_at

図 2-3：システム ER 図

2.10　システムユーザ利用できる機能

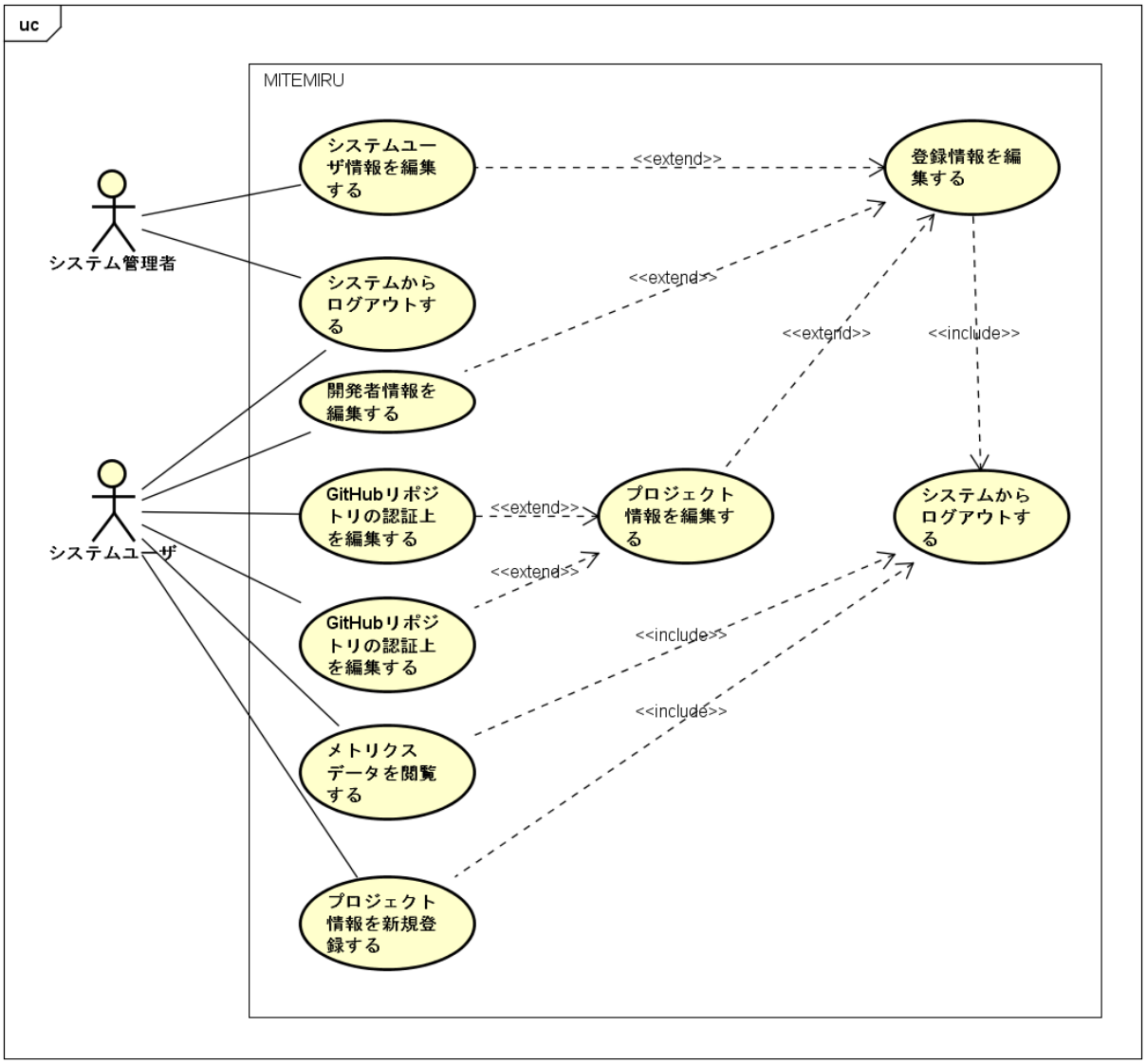


図 2-4：システムユースケース図

本システムで，システムユーザが二つの種類が分けられている．一つ目は普通のシステムユーザである，二つ目はシステム管理者である．

システムユーザとして，本システム上の「プロジェクト情報登録」,「メトリクスデータの閲覧」,「GitHub リポジトリ認証情報の編集」,「Redmine リポジトリ認証情報の編集」,「開発者情報の編集」とその後の一連機能が利用でき，主に自分が所有している情報の管理である．

システム管理者は,「システムユーザの情報管理」機能が利用でき，システム上で既に登録したシステムユーザの情報を管理し，各システムユーザが所有しているデータも見える．システム管理者が本システムに登録した時，ブラウザ上で表示した画面はシステムユーザと違い，本システムのデータベース内の各テーブルの状況を表す画面である．

2.11　システムが提供する機能

これから，本システムが提供できる機能を説明する．現時点で，ユーザの登録機能，所有プロジェクトの登録機能，プロジェクト，あるいは開発者情報の各種管理機能，メトリクス参照機能は実装完了した. システムの機能要件から説明すると,「ログイン・ログアウト機能」,「リポジトリ連係機能」,「メトリクスデータ閲覧機能」と「各種管理機能」を実装した．非機能要件の中の「移行性」部分はまだ考えなかった.

2.11.1　システムユーザの登録，管理

本システムではログインする時，ユーザアカウントを作成し，ログインできる．ログインしたユーザは自分が登録した情報のみ参照，編集できる.

システムを初期利用する場合，ユーザ登録操作を行う必要がある．以下の画面で新しいシステムユーザを作成し，本システムに登録し，ログインができる.

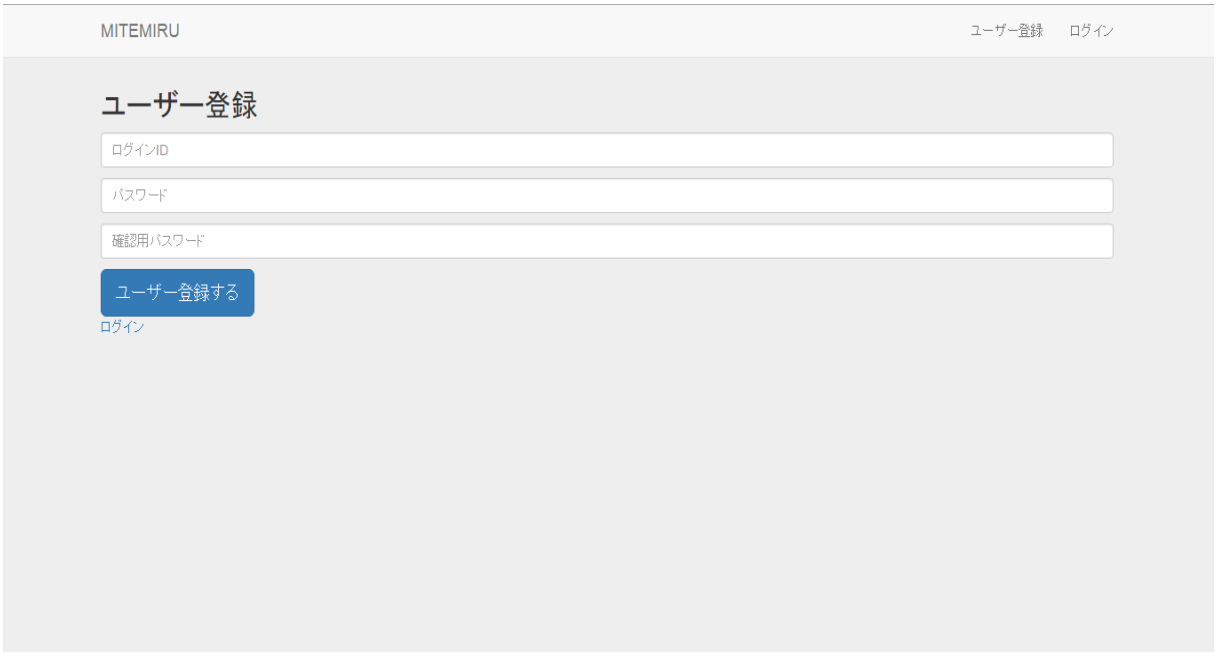


図 2-5：システムユーザ登録画面

2.11.2　リポジトリ情報の登録

本システムでは参照プロジェクトを登録する時，Github リポジトリ，あるいは Redmine リポジトリの最低一つのリポジトリの URL と対応する認証情報を登録する必要がある．登録処理を行う同時，システムが自動的に Redmine リポジトリから開発者を抽出し，データベースに保存する.

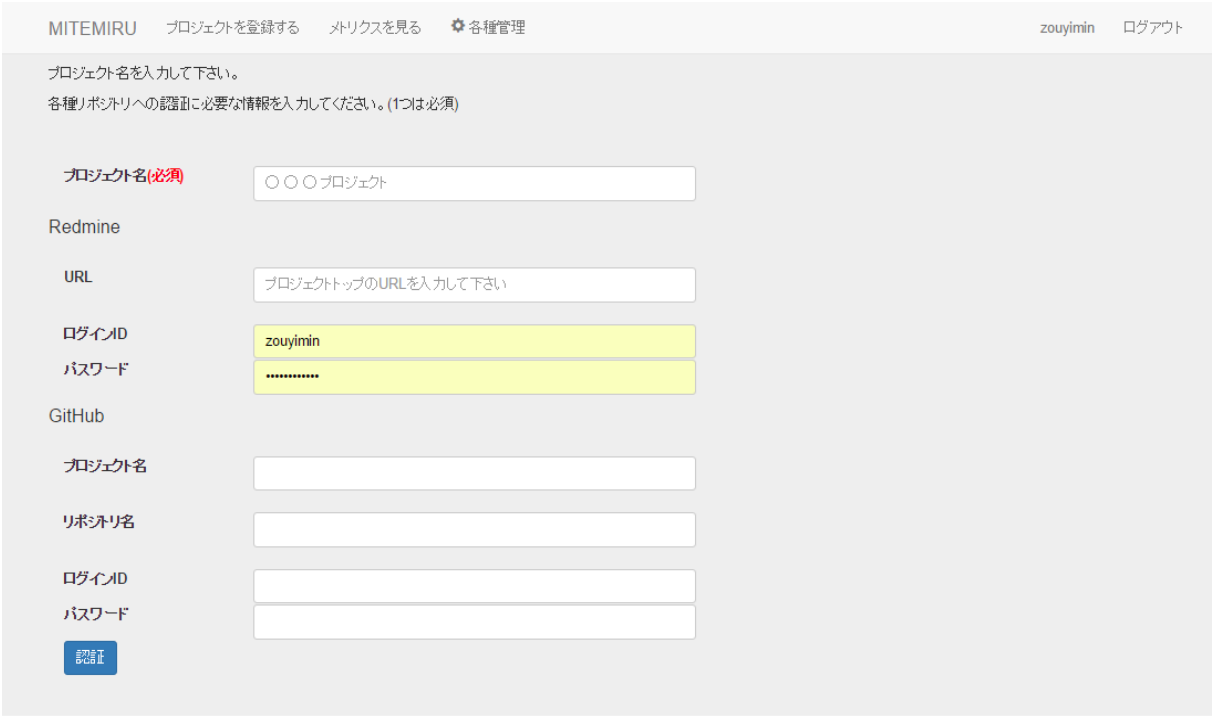


図 2-6：プロジェクト登録画面

登録フローとしては，システムユーザが「プロジェクト登録」画面に移動し，画面上の各入力欄に内容を入れてから，一番下の「登録」ボタンを押すと，システムが「新規プロジェクト登録確認画面」に遷移する．システム裏側で処理を実行し，各リポジトリの API 経由で，システムユーザが入力したリポジトリの URL に対する認証操作を行う．認証失敗した場合，画面上で，該当リポジトリの認証状況部分に，未認証が表示されている．認証が成功した場合，該当リポジトリは URL，あるいはプロジェクト名とリポジトリ名だけ表示される．その後，画面上の「登録する」ボタンを押すと，システムが実際にプロジェクトを登録する操作を行う同時に，リポジトリから開発者の情報もデータベースに登録する．全部の処理終わってから，各種管理機能のプロジェクト一覧画面に遷移し，登録したプロジェクトの情報を確認，編集できる．

プロジェクト登録の流れは図 2-8 に示す．

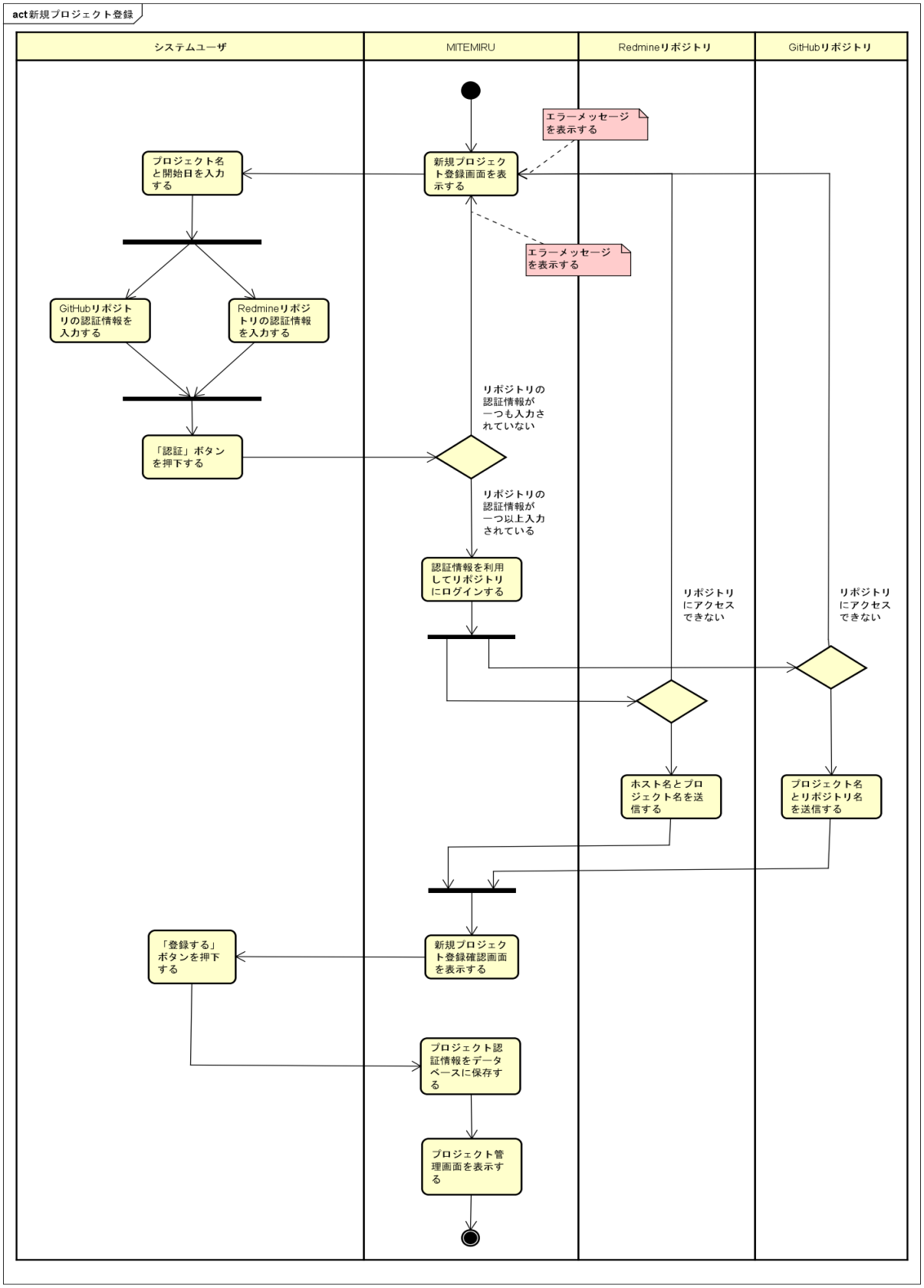


図 2-7：プロジェクト登録フロー

2.11.3 プロジェクト情報の管理

ユーザは既に登録したプロジェクトの情報と開発者の情報を管理でき，リポジトリ認証情報の更新もできる．

下の画面ですでに登録したプロジェクトが表示されている．毎行最後の「編集」ボタンを押すと，プロジェクトの名前を更新できる．「削除」ボタンを押すと，該当プロジェクトと関連するリポジトリ情報，開発者情報全部システムから削除することができる．

認証情報欄で該当プロジェクトのリポジトリのアクセス権限が表示される，アクセスできる場合，「認証済み」と表示されているが，アクセスできない場合，「未認証」と表示される，「未認証」ボタンを押すと，該当リポジトリの認証情報を追加することができる．

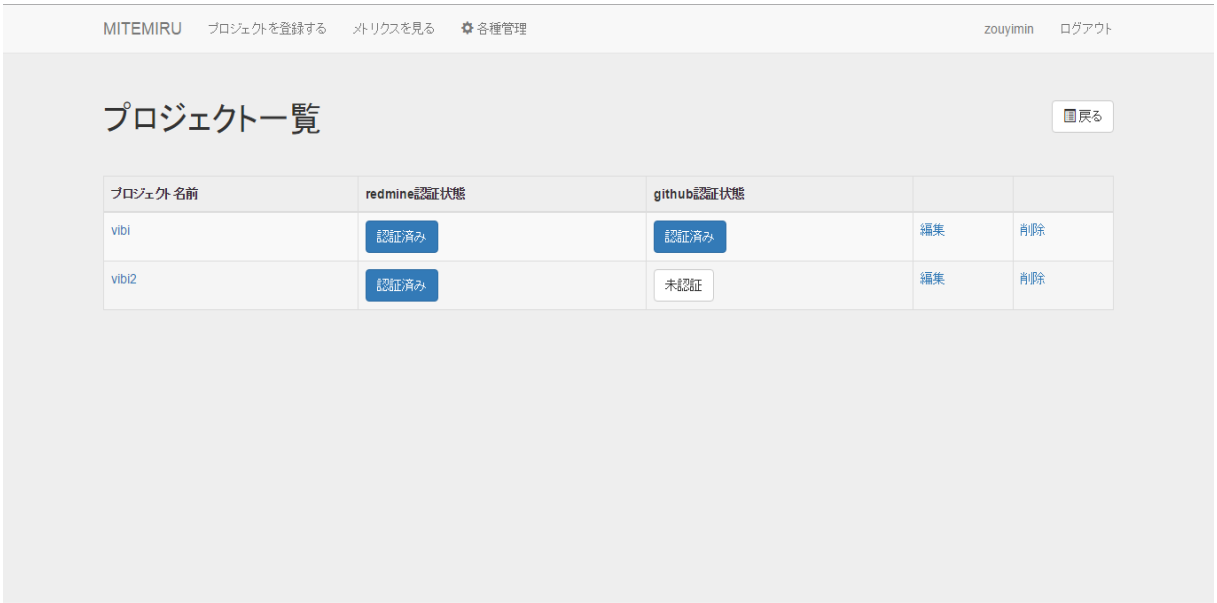


図 2-8：プロジェクト情報管理画面

2.11.4 リポジトリデータの取得

本システムでは GitHub API と Redmine API を利用し，ユーザが既に登録した認証情報を通し，GitHub リポジトリ上からアカウントやプロジェクトに関するデータと Redmine リポジトリ上の担当したチケットデータを取得する[5].

2.11.5 リポジトリデータの集計

Github リポジトリと Redmine リポジトリから取得したデータを利用し，開発者ごとに，定量的なデータを集計する機能を提供する．

メトリクスの処理の流れとして，まずは，選択したプロジェクトのデータをデータベースの中から取得し，その中からリポジトリの URL と認証を抽出する．次は各リポジトリの API を利用し，リポジトリにアクセスし，中のデータを取得する．API から取得したデータを加工し，集計してから，最後に，ブラウザ上でグラフを生成し，システムユーザに描画する．

2.11.6 リポジトリデータの可視化

Github リポジトリと Redmine リポジトリから取得したデータ，およびそれを利用した集計結果をウエブページに伝送し，グラフとしてブラウザ上に描画する，可視化モジュールを提供する．

一部グラフの中に元素をクリックすると，自動的に開発者はある項目に対するランキング機能ができ，更に直感的にチーム内の開発者の貢献度，主体性を把握できる．

現時点, MITEMITE システム上提供できる可視化したメトリクスは「主体性メトリクス」，「発言力メトリクス」と「作業効率メトリクス」，合計三つがある．

一番目の「主体性」メトリクスの可視化モジュールの具体的なイメージは図 2-9 に示す．

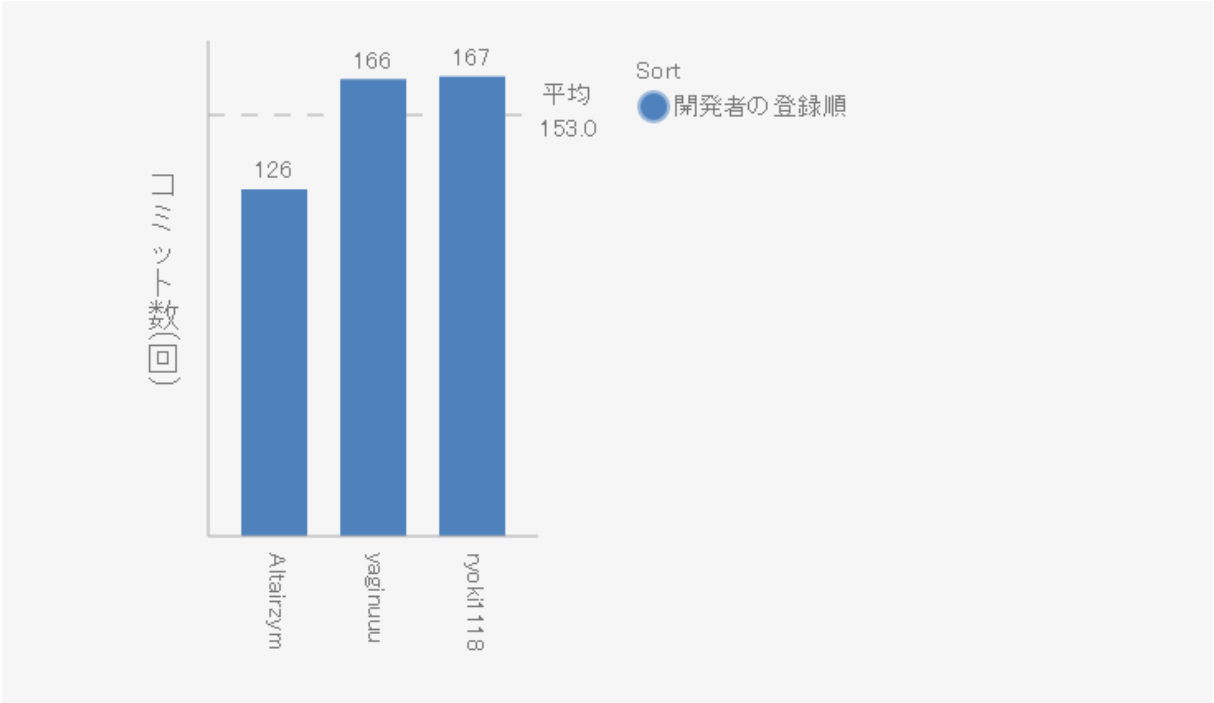


図 2-9：主体性メトリクスの可視化モジュール

システムユーザがこのグラフで選択したプロジェクト内のすべてのソフトウェア開発者のコミット数が閲覧できる．グラウの横軸は開発者の名前，縦軸はコミットの回数（回），点線の横軸は平均コミット数である．平均コミット数は開発者の能力の差と考えなく，合計コミット数/合計開発者人数で計算している．このグラフを参考すると，チーム内の開発者のコミット回数の順位が理解でき，誰が平均コミット数以下も直観的に見える．グラフとなりの「開発者の登録順」ボタンを押すと，自動的にソートでき，コミット回数の順位が表示できる．

このグラフでシステムユーザがチーム内の開発者の実装段階のパフォーマンスが参考でき，各開発者の主体性が分かる．かつ平均コミット数の横軸を利用し，開発者の貢献度も推測できる．

「発言力メトリクス」の可視化モジュールの具体的なイメージは図 2-10 に示す．

MITEMIRU は GitHub API 経由で，選択したプロジェクトに関連ある Github リポジトリから各開発者のコメント情報を取得し，ヒートマップ形式で表現する．このグラフの横軸と縦軸は開発者名を表示している，四方形の元素で各開発者がお互いにコメントした回数を表示する．四方形の色で，開発者の間のコメント回数の頻度が表している，頻度が高くなると，色が濃くなる．開発者間のコメント回数が少ないと，色が薄い．マウスが四方形の元素の上に移動すると，コメント回数が表示される．

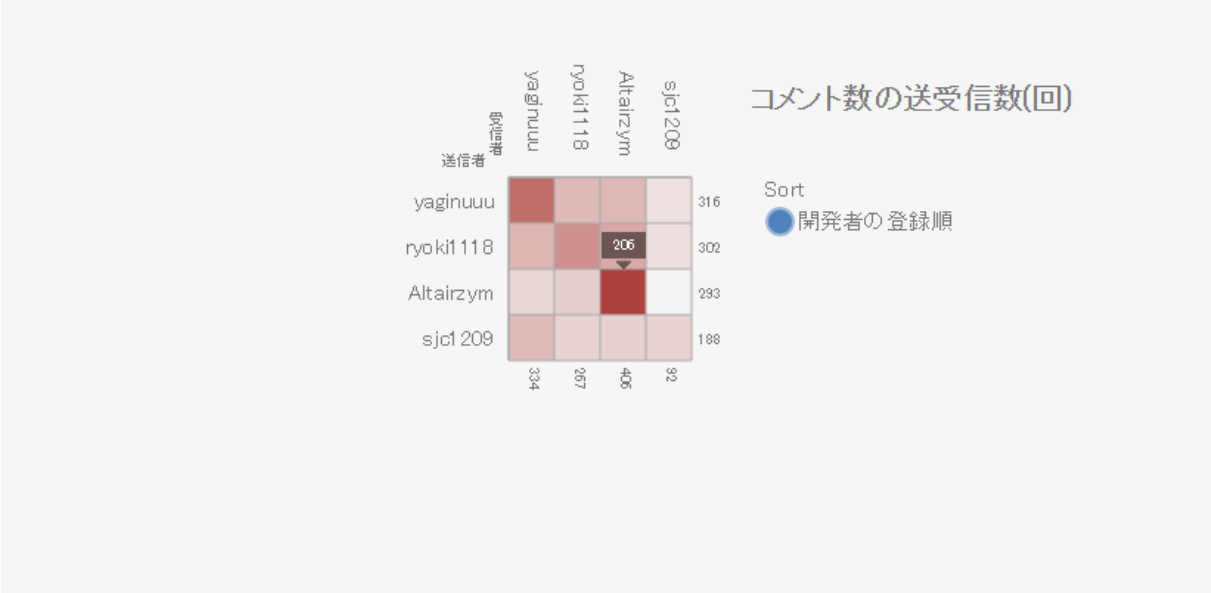


図 2-10：発言力メトリクスの可視化モジュール

システムユーザが発言力メトリクスのグラフを参考すると，チーム内の開発者の間のコメント回数が理解でき，発言力が強い人と弱い人が区別，意識できる．発言力が弱い人にコミュニケーションを促進し，交流場面の問題が早めに発見できる．将来，新しいプロジェクトのチームを編成する時，発言力が弱い人に更に交流し，交流場面の問題が回避でき，プロジェクトの運営を推進することができる．

「作業効率メトリクス」の可視化モジュールの具体的なイメージは図に示す．

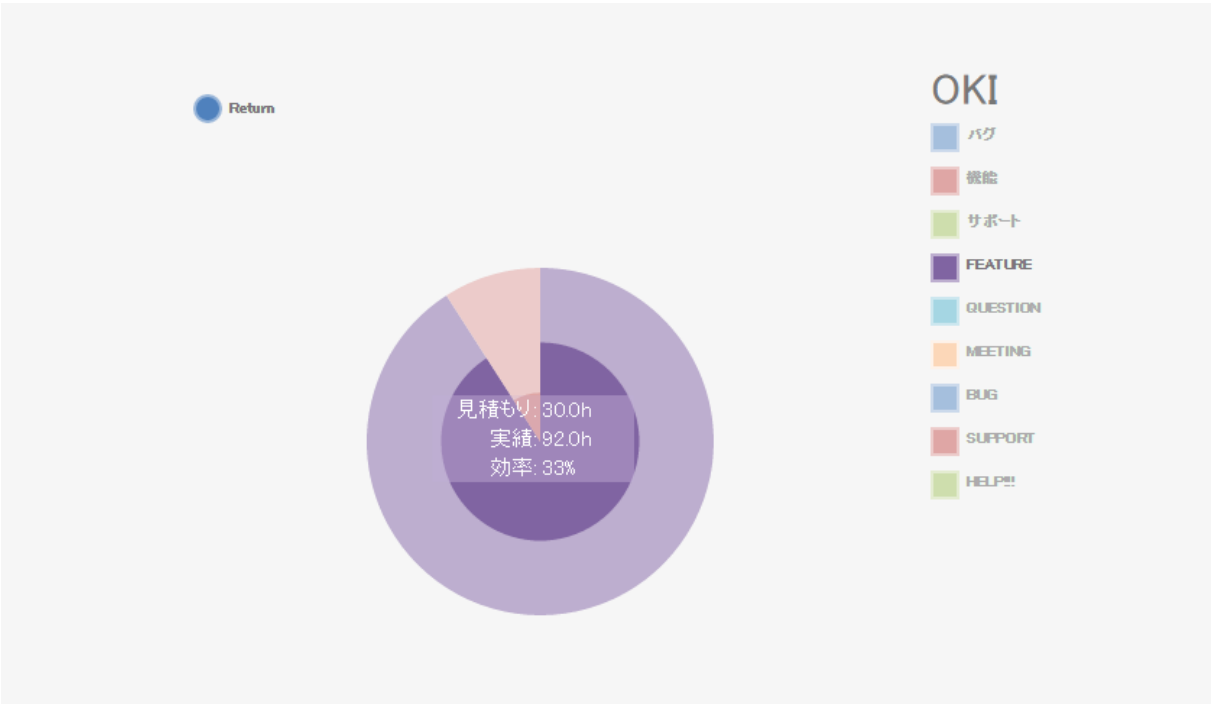


図 2-11：作業効率メトリクスの可視化モジュール

システムユーザはまず「作業効率メトリクス」の可視化モジュールの中から考察したい開発者を選択しクリックすると,選択された開発者に対する作業効率が円グラフで表示される．この円グラフは Redmine 上のチケットのトッラカーごとに，色分けされている．円の半径で，各トッラカーのチケットの合計予定工数と実績工数の関係が表示される．マウスを円グラフの上に移動すると，カバー部分に対応している，画像の隣の標識が同時に強調される．円グラフの中央部分に，該当トッラカーのチケットに対する作業効率が表示される．その部分をクリックすると，グラフは以下の画面に変更する．

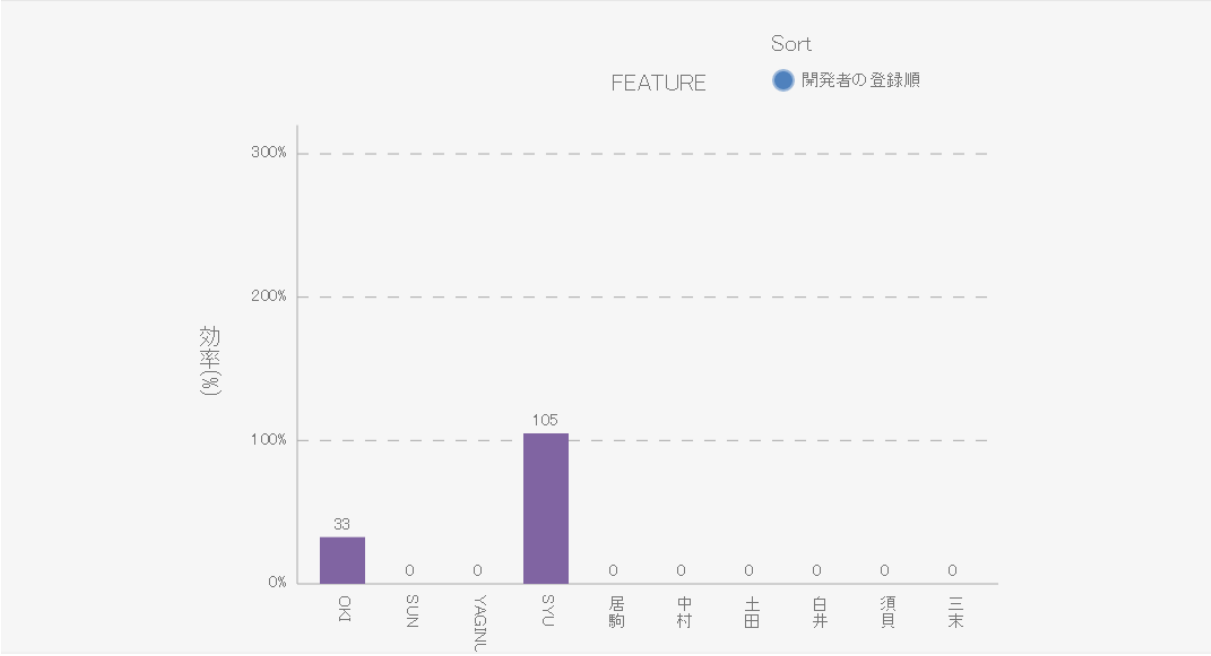


図 2-12：特定トッラカーのチケットの作業効率

このグラフで同じチーム内の他の開発者がこのトッラカーのチケットに対する作業効率が見比べることができる．横軸は各開発者の名前，縦軸は作業効率である．このグラフを利用し，開発者の得意分野も理解し易い，チーム内の開発者がこの種類のチケットに対する作業効率も見える．棒グラフをクリックすると，また該当開発者の円グラフに戻る．

システムユーザが「作業効率メトリクス」の可視化モジュールを利用し，考察したい開発者が Redmine 上で担当したチケットに対する作業効率が考察でき，該当開発者様々な種類のチケットに対する消化能力も理解でき，得意分野が推測できる．例えば，BUG というトッラカーのチケットの作業効率がチーム内で高いことが分かれば，該当開発者が全体と比べると，debug 能力が高いことが分かる．同じ種類のチケットに対する複数の開発者の能力も見比べられる．

メトリクスの処理の流れとして,「メトリクスを見る」画面に遷移し，プロジェクトを選択し,「選択」ボタンを押すと，裏側での処理が始まる．内部処理結果を画面に伝送し，ブラウザ上でグラフを生成し，システムユーザに描画する．もし内部処理が問題ある場合，画面上で,「メトリクスデータ取得失敗した」と言うエラーメッセージが表示される．

また,他のプロジェクトのメトリクスを見たい場合,また画面上のプルダウンリストから,他のプロジェクトを選択してから,「選択」ボタンを押すと，画面上の全てのグラフが自動的

に消える，また新しいメトリクスのグラフが生成される．

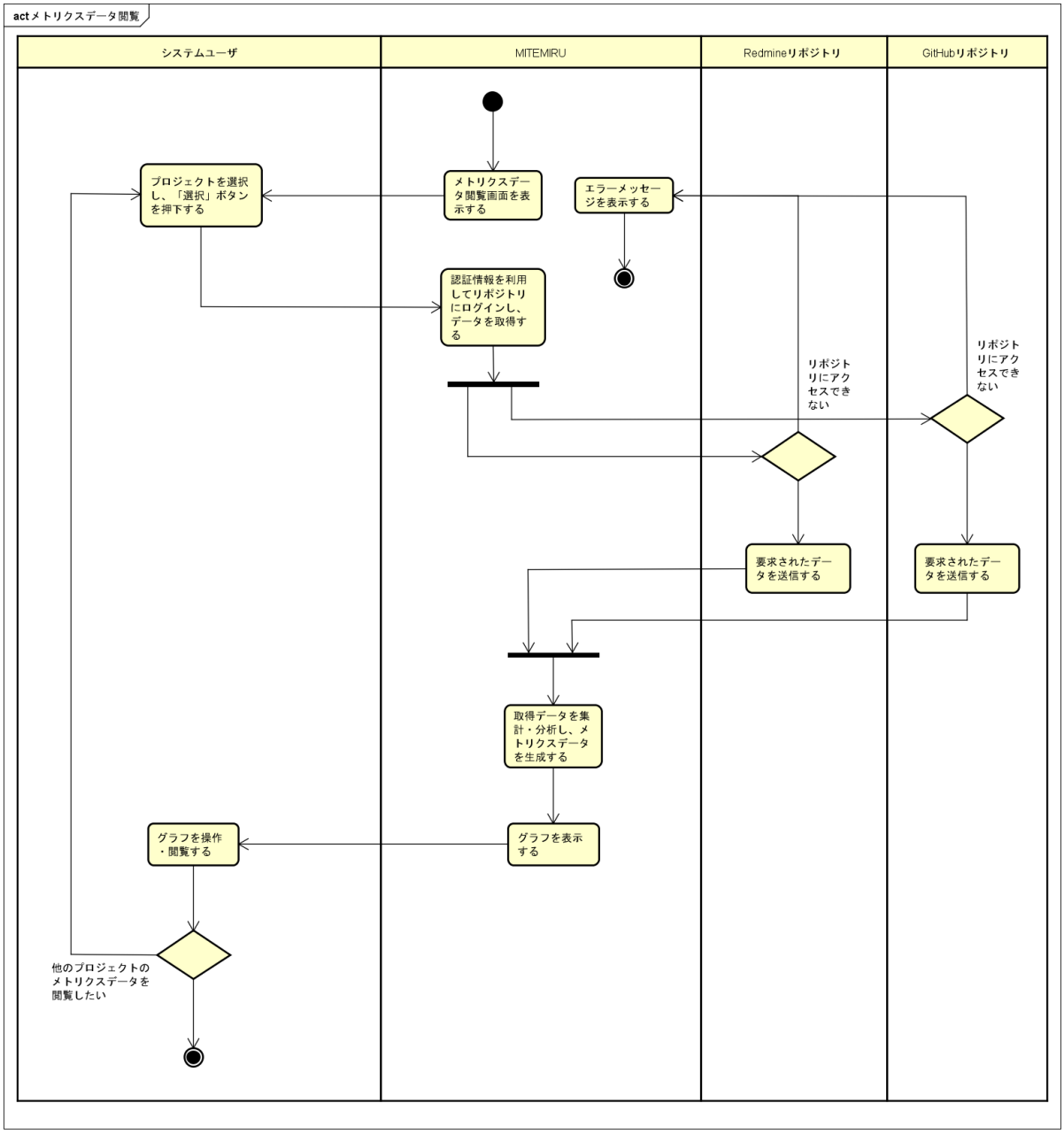


図 2-13：メトリクスデータ閲覧アクティビティ図

第3章 本システム提案手法の決定

3.1 類似システムの調査

本プロジェクトの設計段階で既存の類似システムの調査を行った.

調査の目的は：

- 類似システムの利用者はどんな立場の人であるのか
- どんな価値が利用者に提供できるか
- 実際にどこから，どんなデータを利用しているか
- 分析した結果はどのような手法で表現しているか

以上の目的を中心として，類似システム「EPM-X」と「Metrics Viewer」を発見した.

3.1.1 プロジェクト定量化ツール「EPM-X」

EPM-X は IPA が公開している定量的プロジェクト管理ツールである. EPM-X はリポジトリからデータを取得し，可視化モジュール経由で，システムユーザにリポジトリデータのグラフを表現することができる．EPM-X が提供するグラフはプロジェクト全体としての潜在リスク，進捗状況等のプロジェクト管理情報であり，プロジェクト内のソフトウェア開発者の情報が提供できない．しかし，本プロジェクトはソフトウェア開発者のパフォーマンスを中心とする立場が違う．かつ EPM-X は Redmine/Trac と連携している仕組みであるため，Redmine/Trac と関連がないプロジェクトに利用できない問題を発見した．汎用性が低いため，本プロジェクトの目的も相違である．この文献調査で EPM-X から参考できる部分と参考できない部分を表 1-1 に示す[3].

表 3-1：EPM-X 調査の結果

参考になる部分	参考にならない部分
可視化モジュールを利用し，リポジトリデータを表現する手法	Redmine/Trac と連携している仕組み（リポジトリが固定された問題）
リポジトリと連携し，データを利用すること	プロジェクト内ソフトウェア開発者の関連情報がないこと

3.1.2 個人の振り返りツール「Metrics Viewer」

Metrics Viewer は Metrics WebAPI を中核としてのクライアント側の Web アプリケーションである．Metrics Viewer でソフトウェア開発者が手軽に自分の開発行動履歴を検索でき，それに基づき，参加したプロジェクトに対する個人振り返りことができる．データの可視化部分は Google Chart Tools を利用し，さまざまなグラフを生成した. しかし, Metrics Viewer はソフトウェア開発者の視点から作られたシステム, プロジェクトの管理者の視点ではない. かつ，自分以外の開発者の情報が参考できないため，同じチーム内の開発者と見比べられない問題がある．この問題に関する，本プロジェクトとの目的とは違う．この文献調査で Metrics Viewer システムから参考できる部分と参考できない部分を表 1-2 に示す[7].

表 3-2：Metrics Viewer 調査の結果

参考になる部分	参考にならない部分
個人データをグラフで可視化する手法	リポジトリデータと関連なかった
リポジトリが固定されてないこと	開発者個人向けアプリケーション

「EPM-X」と「Metrics Viewer」二つの類似システムは，本プロジェクトの目的と違う部分があり，参考になる部分を洗い出した，本プロジェクトで活用することにした.

3.2 文献調査

本プロジェクトの設計段階で,「開発者貢献度」,「開発者評価」をキーワードとして，インターネット上で既存の論文と文献を調査した．調査した結果は「開発履歴を利用した風林火山モデルに基づく開発者特性の分析」という論文を発見した. この文献で,「風林火山モデル」を紹介した. 具体的説明すると, この文献であるプロジェクト内の開発者を中心として, 様々な方面から各開発者の能力を評価する手法を提案した[8].

表 3-3：風林火山モデル

分類	特徴
風	機能設計能力，実装の能力
林	突発的なトラブルへの対応性，問題解決能力
火	新しい技術，ツールの導入と運用能力
山	システムテストの能力，機能テストの能力

以上の四つの方面から，実際のプロジェクトのデータを利用し，開発者の能力を分類，評価した．最後，この四つの方面から，開発者能力を示すグラフを生成し，更に直観的に開発者の特徴を示した.

この文献から，本プロジェクトは開発者能力を分類する部分と可視化部分を参考し，特有の開発者を評価するメトリクスを設計した.

3.3 本システムの提案手法

類似システムと文献を調査した上で，本システムはリポジトリのAPIを利用し，リポジトリ内部のデータを取得し，裏側でデータを分析，整理，整形した後，開発者の「主体性」,「発言力」,「役割担当」，と「作業効率」四つの属性を表現することにした．それぞれの属性に対するブラウザ上でグラフを生成し，システムユーザはグラフを考察し，各開発者の能力，パフォーマンスを評価することができる．考えたメトリクスは表3-1に示す.

表 3-4：メトリクス表現手法

メトリクス	メトリクスの表現手法
主体性	開発者の自発の活動と関わり, 実際の活動履歴からプロジェクトに対してコミットの回数を抽出する. 回数が多ければ, 開発者が自分が担当している機能に対する関心が高い，品質を重視し，開発者の主体性が高い.
発言力	開発者はプロジェクトの関心度と関わり, 実際のチケットに対して発言回数が多くなければ, システムの各機能の設計, 実装の討論が積極的に参加でき, プロジェクトの関心度が高いことが示す.
役割担当	担当したチケットのトッラカーの種類から該当開発者の役割が推測できる.
作業効率	開発者の実装能力と関わり，担当したチケットデータから，各チケットのトッラカーごとに，合計予定工数と合計実績工数を比較し，開発者作業効率が判断できる．実績工数が予定工数より短い場合，実装効率が高い，予定より早めにチケットを消化できることを示す. つまり該当開発者は該当トッラカーのチケットを消化した時, 効率が高い. 実績工数が予定工数より長い場合, 実装効率が予定より低い, 該当開発者は該当トッラカーのチケットを消化した時，効率が低い，該当種類のチケットの消化は得意ではないことを示す.

四つのメトリクスを考えた後，各チームメンバは構成管理ツールとチケット管理ツールから取得できるデータの内容をまとめた上で，実際のリポジトリデータを利用し，各メトリクスは本システム上での実現方法を設計した．三つのメトリクスの設計の提案は表3-2に示す.

表 3-5：メトリクスの設計の提案

メトリクス	説明	参照リポジトリ	メトリクスデータ
主体性	自主的な活動, 実績	構成管理ツール	各開発者のコミット数
発言力	プロジェクトを推進するため, 発生したコミュニケーションの回数	構成管理ツール	他の人のチケット, あるいは作業内容等にコメントした回数
役割担当	開発時，担当したチケットのトッラカーの種類	チケット管理ツール	担当したチケットをトッラカー情報
作業効率	チケットの実績工数と予定工数の関係	チケット管理ツール	担当したチケットをトッラカーごとに, 予定工数と実績工数の合計

メトリクスの設計部分を完了後，各メトリクスに対する具体的の実現手法を考え，リポジトリからデータ取得手法を研究し，課題担当教員から可視化の手法を学習した上で，メトリクスの可視化モジュールの設計をした．各メトリクスの可視化モジュールは表3-3に示す.

表 3-6：メトリクスの可視化手法

メトリクス	メトリクスデータ	導出手法	可視化モジュール
主体性	コミット数	GitHub API経由で，開発者の全体コミット数を利用する	棒グラフ
発言力	コメント数	GitHub API経由で，各Issueの中のコメント履歴を開発者ごとに分類し，コメント回数を集計する	ヒートマップ
役割担当	チケットのトッラカの種類	Redmine API経由で，開発者が担当したチケットのトッラカー情報を取得し，集計する	円グラフ 棒グラフ
作業効率	チケットの予定工数、実績工数、トッラカの種類	Redmine API経由で，開発者が担当したチケット情報を取得し，トッラカーごとに分類し，各トッラカーに対してチケットの予定工数と実績工数を合計する	円グラフ 棒グラフ

第4章 プロジェクト概要

年間開発スケジュールを図 3-1 に示す. 開発・実装期間は反復型開発手法でを取り入れる. 中間報告 1 までのプロジェクトの進捗は主に開発構想書の執筆と環境構築，文献調査である [2][3]. 中間報告 2 まで，第三スプリントを実施し，主要な機能の実装が完了した. その後，第四スプリントで，開発作業とエンドユーザテスト同時実施し，実際の顧客，筑波大学他の先生からフィードバックをもらった.

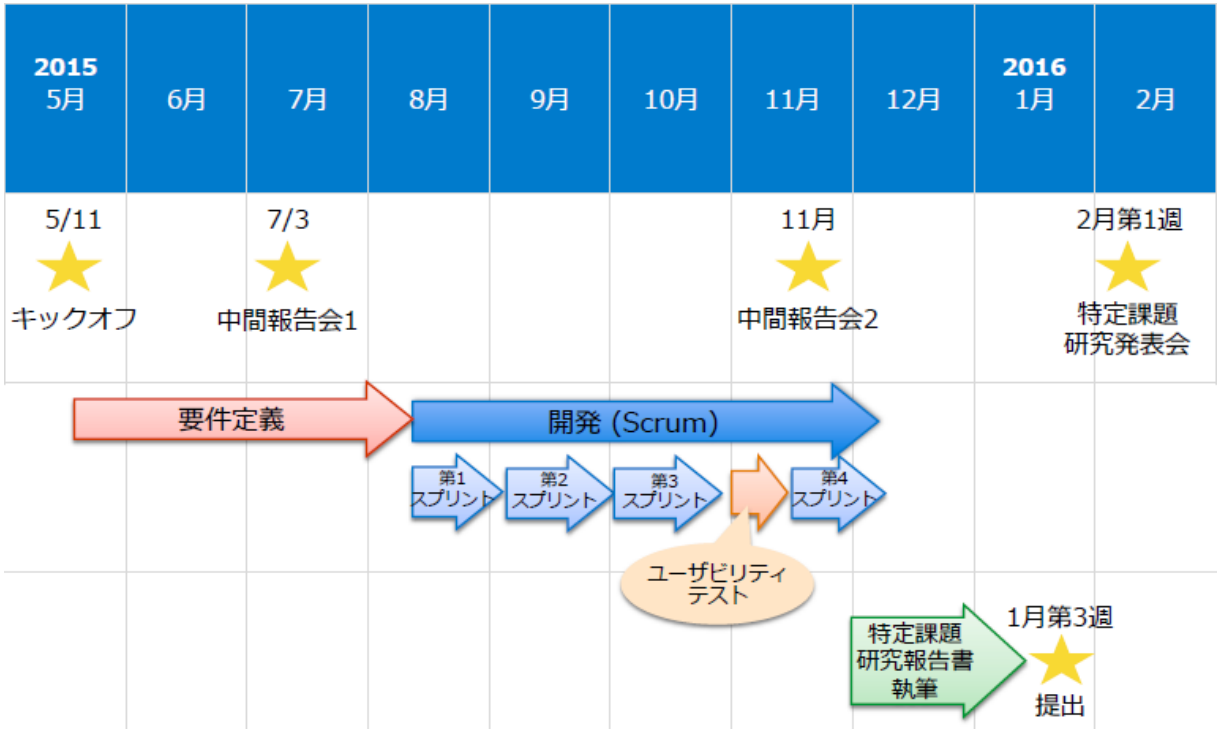


図 4-1：プロジェクトスケジュール

4.1 作業のコアタイム

本プロジェクトの最初の段階で，チーム作業のコアタイムをチームルールとして決めた，そして，その後の四つのスプリントで毎日同じ時間帯で各チームメンバが自分のタスクに集中し，作業をことのした.

原則として，午前 10:00~12:00，午後 13:00~18:00 は作業のコアタイムである. もし個人の予定が入る場合，できるだけ，コアタイムに影響をしないようにする方針を決めた.

4.2 顧客とのミーティング

本プロジェクトが開始する前に，顧客日立と相談し，毎週の金曜日午後 3 時から 4 時までの間，毎週のチーム進捗報告，プロジェクトの管理について，ミーティングすることを決めた. 毎回のミーティングで，顧客が開発チームが今週のベロシティ，ポイント消化数について質問し，もしプロジェクトの管理に問題がある場合，適度にアドバイスをし，開発チームが顧客の指摘に従って，今後のプロジェクト管理に活用する.

ミーティングの時，WebEx と言うオンラインミーティングツールを利用し，開発チームの画面を顧客にシェアし，成果物を報告する．

4.3 システム開発体制

本プロジェクトの顧客は株式会社日立製作所(日立)，課題担当教員は三末和男教授である．本プロジェクトは顧客と課題担当教員の指導の下で筑波大学の学生がシステムの開発を行う．そして，開発途中，日立からソフトウェア開発分野，三末和男教授から情報可視化分野のアドバイスを受けながら，進めた．課題担当教員，顧客，各チームメンバの役割は以下に示す．

表 4-1：システム開発体制

所属	役割	名前
筑波大学	課題担当教員, アドバイザー	三末 和男
	開発チーム	大木 遼太
		柳沼 工也
		孫 嘉成
		鄒 一民
株式会社 日立製作所	顧客, アドバイザー	居駒 幹夫
		土田 正士
		白井 明
		中村 宇佑
		河野 哲也
		須貝 佳彦

4.4 チーム運営の役割分担

表 4-2：チーム運営の役割分担

名前	役割
大木 遼太	チームリーダー スクラムマスタ
柳沼 工也	サブリーダー ドキュメント管理
鄒 一民	スケジュール管理 議事録担当
孫 嘉成	チケット管理 ベロシティ等の計算

4.5 （開発チームの）利用ツール

4.5.1 GitHub

本システムを開発する時，GitHubをバージョン管理ツールとして利用した．本システムのソースコードや関連するドキュメントをGitHub リポジトリ上に保存した．

4.5.2 Redmine

本システムを開発していた時，チケット管理ツールRedmineを利用し，各開発者のタスクをきちんと分割し，進捗を管理した．開発チームの開発ルールと開発の時利用するドキュメントが全部RedmineのWikiに掲載されている．

4.5.3 Dropbox

Dropbox を利用し，調査した文献，機能設計時時に作成した図，ドキュメント等，チームでの活動において生成されるドキュメント等を管理した．課題担当教員，顧客にシェアする必要があるドキュメントも独立フォルダに保存されている．

4.5.4 Slack

チーム内でコミュニケーションする時，Slack を利用し，不明な問題の討論，先生からのアドバイス，顧客とのやり取り，毎日のDaily Meeting等を行った．

4.6 実装した機能のレビュールール

本プロジェクトを開発する時，GitHub を利用し，あるチームメンバが自分のタスクを担当し，既存の機能を修正したい，あるいは新しい機能を開発したい場合，まず新しいブランチを作り，そこで，実装作業を実施する．全部の実装作業が終わるまでではなく，一回途中で，ある程度の内容をコミットし，GitHub 上でそのコミットに対する新しい pull request を作成する必要がある．該当 pull request のタイトルを命名する時，最初が「[WIP]+開発機能」と言う書き方で，具体的の機能が開発中と言う意味で，みんなの前に示す必要がある．

機能を実装してから，該当チームメンバが先ず，自分の pull request のタイトルを「[IR]+開発機能+開発時間」に変更する必要がある．[IR]は，他のチームメンバに該当機能の開発はもう完了したことを示すと言う意味である．次は，該当チームメンバは pull request で，自分が今回の実装に対するやったこと，実装した結果と他のチームメンバに対するレビューの流れを説明する必要がある．最後，該当メンバは他のチームメンバにレビューのお願いを伝え，他のチームメンバが自分は既に書いたレビュー流れに従い，レビューを行う．

全ての新しいソースコードに対する，Master ブランチにマージする前に，該当 pull request が他のチームメンバが二人以上レビューする必要がある．レビュー者が実際に機能開発者のブランチの中に移動し，自分の開発環境で機能を運行し，機能がちゃんと動いているかどうか，既存の他の機能に影響があるかどうかを確認してから，もし現時点の Master ブランチに影響がなければ，該当機能の GitHub 上のチケットにコメントし，Master ブランチにマージする．

4.7 可視化技術に関するレビュー

第一スプリントで，ViBi チーム全員が課題担当教員（三末和男教授）の可視化授業を受けた．課題担当教員が授業で従来の可視化手法を紹介し，既存のリポジトリマイニングシステムの可視化モジュールを例として挙げながら，ViBi チームに優秀な可視化手法を説明した．第三スプリントで課題担当教員と本システム既存の可視化モジュールに対し，三回ミーティ

ングをした．ミーティングで，課題担当教員が既存の可視化手法を指摘した．例えばコミット数メトリクスの中の数字は本システムの利用者にどう役に立つかと指摘し，コミット数グラフに対するいくつの案をアドバイスしてくれた．ViBi チーム全員は先生のアドバイスを参考する同時，実装可能性を検討しながら，より見やすいグラフの設計を修正した．その後，先生の考え方から発想し，D3.js を使い，メトリクスのグラフを修正した．

修正した結果としては，各メトリクスの表示手法を修正した．修正した点は：

表 4-3：グラフ修正一覧

	修正前	修正後
主体性メトリクス	単一開発者のコミット数が表示できる	チーム全員のコミット数が表示できる
	コミット数があっても，比較対象がない	平均コミット数が表示され，各人が平均コミット数に対する比較できる
発言力メトリクス	ある開発者が自分以外の開発者にコメントした回数だけ表示できる，数字だけで，システムユーザとして，情報が足りない	開発者全員がお互いにコメントした回数が表示できる．また色の濃淡で，開発者のコメント頻度が表示でき，活躍している開発者がすぐ見つけられる
	ソート機能がない	ソート機能があり，コメント頻度の順位がすぐ分かる
作業効率メトリクス	単一開発者の Redmine リポジトリ上で消化したチケットの見積もり工数と実績工数情報が表示できる	開発者全員の情報が表示でき，円グラフの中の円の半径で，見積もり工数と実績工数の関係が表示できる

4.8 第一スプリントの実績

第一スプリントの中に主にシステム機能の設計，ruby の学習，メトリクスの検討，データベースの設計，と文献調査を行った．文献調査の結果をまとめ，文献から考察項目を洗出し，分類し，沢山の角度から開発者を評価するリストを作成した[10].

第一スプリントの最後に，Redmine リポジトリの登録機能と.git ファイルのアップロード機能の実装を完了し，登録した Redmine リポジトリ一覧画面の作成とリポジトリ情報の更新，詳細を見ると言う各種管理機能を実装した[9].

4.9 第二スプリントの実績

第二スプリントに入り，主に開発作業を行った．第二スプリントの中に，二つグループを分け，一つのグループはメトリクスの可視化部分を着手する，もう一つグループは既に実装したリポジトリ登録機能と.git ファイルのアップロード機能を修正し，プロジェクト登録機能に変更する作業を行った．

第二スプリントの最後，プロジェクト登録チームの成績としては，プロジェクト登録の流れを変更し，Redmine プロジェクトを登録してから開発者を登録し，その後 Redmine プロジェクトに対する github リポジトリを登録する流れに変更した．メトリクス可視化チームは github リポジトリと Redmine リポジトリからデータを取得し，指定した開発者のコミット数，他のメンバにコメントした回数，Redmine 上担当したチケット数，Redmine チケットの工数メトリクスを実装し，ブラウザ上で表示できるようにした．

第二スプリントの最後，顧客レビューの時，実際に大学の教員あるいは PBL の他のチームに，本システムを実際に利用するユーザビリティテストを実施したいと言う要望が指摘し，第三スプリント終了後実施する予定である．

4.10 第三スプリントの実績

第三スプリントでシステム上のプロジェクト登録流れ再び変更し，プロジェクトを作成した上で，そのプロジェクトに対する Redmine リポジトリの URL あるいは github リポジトリの URL を同時に作成し，認証情報を入力してから，またデータベースにリポジトリの情報，開発者情報を保存する仕組みに変更した．かつ各入力欄のバリデーションをつけ，入力内容が不具合場合，エラーメッセージが画面上で表示し，システムを簡単にできるように設計し，実装した．

プロジェクト登録画面を図 4-2 に示す．

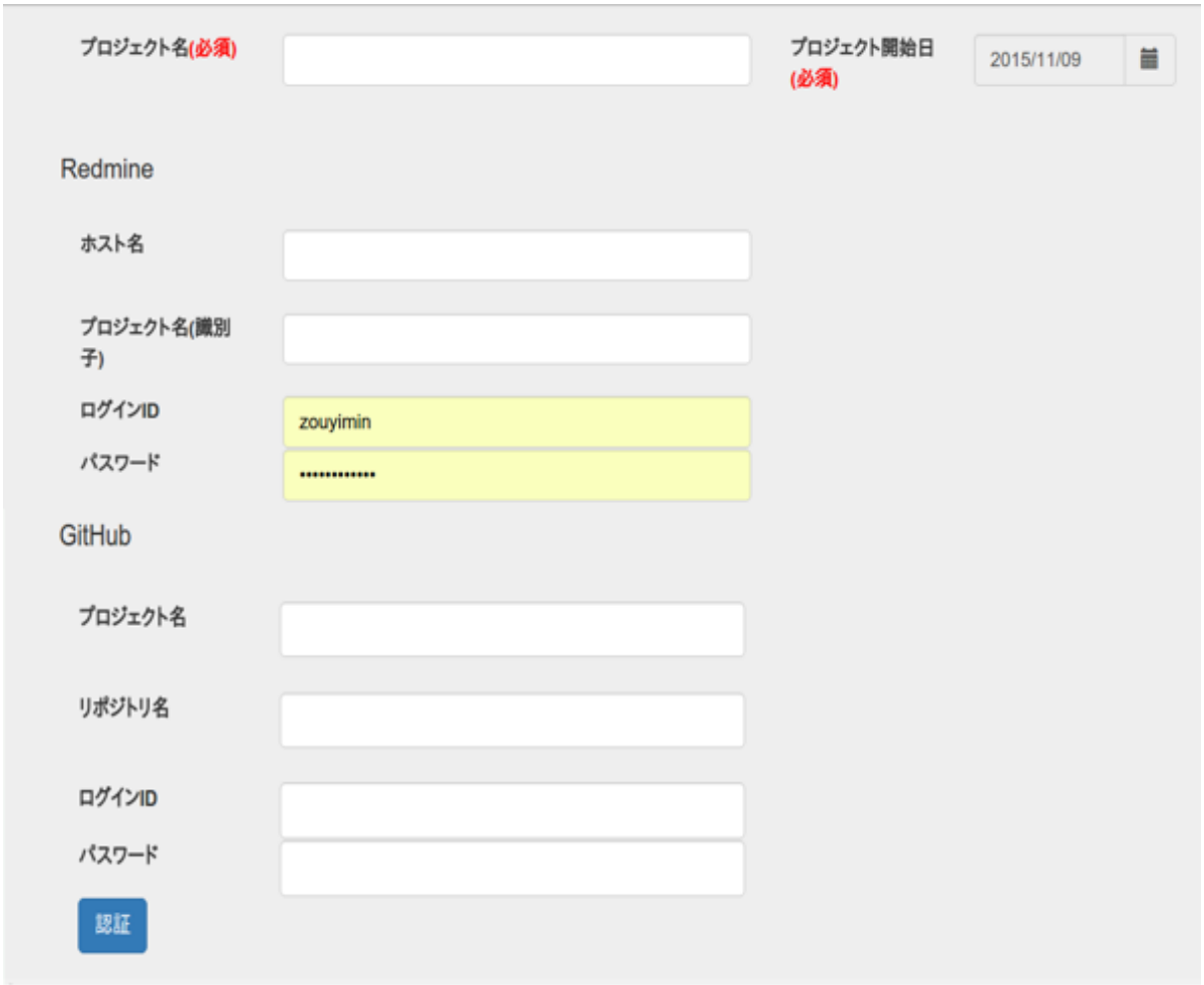


図 4-2：修正したプロジェクト登録画面

そして各種管理機能の中に，既に登録したプロジェクトの情報，Redmine と Github の認証情報，開発者の情報，システムユーザの情報を管理する各種管理機能を修正した．
メトリクスの方は，redmine 上担当したチケット数メトリクスを削除し，残りの三つメトリクスの可視化モジュール部分を修正してから，一ページで表示できるように修正した．そして GitHub API 経由で取得できる情報を参考しながら，新しいメトリクスを洗出し，顧客に提案した．

4.11 第四スプリントの実績

第四スプリントでやったタスクは主に第三スプリントまで実装できなかった機能，システム説明ドキュメントの作成，エンドユーザテストの準備と実施を行った．

システム上の具体的の機能としては，まず，GitHub API から，各開発者のコミットデータを集計し，各コミットデータに対する，編集したファイルの拡張子を取得した上で，チーム内のすべての開発者が編集したファイルの回数を統計できる「専門性メトリクス」を実装した．

次は，プロジェクトの登録フロー部分，あるプロジェクトを主体として，Redmine リポジ

トリと GitHub リポジトリを登録する時，Redmine リポジトリの URL と GitHub リポジトリのチーム名，プロジェクト名を登録する仕組みに修正した．

ドキュメント部分は，顧客日立に提出する「開発構想書」の整理と Redmine リポジトリ上の Wiki 部分の整理，プロジェクト開発に必要な「ER 図」，「システム構成図」，「ユースケース図」と「アクティビティ図」を将来研究課題報告に使えるように整理した．

第四スプリントでエンドユーザテストを行った．日立と筑波大学の先生にテストアンケートを送り，実際のシステムを利用していただき，様々な視点から，開発チーム以外の人がこのシステムに対する感想，フィードバックをもらった．そして，第四スプリントの最後段階で，開発チームがアンケートの結果を集計，整理し，問題点を洗出してから，各問題に対するチケットを発行し，実装をした．

第5章 エンドユーザ評価テストの実施

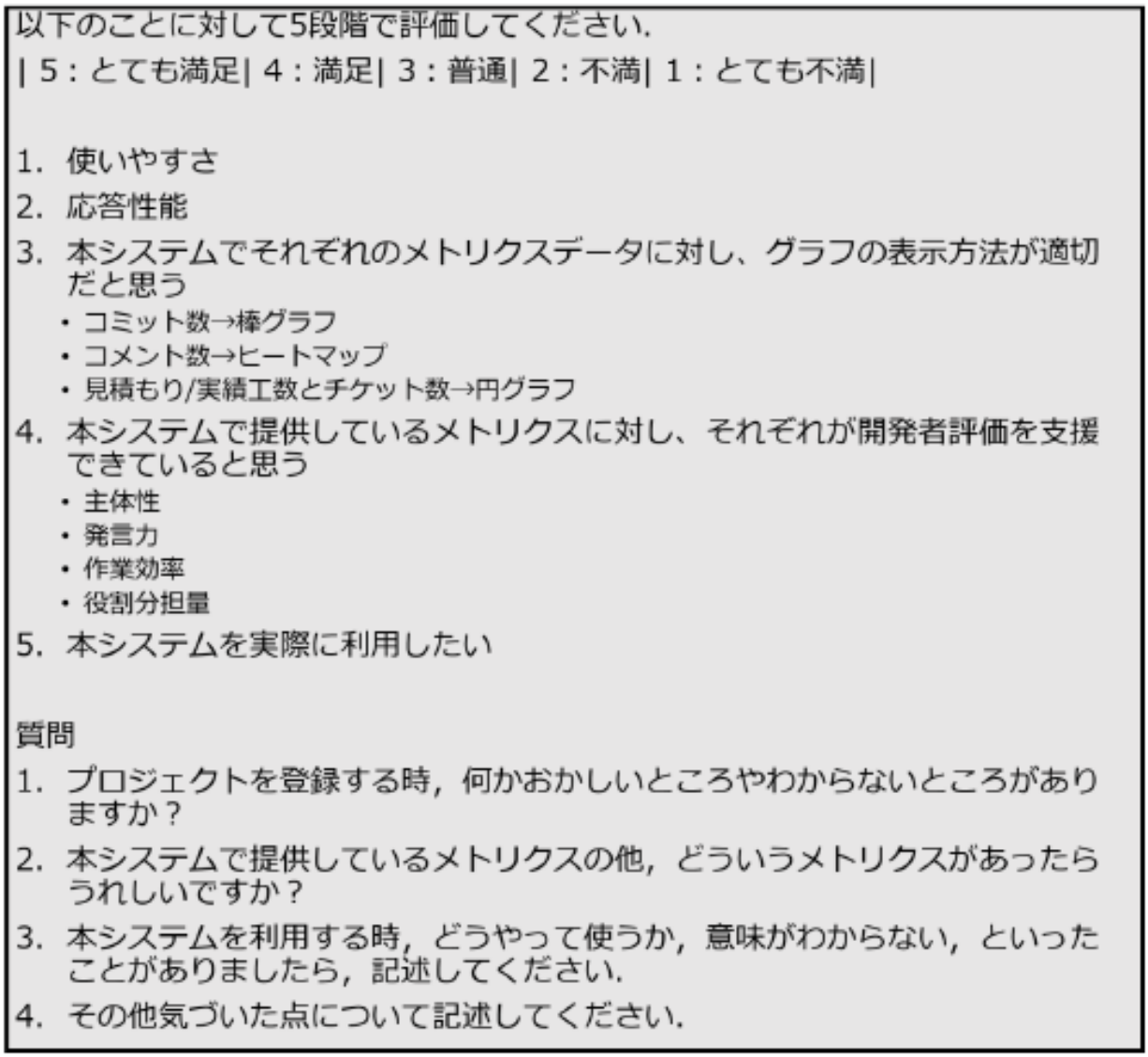
5.1　　エンドユーザ評価テストの概要

第四スプリングで，本システム上で既存の機能と各機能に対するシステムユーザに開発者の評価を支援する価値があるかどうかを図るため，実際にMITEMRUシステムを利用してもらい，システムユーザのフィードバックを聞くを目標として，エンドユーザ評価テストを行った．事前に用意してあるRedmineリポジトリとGitHubリポジトリを利用し，評価テストで各機能を利用してもらう．メールやSlackの通信手段でアンケートの実施案内をエンドユーザに送るという配布形式で，各エンドユーザが自分のローカル環境のブラウザでクラウドサーバーHeroku上のMITEMRUシステムを利用し，アンケートを答えることである．

エンドユーザー評価テストの対象者は，筑波大学の高度ITコースでPBLプロジェクトを管理する経験がある教員と日立製作所のプロジェクトマネージャー合計9人が評価テストの対象者である．テストの対象者は全員プロジェクトを管理する経験があり，MITEMRUシステムのユーザとして最も適切だと考えた．

テストの流れは，まず評価対象者はMITEMRUシステムを利用し，所有するプロジェクト情報をMITEMRUシステムに登録する．次に，登録したプロジェクトに対する生成したメトリクスのグラフを考察し，プロジェクト内の開発者を評価する．最後，アンケートに書いてある複数の問題を回答し，自分の利用感想を書く．

アンケートはシステムの使いやすさ,各機能の応答時間,グラフが表現する情報の適切さ,メトリクスの価値などの角度から評価項目を設置し，システムを利用する際の感想や意見などの質問も含めている．アンケートは図 5-1 に示す．


以下のことに対して5段階で評価してください.

| 5：とても満足| 4：満足| 3：普通| 2：不満| 1：とても不満|

1. 使いやすさ
2. 応答性能
3. 本システムでそれぞれのメトリクスデータに対し、グラフの表示方法が適切だと思う
   - コミット数→棒グラフ
   - コメント数→ヒートマップ
   - 見積もり/実績工数とチケット数→円グラフ
4. 本システムで提供しているメトリクスに対し、それぞれが開発者評価を支援できていると思う
   - 主体性
   - 発言力
   - 作業効率
   - 役割分担量
5. 本システムを実際に利用したい

質問

1. プロジェクトを登録する時，何かおかしいところやわからないところがありますか？
2. 本システムで提供しているメトリクスの他，どういうメトリクスがあったらうれしいですか？
3. 本システムを利用する時，どうやって使うか，意味がわからない，といったことがありましたら，記述してください.
4. その他気づいた点について記述してください.

図 5-1：エンドユーザ評価テストのアンケート

5.2 エンドユーザ評価テストの結果

テストの結果は以下の二つの点である，一つ目はシステムを利用する時発生した問題，不明点，二つ目はメトリクス可視化モジュールの表現手法の問題である．システムを利用する時，各機能の操作手順の説明文が不十分の問題とある一部の機能の応答時間が予想よりかかる問題が指摘された．メトリクスの可視化モジュールに対する問題は主にグラフで表現した情報の意味が理解しにくい，グラフに対する操作が分からない，グラフで表現できる情報が少ないという問題が指摘された．ViBi チームがそれぞれの指摘に対して，第四スプリントの最後，メトリクスの可視化モジュールとある部分の機能を修正した．

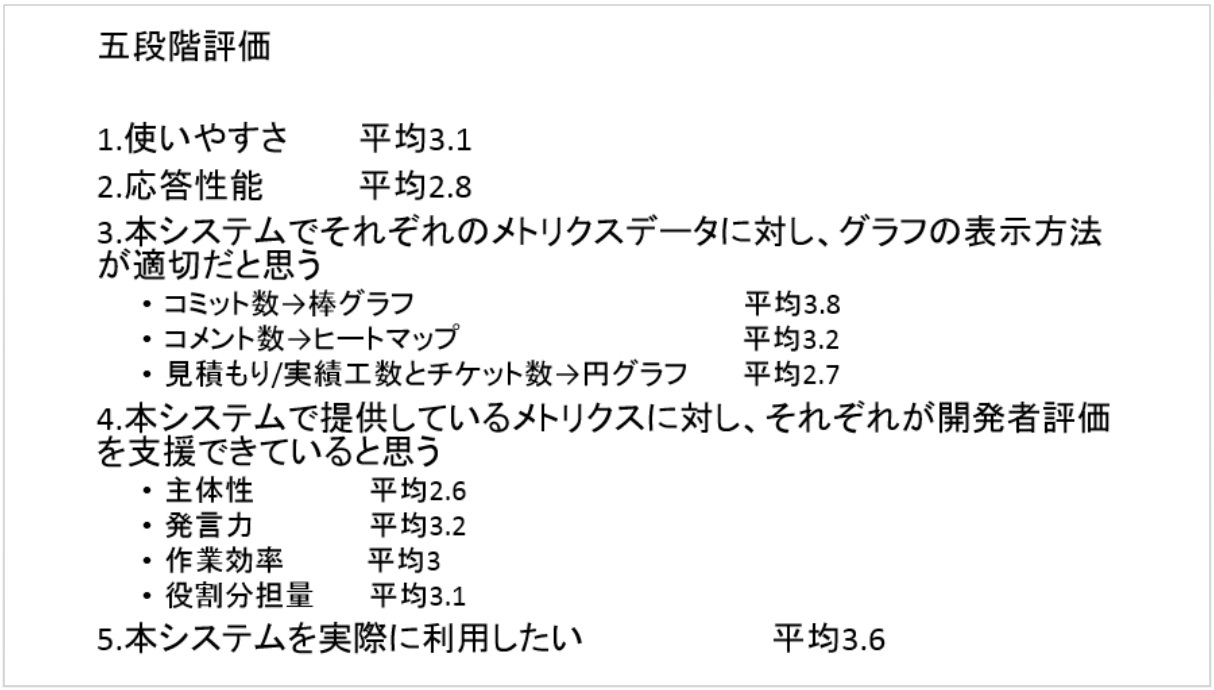


図 5-2：エンドユーザ評価テストの結果(1)

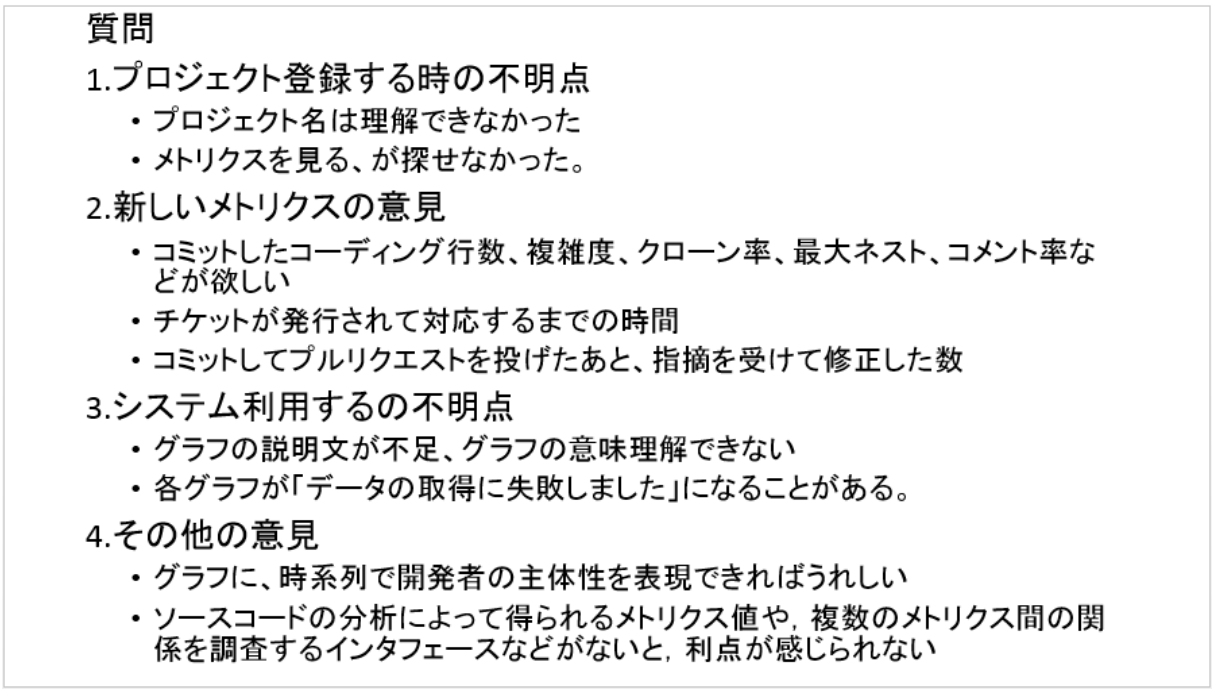


図 5-3：エンドユーザ評価テストの結果(2)

5.3　　結果に対する改善案

エンドユーザ評価テストの結果に対する改善案は表 5-1 に示す.

表 5-1：エンドユーザ評価テストの結果に対する改善案

問題	改善案
説明文の不十分	ヘルプページを作成する．システムユーザがシステムを利用する時，操作等不明がある場合，ヘルプページを参照し，不明点を解決する.
可視化モジュール提供できる情報の不足	メトリクスの可視化部分のユースケースを検証し，より多くのユーザが求める情報を提供できるようなデータを取得し，可視化モジュールを修正したり，自由な分析・評価を実行できる環境を実現する.

第6章 プロジェクトにおける筆者の役割

6.1 メトリクスに対する文献調査

第一スプリントで，本システムが開発者のどんな動きを定量し，どんな立場から開発者の能力を表現するか，取得したデータをどう利用し，メトリクス化にする問題に対し，チーム内の各メンバは「開発者能力判定」,「開発者貢献度」,「開発者モチベーション」,「開発者貢献度可視化」の様なキーワードを使い，インターネット上で文献を調査し，チーム内で共有した．その後，各メンバ発見した文献をまとめ，分類し，顧客に調査の結果を報告した．

筆者が調査した結果としては,「リポジトリのリアルタイムな可視化にもとづく PBL の支援環境」と言う文献を発見し，この文献の中にどうやってリアルタイムで PBL の各チームの進捗，編集しているファイルを表示する手法を説明した．この文献からそのシステムの可視化手法とデータを取得する手法を洗出し，他のチームメンバと共有した[8].

その後，文献「IZMI 開発者と成果物の編集頻度に着目したソフトウェア開発リポジトリの可視化ツール」からリポジトリデータを利用し，開発者のモチベーションの評価方法とリポジトリデータの利用方法を学習した．本システムでは github API と redmine API からデータを取得する仕組みもう事前に決定し,成果物の編集頻度の取得は実現可能性が低いけど,その一方，他のデータから，開発者の主体性を評価するやり方を考え，本システム上で活用できると考え，チーム内で相談してから，実際の機能「開発者コメント数メトリクス」に実現した[6].

第二スプリントに入り，システムの開発段階が始まり，本システムを機能範囲で二つのチームを分割した．一つ目はプロジェクトデータ，開発者データを登録と管理する機能を担当するチーム．二つ目は Github API，Redmine API を利用し，データを集計し，D3.js と言う JavaScript のライブラリで, データを可視化する機能を担当するチームである．筆者は二つ目のデータ可視化チームに参加し，GitHub と Redmine 上のデータを取り，加工集計するアルゴリズムの実装と分析結果を json 形式で js ライブラリに伝送する部分を担当した．

6.2 D3.js の学習

D3.js は多量の図を実現でき，かつ軽量的なライブラリである．実装段階に入り，筆者は主に可視化部分を担当する予定があるため，D3.js を導入した．D3.js で可視化モジュールを実装する必要があるため，インターネット上で D3.js の学習ビデオを見ながら，サンプルを作成し，D3.js の使い方を他のチームメンバに共有した．

6.3 発言力メトリクスの設計と実装

筆者はまず GitHub API と Redmine API のドキュメントを研究し, 両者から取得できる情報のスコープを決定した．それに基づく，新しいメトリクス「ある開発者がチーム内の他のチームメンバの issue にコメントした回数」を提案し，このメトリクスは開発者のコミュニケーション力と主体性が表現できと考え，チーム内で合意し，顧客日立に報告した．

そのあと，このメトリクスに対するブラウザ上のグラフ形式を考え，D3.js から多数のサンプルを参考しながらグラフ形式を決定した．以上の設計の部分をきちんと考えた上で，実際の開発作業に入り，データの取得とループ処理の部分を実装した．

この機能を実装した時，最も工夫した点としては，GitHub API の仕様として，一定時間内の接続制限がある．非認証の状態で，ユーザが本システムを利用する時，5 回以内メトリクスを実行すると，アクセス接続制限回数になり，リポジトリからデータが取れなかった問題が起こった．この問題に対して，筆者はまずチームに共有し，その後インターネットで解決策を検索し，既存のロジックを修正してから，アクセスが接続制限の問題を解決した．

実装段階で，Ruby の文法問題が発生した場合，筆者は積極的に他のチームメンバとコミュニケーションお諮り，問題を解決しつつ実装を進めた．

内部処理を実装した後，可視化部分の実装を始め，Javascript 言語を使い，メトリクスを実装し，裏側の内部処理部分を連動させ，新しいメトリクス機能を実装した．しかし実際運用する時，内部処理部分のアルゴリズムが複雑で，実行開始から処理終了までの時間がかなり掛かる問題が発生し，実際利用者がこのシステムを利用する時の体験感を影響しないように，チーム内で討論し，結論としてはページ側で ajax 手法を利用し，内部処理を呼び出す仕様に変更した．

この部分の可視化モジュールを実装した時，工夫した点としては，本来の D3.js が提供した類似グラフサンプルの中に，矢印がなかった．筆者は類似グラフサンプルを研究し，D3.js の他の実例を参考しながら，矢印の画像をグラフの中に追加した．

この機能の第三スプリントまでイメージは以下の図 3 の中の画面である．この図は Altairzym と言う対処開発者として，他のチームメンバへのコメント数を表している．コメント回数が多くなると，線は更に太くなる．

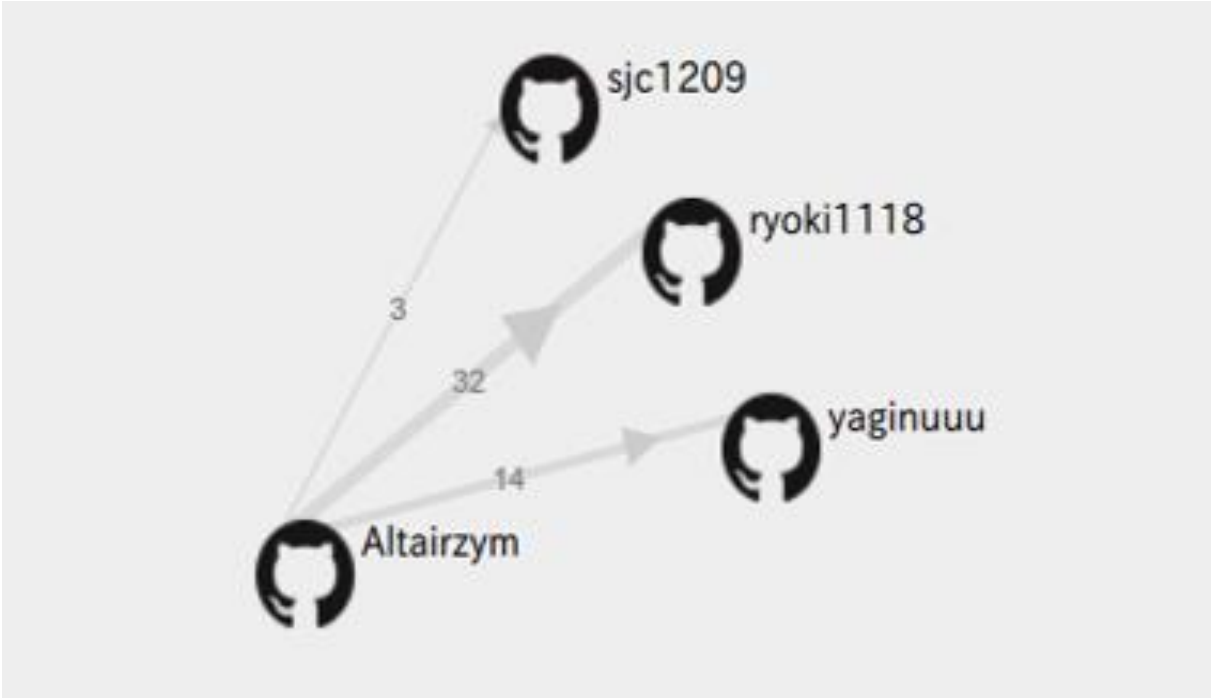


図 6-1：元開発者発言力メトリクス

6.4　　主体性メトリクス内部処理の実装

このメトリクスを実装する前に，既にこの機能のイメージ図と可視化手法をチーム内で共有したかつ可視化手法は他のチームメンバが担当する予定があるので，この機能について筆者の作業内容としては Github から各メンバのコミット情報を取得すると可視化モジュールと連動させる部分である.

工夫した点としては，実際の開発段階で，Github API 経由，二つのコミット情報を取得する手段があり,かつ二つの手法でも master ブランチにコミットした回数だけ記録したので，この問題についてチームに共有し，討論の結論としては，現時点のシステムは開発者が master ブランチにコミットした回数だけ表示することにした. 筆者はその後，リポジトリからデータを取得し，加工し，可視化モジュールとの連動を実装した. 機能のイメージは図 6-2 に示す.

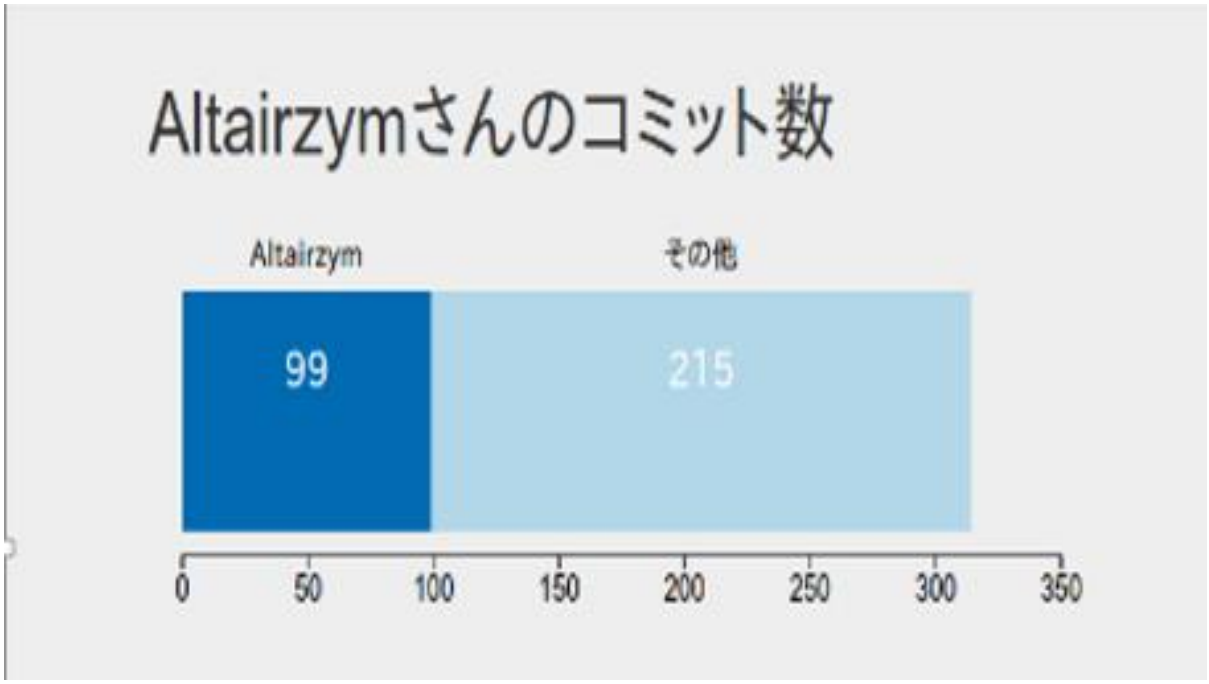


図 6-2：元の開発者主体性メトリクス

その後，顧客が一人の開発者のコミット数を表示ではなく，チーム内の他の開発者のコミット数を同時表示，見比べたいと言う要件があるため，筆者はまた要件に対応し，裏側の処理のロジックを修正し，機能を実装した.

第四スプリントまでの機能のイメージは図 6-3 に示す.

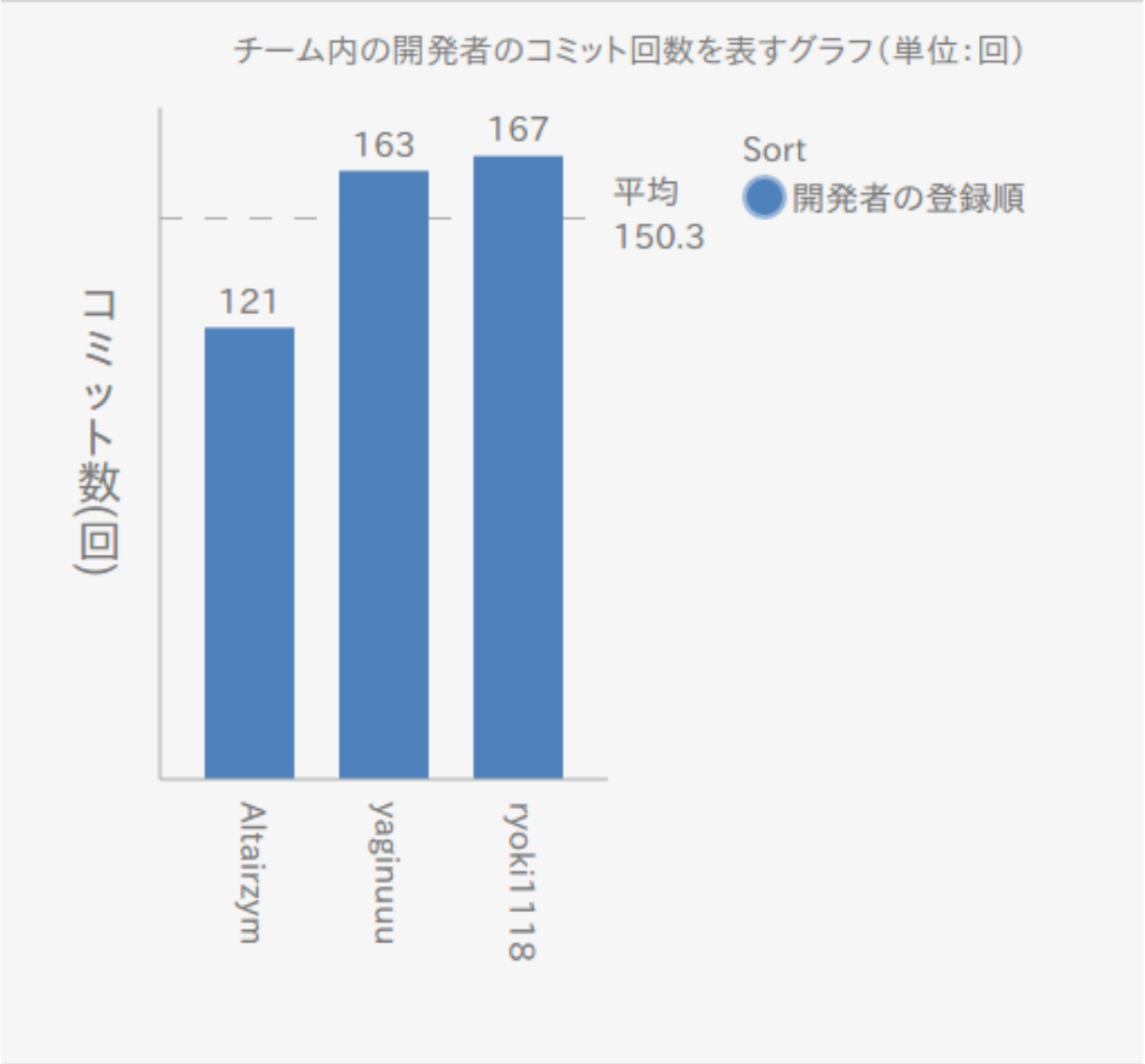


図 6-3：修正した開発者主体性メトリクス

6.5　作業効率メトリクス内部処理の実装

筆者は，このメトリクス機能のコントローラの実装部分を担当した．コントローラの実装は主に redmineAPI から各開発者が担当したチケットの種類，各種類のチケットの実績工数の計算，予定工数の計算，最後 json 形式の文字列を作成し，可視化モジュールに伝送するロジックである．

この機能を実装完了のイメージは以下の図 6-4 に，redmine のチケットのトラッカーごとに，予定工数と実績工数を示す．

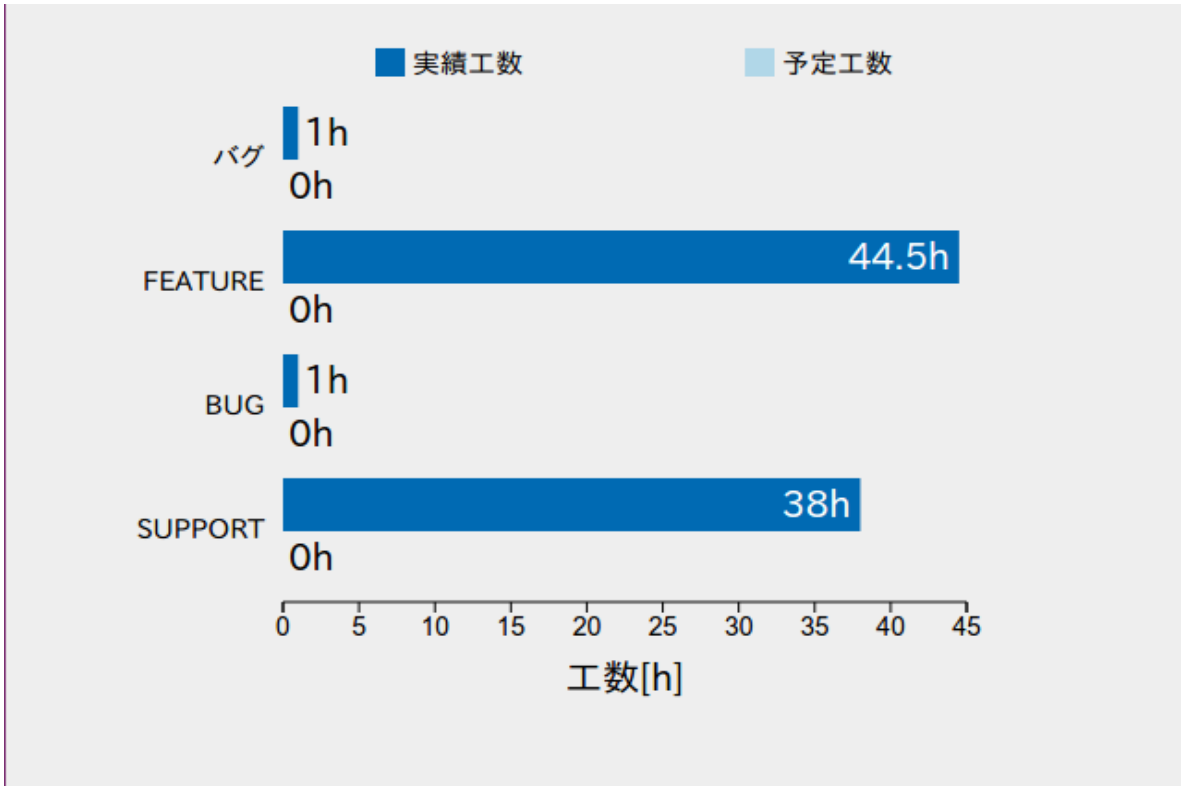


図 6-4：元開発者作業効率メトリクス

しかし顧客レビューの時，顧客の方から担当しなかったチケットの実績工数と予定工数を表示しないようにしたい，かつチーム内の他のチームメンバの担当したチケット状況も観察したいという要望があったため，その要望に対して，裏側の処理を修正した．

修正した成果としては，ある Redmine リポジトリの中のすべての開発者のトラッカーごとに，実績工数と予定工数を算出できるように実装した．

具体的なイメージは図 6-5 に示す．

図 6-5：修正した開発者作業効率メトリクス

この機能の工夫した点は，リポジトリからデータを取得し，配列を加工する部分である．生データは普通の配列だけど，裏側で，二次元配列に変更する処理を実装する必要が難しかった．

6.6 作業効率メトリクスの修正

このメトリクスは他のチームメンバが実装した．しかし筆者は Redmine API のドキュメントを読む時，Redmine API 経由でデータを取得する時，既に自動的にページネーションされたことを発見した．もしこの問題を修正しないと，一回だけ Redmine API に問合せすると，全てのデータが取得できない．この問題に対し，筆者は既存のロジックの上に修正し，ページネーションに対する処理部分を実装した．

第二スプリントの最後，顧客レビューの時，このメトリクスは一人の開発者に対する担当したチケット数は表現できるけど，他の開発者と見比べなければ，あまり価値がないと言う指摘をもらい，現時点このメトリクスに関する機能を全部削除した．

第四スプリントで，「担当チケットメトリクス」で表現したい情報が既に「生産性メトリクス」に含まれているため，「担当チケットメトリクス」を削除した．

6.7 グラフ一覧機能の実装

第二スプリントの最後以上の四つのグラフ機能は全部独立し，他のグラフを見たい場合，ページ遷移をする必要がある．遷移する回数が多くなると，利用者に対する体験感があまり良くないとチーム内で意識をし，第三スプリント内で，四つのグラフを一つのページにまとめるかつページの最初にプロジェクト選択リストを表示し，そこでプロジェクトをけっていしてから，グラフを描画すると言うタスクを決めた．そして筆者は第三スプリントで，この部分の修正作業を担当した．

まずグラフに関する全ての javascript ファイルを一つのページにまとめ，javascript の ajax

部分の URL の設定を変更した．次はコントローラ内部で，すべてのデータ収集，集計に関するメソッドを一つのコントローラに移行し, ruby の方でルーティングの設定の変更を実施した．最後は他の要らないファイルとルーティングを整理した．

プルダウンリストに対して，まず裏側で現在登録しているユーザが登録したすべてのプロジェクト一覧をデータベースから取得し，画面のプルダウンリストの中に表示できるように実装した．次はプルダウンリストとグラフの描画メソッドに連動できるため，画面上の「選択」ボタンを押した同時，ページ内の全てのグラフを削除した上で，javascript でグラフを描画するメソッドを呼び出し，描画できるように実装した．

第四スプリントで顧客から「メトリクス見る」画面で, 表示できないグラフの空間を残し，グラフが表示できない場合，その空間でエラーメッセージを表示したいと言う要望が発生した．筆者はこの要望に対し，画面上の svg 元素の固定，svg の枠線の作成とメトリクスが必要なデータが取得失敗した場合，画面上でエラーメッセージの表示機能を実装した．

6.8 Ubuntu 環境の Mysql 設定問題の発見

第三スプリントで，他のチームメンバの作業内容をレビューし，実際にプロジェクトのデータをデータベースに保存した時，筆者の方は文字化けのエラーが発生し，解決できなかった．しかし同じチームのもう一人は自分の開発環境で実行する時，そのような問題は全然なかった．その後，筆者とその作業の担当者はミーティングし，問題の解決方法を調査し，沢山の解決方法を実際に試してから，Ubuntu 開発環境で Mysql の設定は Mac の開発環境と違い，漢字を保存すると，文字化け問題は必ず発生すると言う結論が解明した．この問題を回避するため，Ubuntu を用い，開発しているメンバはできるだけ文字の入力を回避することにした．

6.9 専門性メトリクスの設計と実装

第四スプリントで筆者が専門性メトリクスの設計と実装タスクを担当した．この前顧客とのミーティングした時，顧客がチーム内の開発者が編集したファイルの内容，あるいは編集したファイルの種類から，開発者の能力を測りたい要望があった．この要望に対し，第三スプリントの最後，開発チームが専門性メトリクスを提案して，第四スプリントで実装する予定だった．筆者は実査に GitHub API から得られるデータの内容を調査し，専門性メトリクスの裏側処理のロジック部分とビューに伝送するデータの仕様の設計を考え，提案した．チーム内で合意を取ってから，実際にこのメトリクス裏側処理の実装を担当した．

このメトリクスの裏側処理を実装した時一番工夫した部分は，GitHub API が提供しているデータ仕様は複雑し，もし直接ある開発者のコミット情報だけ取得すると，コミット情報の中に，該当開発者が他の開発者の pull request を master にマージする履歴も含まれている，もしこの様なデータを無視すると，システムが他の開発者が編集したファイルが該当開発者に認証する可能性がある．この問題を解決するため，筆者が他のチームメンバと積極的に相談し，合意を取ってから，ロジック部分の設計を修正し，機能を実装した．

しかしこの機能は開発者が編集したファイルの拡張子の回数を集計する時，三段階ループの必要があるため，実際に運行すると非常に時間かかる問題が発生した．結局，第四スプリントの最後に master にマージしなかった．

裏側処理で集計した結果は以下の図の感じである．

{"developers":["Altairzym", "mikio-ikoma", "mitemiru-reviewer", "ryoki1118", "sj
c1209", "yaginuuu"],"extensions":[".rb", ".js", ".erb", ".css", "Gemfile", ".sam
ple", ".lock", ".coffee", ".env", ".swp", ".png", ".tsv", ".generators", ".name"
, ".rakeTasks", ".xml", ".iml", ".txt", ".csv", ".tmp", ".tmp2", ".scss", ".yml"
, ".pdf", ".json", ".gif", ".gitignore", ".md", "Rakefile", ".keep", ".jbuilder"
, "bin/bundle", "bin/rails", "bin/rake", "bin/setup", "bin/spring", ".ru", ".rak
e", ".html", ".otf", ".eot", ".svg", ".woff", ".ttf", ".woff2", ".ico", ".jpg",
".rspec"],"developers_edit":[[121, 83, 65, 4, 4, 3, 3, 4, 1, 1, 3, 1, 1, 1, 1, 8
, 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,
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, 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, 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, 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, 0, 0], [112, 116
, 111, 2, 2, 0, 3, 11, 0, 10, 0, 0, 0, 0, 0, 0, 0, 5, 2, 4, 4, 2, 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, 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, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [473, 137, 251, 29, 20, 3, 18
, 13, 1, 0, 111, 1, 1, 1, 1, 8, 1, 4, 0, 0, 0, 6, 36, 11, 13, 14, 10, 11, 2, 21,
 12, 2, 2, 2, 2, 2, 2, 2, 6, 2, 4, 4, 6, 6, 4, 2, 6, 1]]}

図 6-6：専門性メトリクスの裏側処理結果

この図の中に，開発者名リスト(developers)，すべての開発者が編集したファイルの拡張子リスト(extensions)，と各開発者が各拡張子に対すファイルの編集回数(dvelopers_edit)を全部記録していて，将来的に，円グラフを作成し，各開発者の得意分野を表示する予定だった．

第7章 終わりに

本プロジェクトにおいて，筆者らは顧客の要求に基づき，既存プロジェクト管理ツール「EPM-X」と「Metrics Viewer」の特長を参考の上，リポジトリデータを利用したソフトウェア開発プロジェクト管理者向け個人評価支援システムを開発した．

12 月 9 日に第四スプリントが完了し，プロジェクトの開発が終了した．本プロジェクトの準備段階において，筆者は類似システムの調査と文献調査を行った．開発段階において，筆者は主に「主体性メトリクス」,「発言力メトリクス」,「作業効率メトリクス」と「役割担当メトリクス」の実装機能担当した．具体的には，API 仕様の調査とそこからデータの所得方法，メトリクスデータの加工，そして加工した結果を JS に伝送する仕様の変換をした．筆者は本プロジェクトを通じて，Git，Redmine ツールに関する知識を学び，ruby を使用したシステムの開発とアルゴリズムの設計技術を磨いた．また，プロジェクト管理，品質保障に関する知識を学び，顧客やチームメンバーとのコミュニケーションを図った．プロジェクトの成功にとって，コミュニケーション力は非常に重要であることを体験できた．かつ，外国人としてどうやって日本人と一緒に協力し，プロジェクトを運営していくかについて学ぶことができた．本プロジェクトの経験を将来に活かすことができると考えている．

謝辞

本研究開発プロジェクトを行うにあたり，貴重な時間を割いてご指導いただいている株式会社日立製作所の居駒幹夫様，土田正士様，須貝佳彦様，白井明様，中村宇佑様，河野哲也様に心より感謝いたします．

委託元教員である三末和男教授から大変貴重なご助言，ご指導をいただきました．深く感謝いたします．

指導教員である田中教授には，様々なご指摘とご助言をいただきました．大学院入試から指導してくださった二年間，本当にお世話になりました．深く感謝しています．

事務室の木村さんには，毎回 ViBi チームが日立製作所とミーティングする時の会議室を予約していただき，ありがとうございます．

また，チームの構成員である大木遼太，柳沼工也，孫嘉成からも多くの助力と意見をいただきました．ありがとうございます．

最後には，自分が日本に来てから，ずっと支えてくれた両親，すべての友人に心より感謝いたします．

参考文献

[1] 先導的 IT スペシャリスト育成推進プログラム拠点間教材等洗練事業 PBL 教材洗練 WG，PBL(Project Based Learning)型授業実施におけるノウハウ集，2011.

[2] Jonathan Rasmusson ，アジャイルサムライ−達人開発者への道，オーム社出版，2014. 小川明彦，坂井誠，“チケット駆動開発”翔泳社，2012.

[3] 国立大学法人和歌山大学，2013 年度ソフトウェア高額分野の先導的研究支援事業「IPA EPM-X」の機能拡張によるプロアクティブ型プロジェクトモニタリング環境の構築 –次世代の定量的プロジェクト管理ツールとリポジトリマイニング研究基盤- 」成果報告書，2015.

[4] 阿萬裕久，野中誠，水野修，“ソフトウェアメトリクスとデータ分析の基礎，”コンピュータソフトウェア，pp.12-28，Vol.28，No.3，2011.

[5] 福安直樹，リポジトリのリアルタイムな可視化にもとつく PBL の支援環境，電子情報通信学会技術研究報告．SS，ソフトウェアサイエンス pp.110(458)，pp.121-126，2010.

[6] 大蔵君治，IZMI 開発者と成果物の編集頻度に着目したソフトウェア開発リポジトリの可視化ツール，コンピュータソフトウェア，28(3)，pp.147-152，2011.

[7] 坂元康好, Metrics Viewer: サービス指向リポジトリマイニングを活用したソフトウェアメトリクス可視化ツール，電子情報通信学会技術研究報告，MSS，システム数理と応用 112(457)，pp.127-132，2013.

[8] 五田篤志ら，開発履歴を利用した風林火山モデルに基づく開発者特性の分析，情報処理学会，研究報告ソフトウェア工学(SE)，2014-SE-185(9)，pp.1-6，2014.

[9] 井垣宏ら，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 56(2)，pp.701-713，2015.

[10] 坂元康好，柗本真佑，中村匡秀，サービス指向リポジトリマイニングを効率化するキャッシュ機構の実装 電子情報通信学会研究報告，Vol.113，No.269，SS2013-44，pp.73-78，2013.

[11] 山田悠太, 藤原賢, 吉田則裕，飯田元，トピック抽出に基づく開発者の活動に着目したリポジトリ可視化手法，研究報告ソフトウェア工学(SE)，2012-SE-178(16)，pp.1-7，2012.

付録　開発構想書

目次

1 序論

1.1 本書の概要

本書は，筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻の「高度 IT 人材育成のための実践的ソフトウェア開発専修プログラム」の一環である，「研究開発プロジェクト」における成果物の開発構想について述べたものである．成果物の開発目的や概要，対象利用や開発体制など，開発に関する情報を明確にするものである．

1.2 問題提起

ソフトウェア開発プロジェクトの多くは複数のソフトウェア開発者による共同作業である．しかしプロジェクト管理者にとって，そのプロジェクトに対するソフトウェア開発者のパフォーマンス(能力や経験)を定量的に測れないという課題がある．プロジェクト管理者の主観的かつ曖昧な評価によって，ソフトウェア開発者のモチベーションの喪失し，ひいてはプロジェクト運営の失敗につながる場合も少なからず存在する．この問題の解決のため，プロジェクト管理者が複数のソフトウェア開発者に対して定量的な根拠による客観的な評価をできるようなソリューションが望まれている．

1.3 本プロジェクトの方針

本プロジェクトは，株式会社日立製作所(以下，日立)による問題提起(第1.2節)によって発足したプロジェクトである．

株式会社日立製作所により提示された問題を解決するため，本プロジェクトではMITEMIRUの提案および開発をおこなう．MITEMIRUは，評価対象者が関連するソフトウェア開発プロジェクトに対応する構成管理ツール等のリポジトリから取得したデータを集計・分析し，加工する．加工データをグラフとして可視化することにより，ソフトウェア開発者のパフォーマンスに関連する定量的な評価材料として提供し，日立に提示された問題を解決する．

MITEMIRUの実装にあたり，まずはリポジトリマイニングやプロジェクト管理等に関連する文献調査により，リポジトリ上に存在するデータと対応するソフトウェア開発者のパフォーマンスの関係を明確にする．次に，この知見に基づき，パフォーマンス評価の判断材料となるメトリクスとその要素なるメトリクスデータ，およびメトリクスデータの表現に必要となるリポジトリ上のデータを検討する．そして，提供するメトリクスデータと具体的な可視化方法を決定する．

なお本プロジェクトでは，多数のソフトウェア開発者が共同作業するソフトウェア開発プロジェクトを対象に，客観的な各ソフトウェア開発者の評価支援を可能とする環境構築を目標とする．

2 システム利用者

2.1 対象となる利用者

MITEMIRU は, 10 名程度のソフトウェア開発プロジェクトの管理者を利用対象者とする. 例として, 企業における小規模なソフトウェア開発プロジェクト(大型プロジェクトを分割した, 小型プロジェクトを含む)の管理者や, PBL の学生評価をおこなう指導教員等が挙げられる.

2.2 利用者が抱える問題

利用者が抱える問題として, ソフトウェア開発者の不適切な評価が挙げられる. ソフトウェア開発プロジェクトは複数の開発者による共同作業であり, プロジェクトの推進には各開発者の役割分担や能力が大きく影響を与える. そのため, プロジェクト開始以前にソフトウェア開発者を適切に選定しなければならない.

他方, 企業における人事評価や PBL における成績判定等, プロジェクト開始後には定期的なソフトウェア開発者の評価を実行しなければならない.

いずれのケースにおいても厳密かつ適切な評価は必要不可欠であるが, 定量的な根拠に基づいた開発者の経験および能力の把握はプロジェクト管理者にとって困難であり, これは問題の一因として考えられる.

MITEMIRU はこの原因を解消するため, ソフトウェア開発者評価のための定量的な判断材料を提供する.

3 システムの概要

3.1 システムの役割

MITEMIRU は, ソフトウェア開発者評価の定量的な根拠に基づくパフォーマンスの客観評価を支援するシステムである. MITEMIRU は, バージョン管理ツールやチケット管理ツール等のリポジトリに蓄積されたデータを収集し, 集計や分析をおこなう. その後, この集計・分析結果をソフトウェア開発者の評価に関連する定量的な判断材料(メトリクスデータ)として, 利用者であるプロジェクト管理者に提供する. また MITEMIRU は直截的な解釈を実現するため, メトリクスデータをグラフとして可視化し, 利用者に提供する. これにより, 2.2 節にて述べた問題の一因解決が期待できる.

3.2 システムが備える機能

3.1 節にて述べた役割を実現するために, MITEMIRU は, 大きく 4 つの機能を備える.

1 つ目は, 「リポジトリ連携機能」である. 3.1 節で述べた通り, MITEMIRU はバージョン管理ツールやチケット管理ツール等のリポジトリにアクセスし, 蓄積されたデータを取得する. リポジトリへのアクセスには認証が必要であり, データ取得時に認証情報

を利用しなければならない．MITEMIRU ではリポジトリへの円滑なアクセスを実現するため，認証情報を予め登録するための「リポジトリ連携機能」を実装することとする．この機能は，利用者が保有する複数のリポジトリに対する認証情報を紐付け，MITEMIRU のデータベースに保存するものである．

2 つ目は，「メトリクスデータ閲覧機能」である．これは，評価対象となるソフトウェア開発者の定量的なパフォーマンス評価の判断材料を提供するための，主要な機能である．MITEMIRU はリポジトリ連携機能で予め登録された認証情報を利用して，リポジトリへアクセスし，蓄積されたデータを収集する．その後，集計・分析を実行した後グラフとして可視化し，利用者に提供する．利用者は，このグラフから得られた知見をパフォーマンス評価の判断材料として利用する．グラフはプロジェクト毎に確認でき，そのプロジェクトに所属する全ての開発者に関連するメトリクスデータが表示される．一方，ソフトウェア開発者を指定し，関連する全てのプロジェクトでのパフォーマンスを時系列に並べて確認することも可能とする．

3 つ目は，「各種管理機能」である．主にデータの管理を目的としており，リポジトリ連携機能で登録されたプロジェクトやリポジトリの認証情報の更新や削除を実行する機能である．

これらの主要機能に加えて，システムユーザのログイン・ログアウト機能や，メトリクスデータのエクスポート機能等，様々な機能を検討している．詳細は第 4.2 節に述べる．

3.3 システムの形態

図 1 にシステムの概要図を示す．MITEMIRU は，リポジトリへのアクセスが可能である環境にインストールされなければならない．特に企業では，外部ネットワークから隔離された環境にリポジトリが設置されることが大いに想定されるため，MITEMIRU はその環境に応じてインストールされる必要がある．そこで，MITEMIRU はオンプレミス型の運用形態を想定する．また MITEMIRU は汎用性を高めるため，ブラウザ経由で利用されるものとする．なお複数人による利用も想定されることから，ログインによる利用者の識別を必須条件とする．

4 システムの要件

4.1 機能要件

本プロジェクトは時間が制約されているため，限定されたスコープで開発を実施する．そこで，顧客とのミーティング等で機能要件の洗い出しをおこなった(表 1)．またそれら対して優先度を設定し，優先度の高いものから実装することとする．優先度の定義は，以下の通りである．

- 高: 利用者の最低限の要件を満たすために，システムの動作上，必須となる機能
- 中: 利用者の要件を満たすため必要ではあるが，不足してもシステムを動作させられる機能
- 低: 利用者の要件として求められてはいるが，必要ではない機能

表 1: 機能要件と優先度

主機能	機能要件	概要	優先度
ログイン・ログアウト機能	システムユーザの新規登録	システムの利用者を一意に識別するためのシステムユーザ情報を，データベースに保存する.	高
	ログイン	保存されたシステムユーザ情報を利用し，システムにログインさせる.	高
	ログアウト	システムからログアウトさせる.	高
	システムユーザ情報管理	データベースに保存されたシステムユーザ情報に対し，更新および削除をおこなう(ログイン時のシステムユーザ情報に限る).	中
リポジトリ連携機能	リポジトリ認証情報の保存	リポジトリの問い合わせに必要となる認証情報をデータベースに保存する.	高
	開発者情報の登録	リポジトリに関連する全ての開発者情報をデータベースに保存する.	中
メトリクスデータ閲覧機能	プロジェクトの選択	メトリクスデータを閲覧したいプロジェクトを選択できる.	高
	メトリクスデータの生成と可視化(同一プロジェクト)	メトリクスデータを閲覧したいプロジェクトを選択し，リポジトリからデータを取得した後，メトリクスデータとして集計･分析結果を生成する. それをグラフとして可視化し，利用者に提供する．なお，選択されたプロジェクトに所属する全ての開発者に関連するメトリクスデータを算出する.	高
	メトリクスデータの生成と可視化(複数プロジェクト)	メトリクスデータを閲覧したい開発者を複数選択し，リポジトリからデータを取得した後，メトリクスデータとして集計・分析結果を生成する. それをグラフとして可視化し，各開発者のパフォーマンスを比較する形式で利用者に提供する．なお，選択された開発者が所属する各プロジェクトのリポジトリからデータし，それぞれの開発者に応じてメトリクスデータを算出する.	
	メトリクスデータの生成と可視化(単一開発者)	メトリクスデータを閲覧したい開発者を選択し，リポジトリからデータを取得した後，メトリクスデータとして集計・分析結果を生成する. それをグラフとして可視化し，利用者に提供する. なお，選択された開発者に関わる全てのメトリクスデータを算出し，時系列に応じてグラフとして表示する	中

	レポートファイルの出力	メトリクスデータの結果とグラフを外部ファイルとして出力する.	低
各種管理機能	プロジェクト情報管理	データベースに保存されたリポジトリの認証情報に対し，更新および削除をおこなう.	高
	開発者情報管理	データベースに保存された開発者情報に対し，更新および削除をおこなう.	中
	システムユーザ情報管理	データベースに保存されたシステムユーザ情報に対し，更新および削除をおこなう(スーパユーザのみ利用可能，全てのシステムユーザ情報が対象).	中

4.2 非機能要件

4.1 節と同様に, 表 2 に非機能要件を示す. なお, 表 2 に記載がない項目については, 考慮しないものとする.

表 2: 非機能要件と優先度

項目	特徴	概要	優先度
性能・拡張性	リソース拡張性	システムで利用するリソース(リポジトリデータ)の取得先が簡便に追加可能であること.	高
移行性	移行方式	アプリケーション形式でのインストールが可能であること.	高
	移行対象(機器)	汎用型コンピュータへ簡便に移行可能であること.	高
	移行対象(データ)	汎用型コンピュータで利用可能な範囲のデータベースに移行可能であること.	高

5 開発体制

5.1 ステークホルダー

本プロジェクトの主要なステークホルダーを表 3 に示す. 本プロジェクトの顧客は株式会社日立製作所(日立)であり，三末和男教授の指導の元で筑波大学の学生がシステムの開発をおこなう. 加えて，日立はソフトウェア開発分野，三末和男教授は情報可視化分野のアドバイザーとして, 開発に参加する.

表 3: ステークホルダー

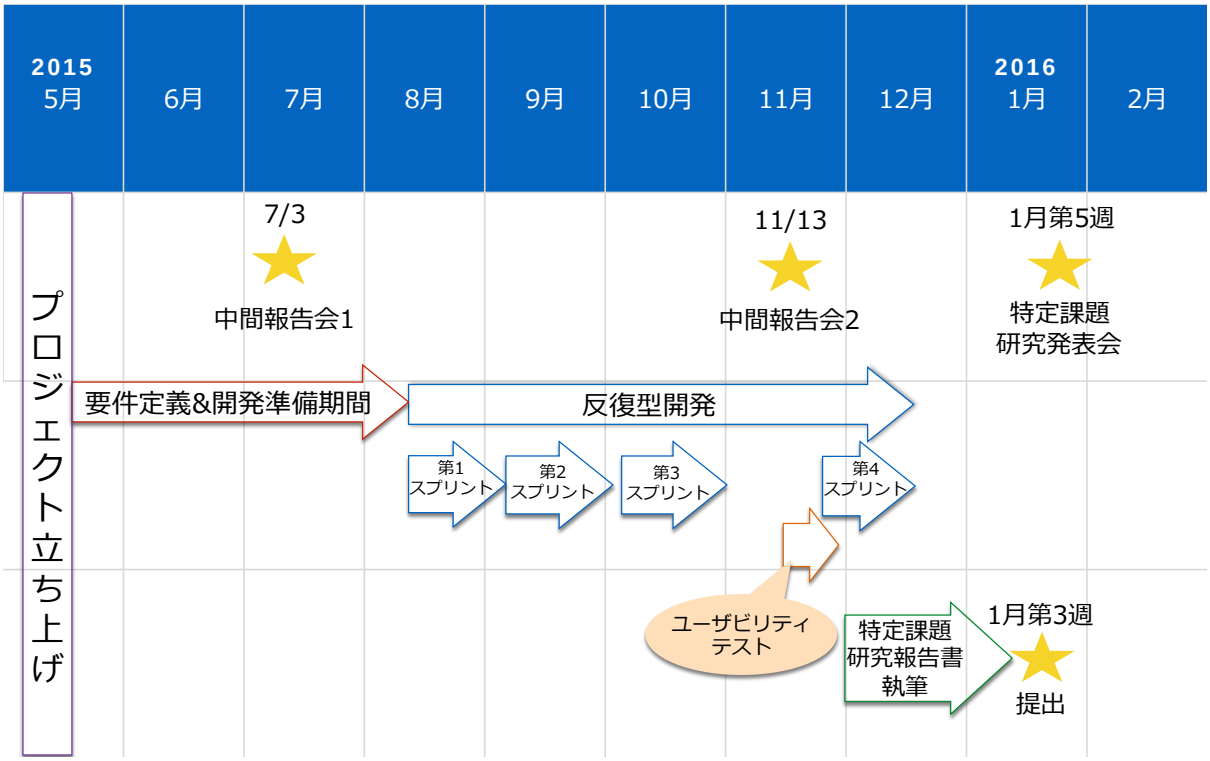
所属	役割	名前
筑波大学	課題担当教員, アドバイザー	三末 和男
	開発チーム	大木 遼太

		柳沼 工也
		孫 嘉成
		鄒 一民
株式会社 日立製作所	顧客, アドバイザー	居駒 幹夫
		土田 正士
		白井 明
		中村 宇佑
		河野 哲也
		須貝 佳彦

5.2 スケジュール

本プロジェクトは，図 2 に示すスケジュールを想定している．本プロジェクトでは期間の延長は一切認めらないため，大きなスケジュールの変更は想定していない．なお，スケジュールに加えて開発チームメンバーの入れ替えや増員も不可能であるため，適切なプロジェクト遂行のためにはスコープを調整が必要不可欠となる．そこで本プロジェクトでは反復型開発手法を採用し，顧客との密なコミュニケーションによる優先事項の検討と，それに基づいた機能の実装予定している．

図 2: 本プロジェクトのスケジュール



5.3 開発における利用ツール

5.3.1 GitHub

本プロジェクトの成果物のうち, MITEMIRU のソースコードや関連するドキュメントを GitHub リポジトリ上に保存する. なお, これらのデータはすべてオープンリポジトリ上に保存し, 公開されるものとする.

5.3.2 Redmine

プロジェクトの進捗管理には,オープンソースのプロジェクト管理ソフトウェアである Redmine を利用する. またプロジェクトに関連する情報は逐次 Wiki に記載され, 更新されるものとする.

5.3.3 Dropbox

オンラインストレージサービスである Dropbox を利用し, チームでの活動において生成されるデータやドキュメント等を管理する. 必要に応じて学生チームと指導教員, アドバイザーとのデータ共有も同様におこなうものとする.

5.3.4 Slack

チーム内コミュニケーションツールとして, Slack を利用する. なお, 学生チームと指導教員, およびアドバイザーとのコミュニケーションについては, 簡便なものに限り Slack 上でおこなうものとする.

5.4 開発におけるリスク

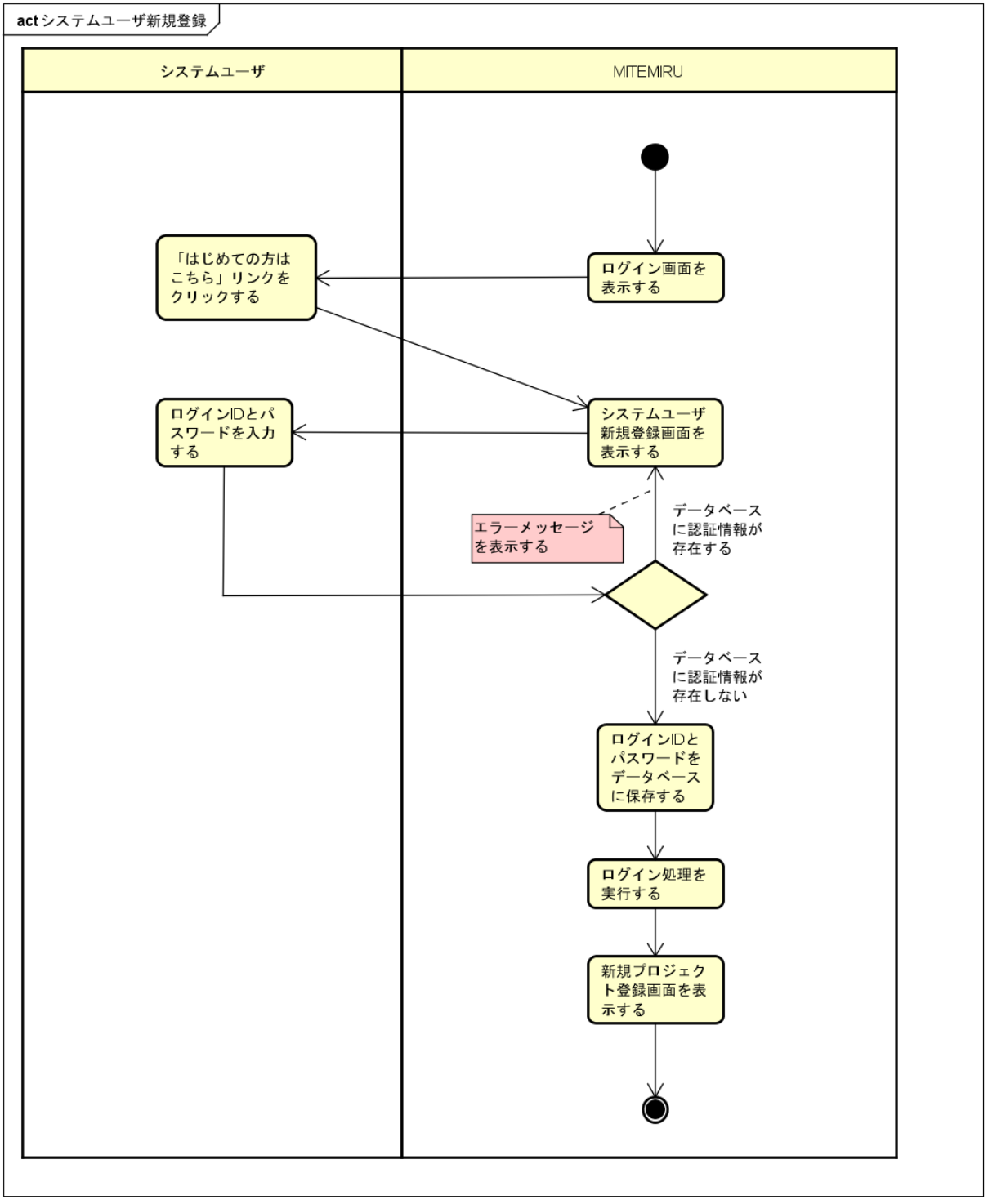
5.4.1 利用者の制約

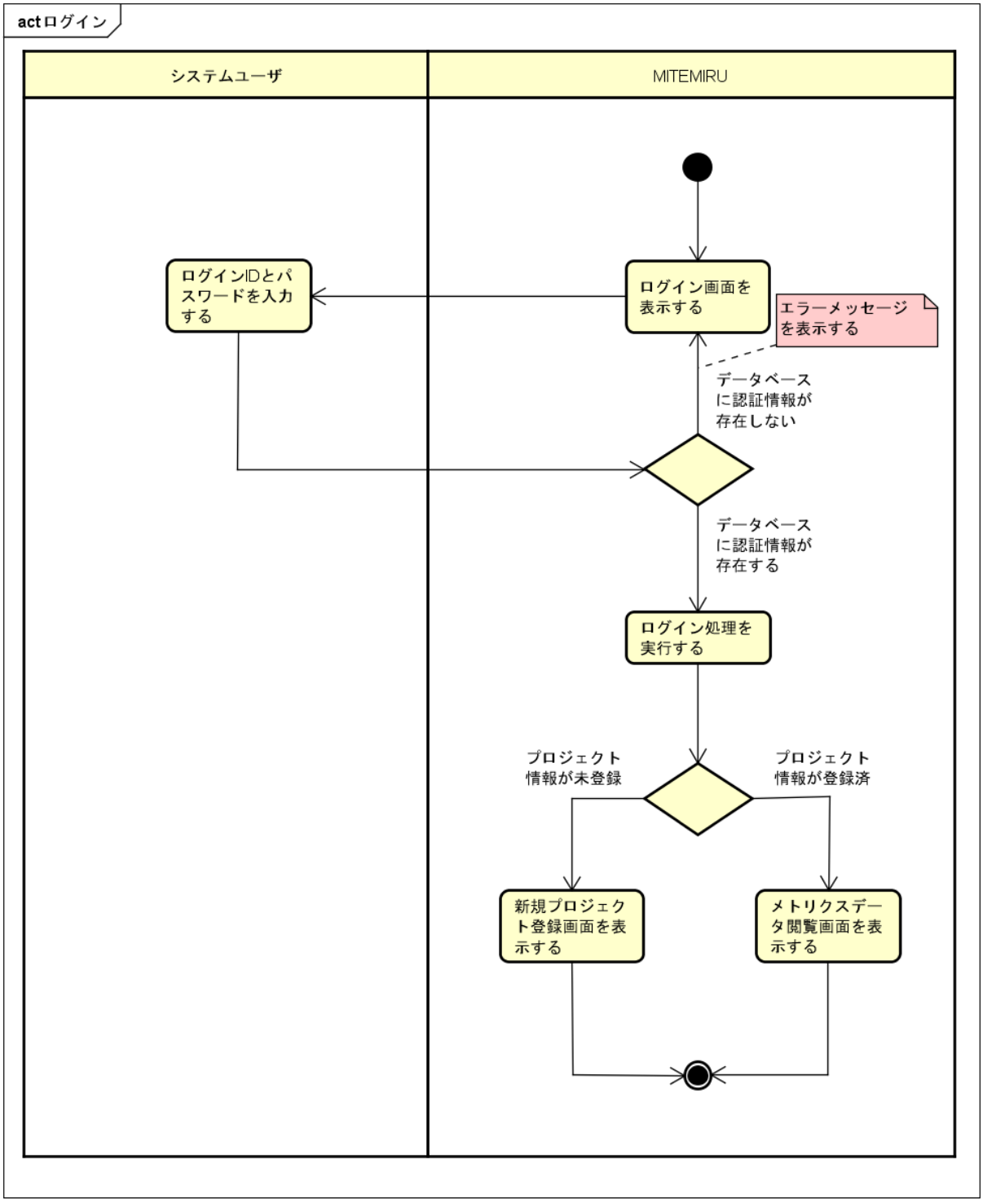
MITEMIRU ではメトリクスデータ閲覧機能の実装時に, 利用対象となるリポジトリを選定する必要がある. そのため, 利用者によるレビューを実施するためには対象となる利用者が数多く使用しているリポジトリを選定しなければならず, 場合によってはレビューを実行できない可能性も考えられる.

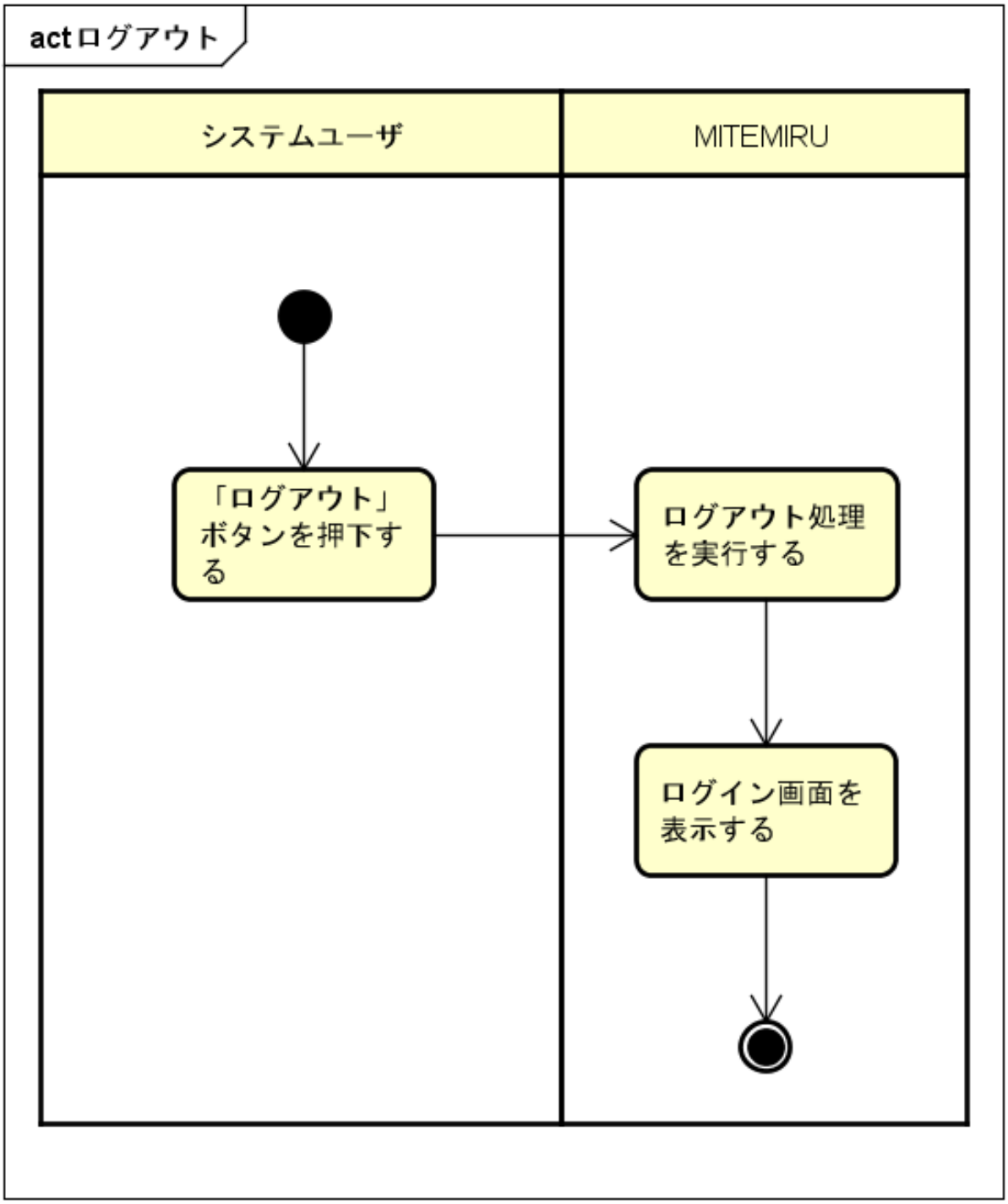
5.4.2 就職活動

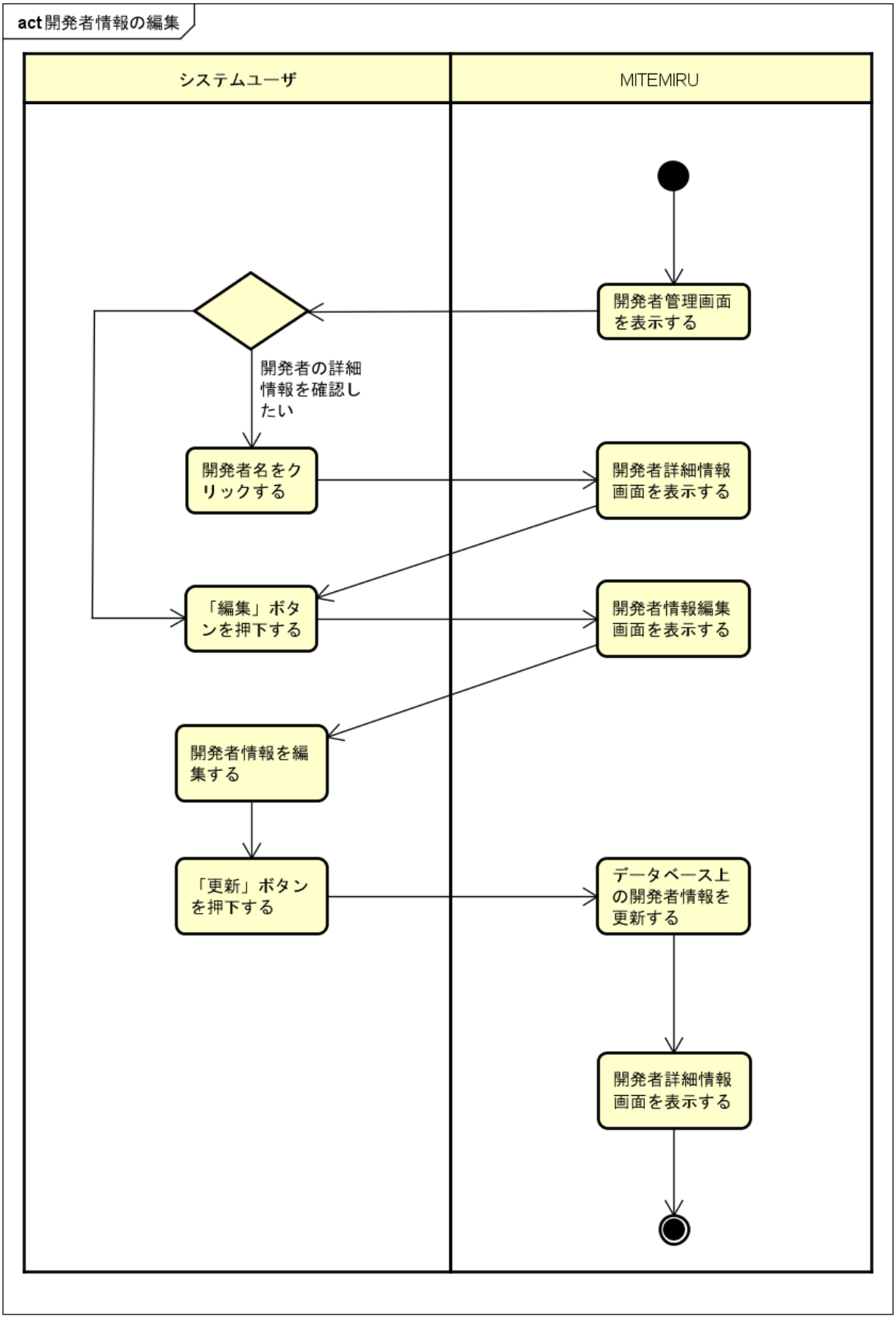
2016 年度卒業者向けの就職活動は, 経団連の採用選考に関する指針変更により, 広報活動を２０１５年の３月１日以降, 選考活動を２０１５年8月1日以降におこなうとの指針を提示している. しかし現状は, 「倫理憲章」賛同企業もインターンシップやジョブマッチングといった形で様々な広報活動を実施している. その結果, チームメンバーの就職活動は常に活発な状態であり, プロジェクトの参加時間が限られてしまうといった現状がある.

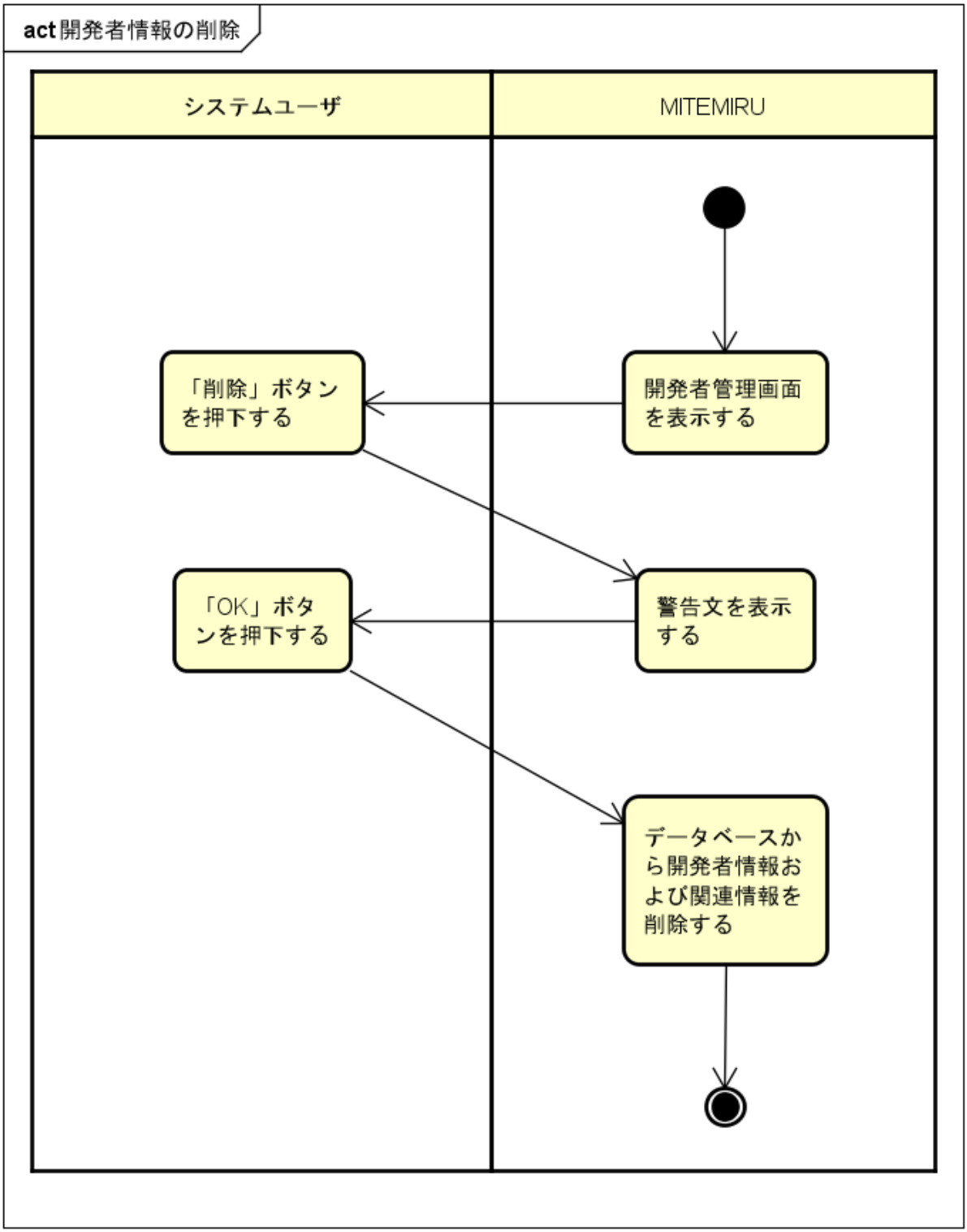
付録　アクティビティ図

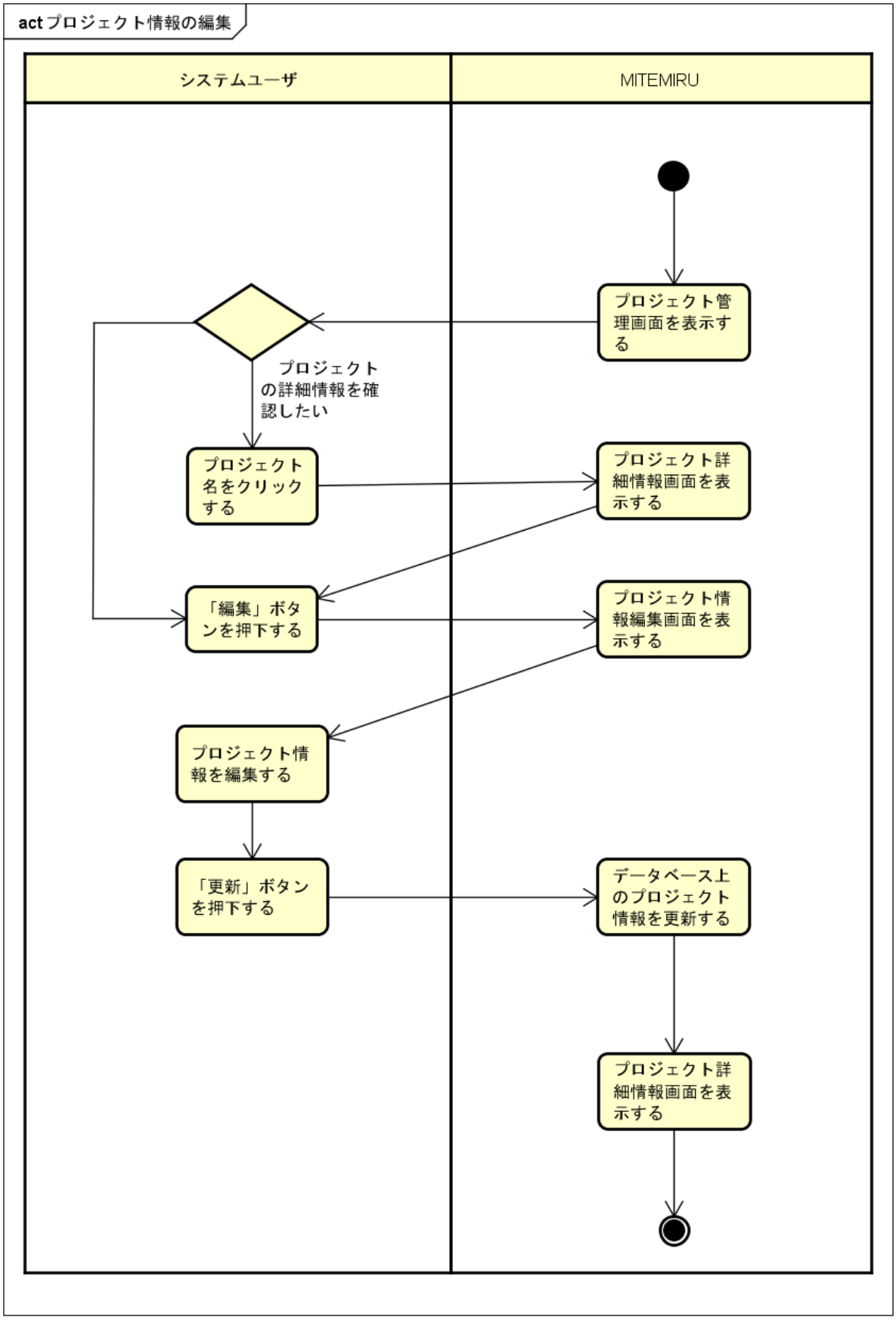

actシステムユーザ新規登録
システムユーザ
MITEMIRU
「はじめての方はこちら」リンクをクリックする
ログイン画面を表示する
ログインIDとパスワードを入力する
システムユーザ新規登録画面を表示する
エラーメッセージを表示する
データベースに認証情報が存在する
データベースに認証情報が存在しない
ログインIDとパスワードをデータベースに保存する
ログイン処理を実行する
新規プロジェクト登録画面を表示する

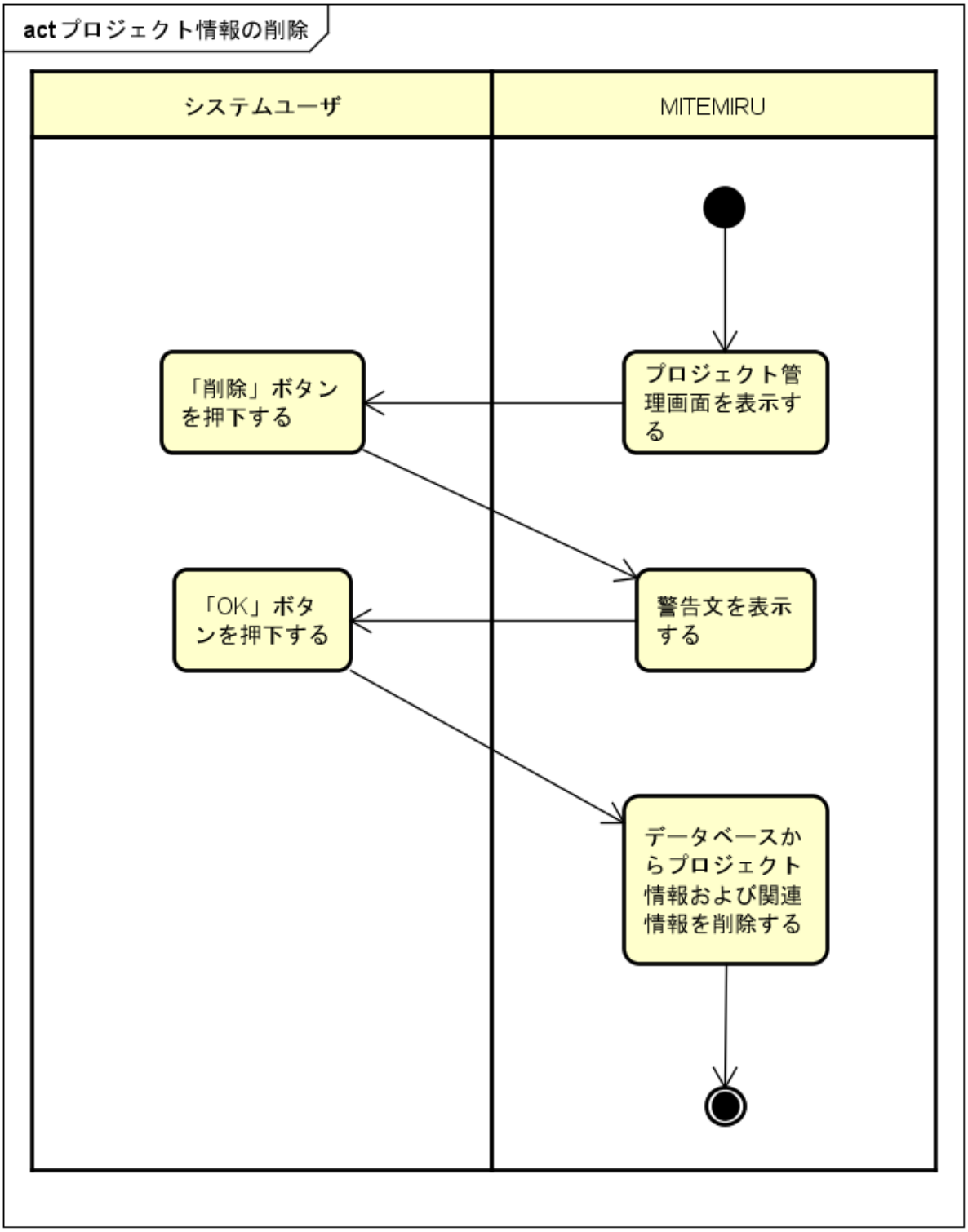

actログイン
システムユーザ
MITEMIRU
ログインIDとパスワードを入力する
ログイン画面を表示する
エラーメッセージを表示する
データベースに認証情報が存在しない
データベースに認証情報が存在する
ログイン処理を実行する
プロジェクト情報が未登録
プロジェクト情報が登録済
新規プロジェクト登録画面を表示する
メトリクスデータ閲覧画面を表示する

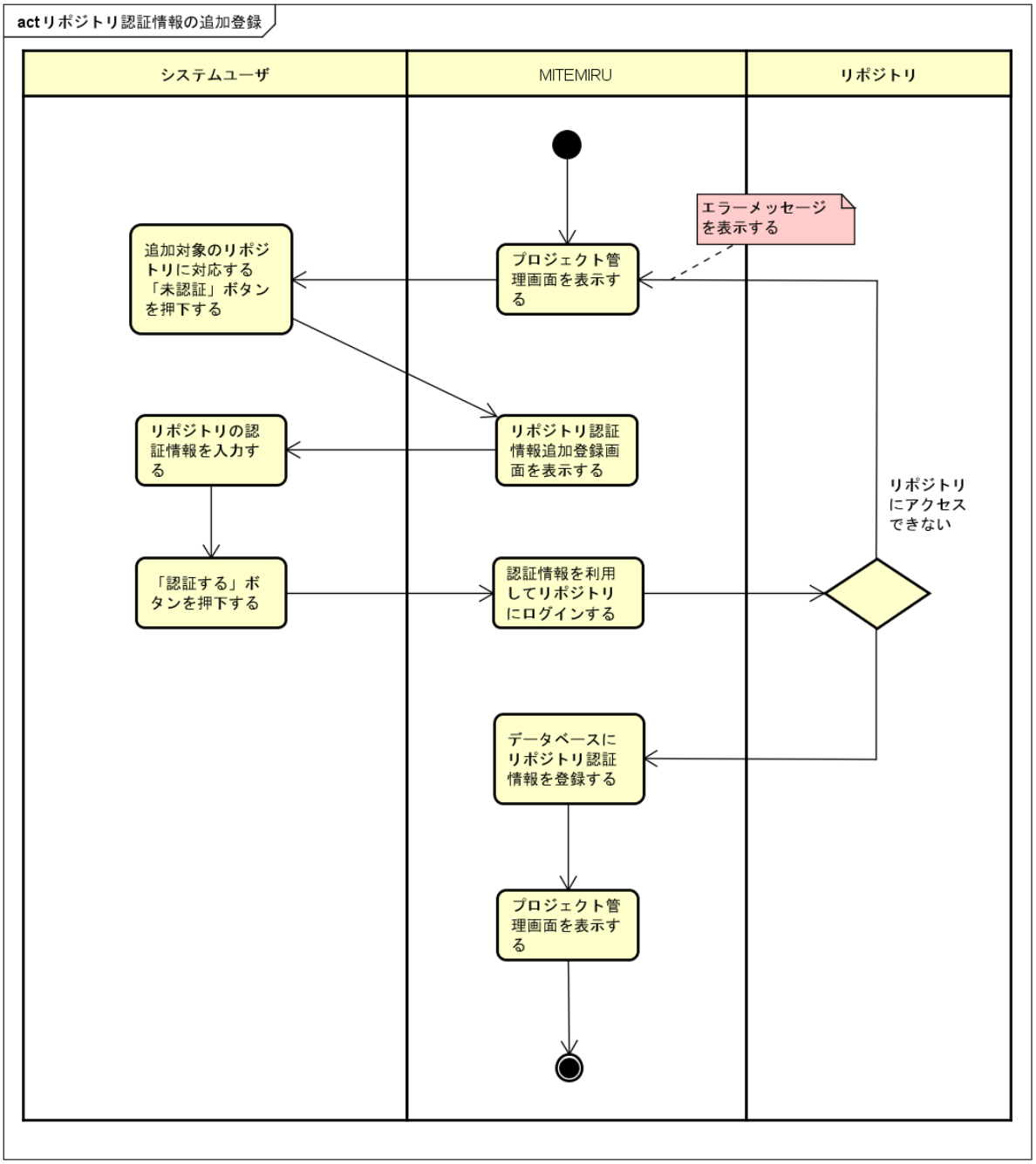

actログアウト
システムユーザ
MITEMIRU
「ログアウト」ボタンを押下する
ログアウト処理を実行する
ログイン画面を表示する

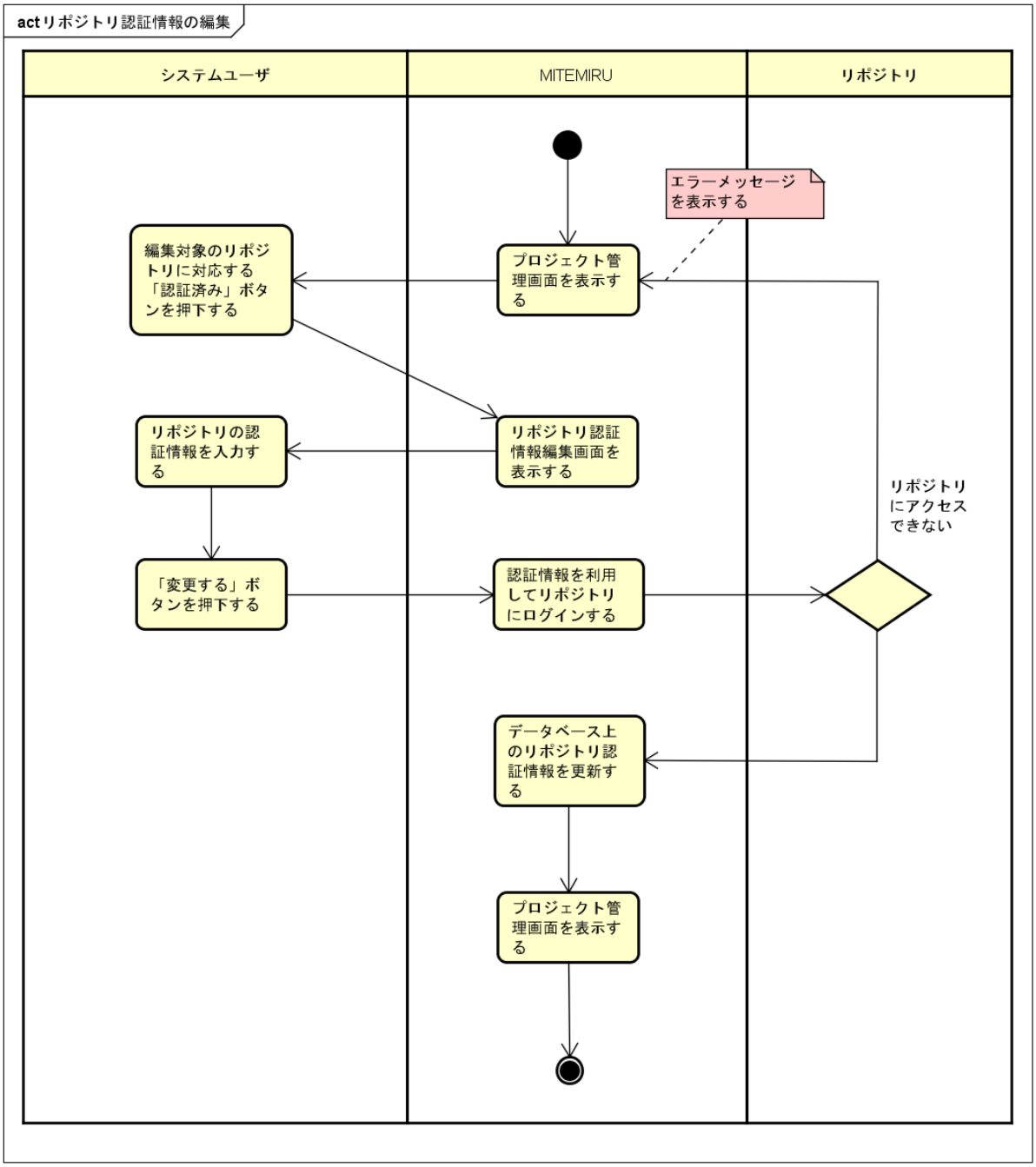

act 開発者情報の編集
システムユーザ
MITEMIRU
開発者管理画面を表示する
開発者の詳細情報を確認したい
開発者名をクリックする
開発者詳細情報画面を表示する
「編集」ボタンを押下する
開発者情報編集画面を表示する
開発者情報を編集する
「更新」ボタンを押下する
データベース上の開発者情報を更新する
開発者詳細情報画面を表示する

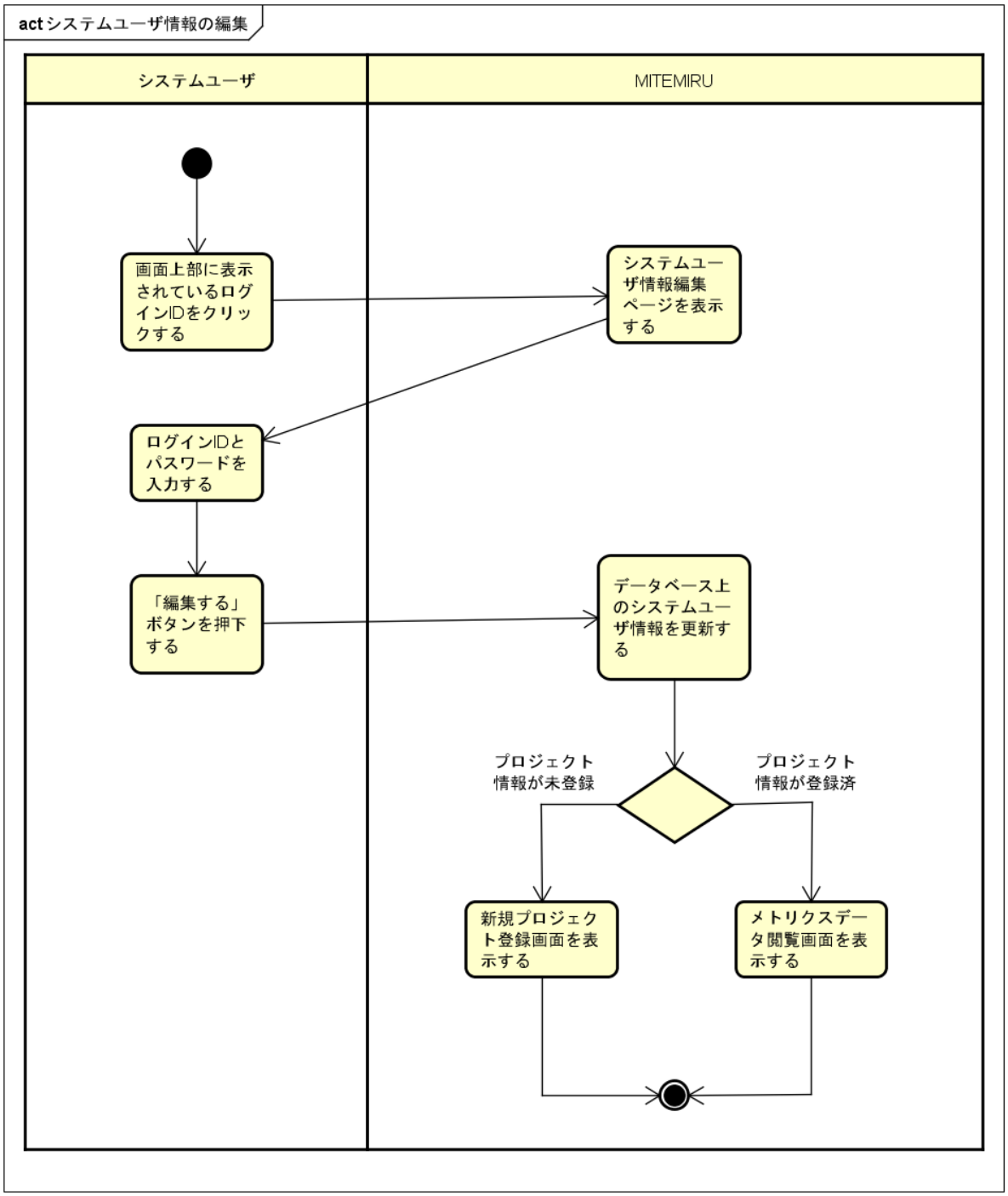

act開発者情報の削除
システムユーザ
MITEMIRU
「削除」ボタンを押下する
開発者管理画面を表示する
「OK」ボタンを押下する
警告文を表示する
データベースから開発者情報および関連情報を削除する

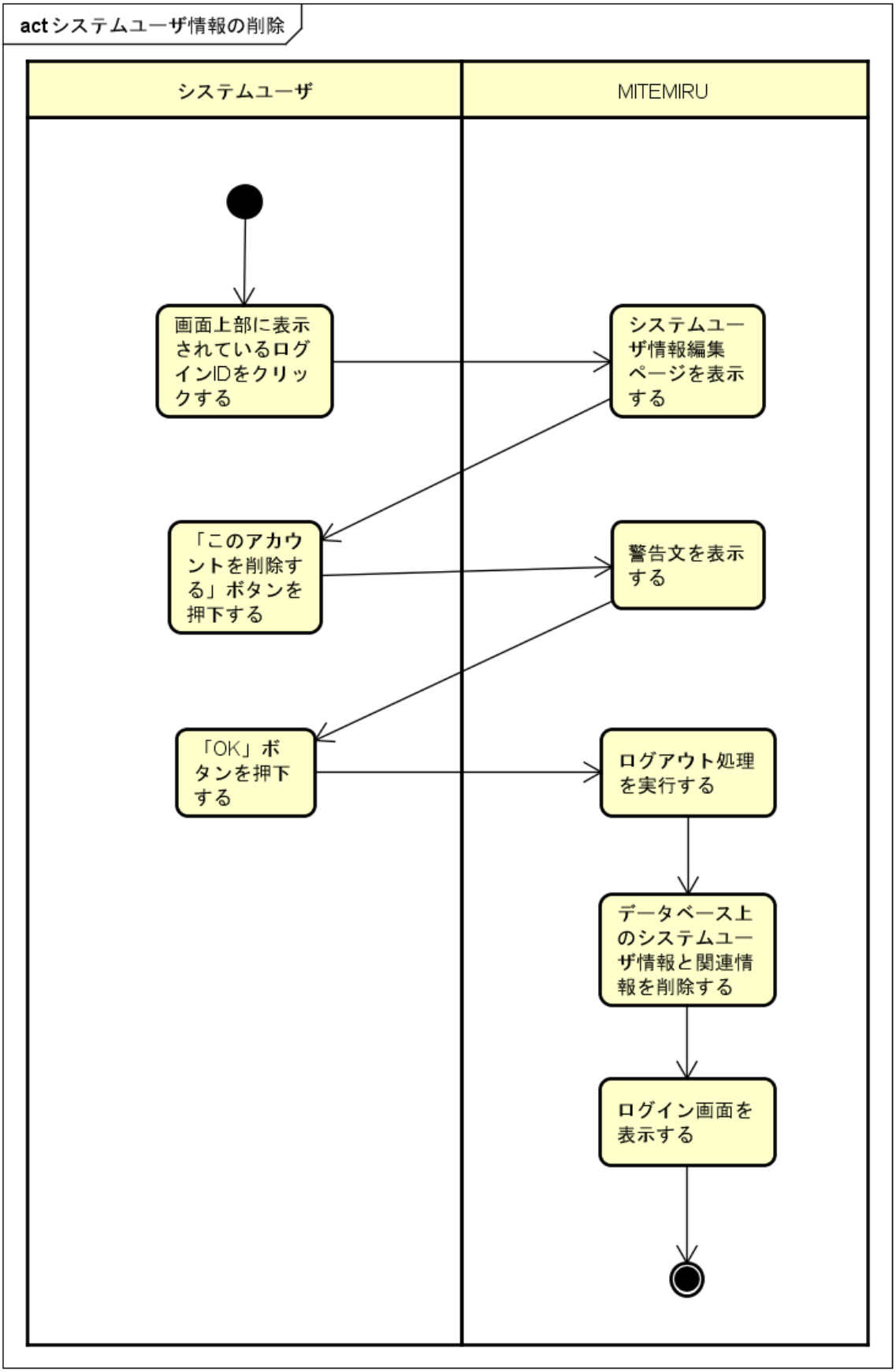

act プロジェクト情報の編集
システムユーザ
MITEMIRU
プロジェクト管理画面を表示する
プロジェクトの詳細情報を確認したい
プロジェクト名をクリックする
プロジェクト詳細情報画面を表示する
「編集」ボタンを押下する
プロジェクト情報編集画面を表示する
プロジェクト情報を編集する
「更新」ボタンを押下する
データベース上のプロジェクト情報を更新する
プロジェクト詳細情報画面を表示する

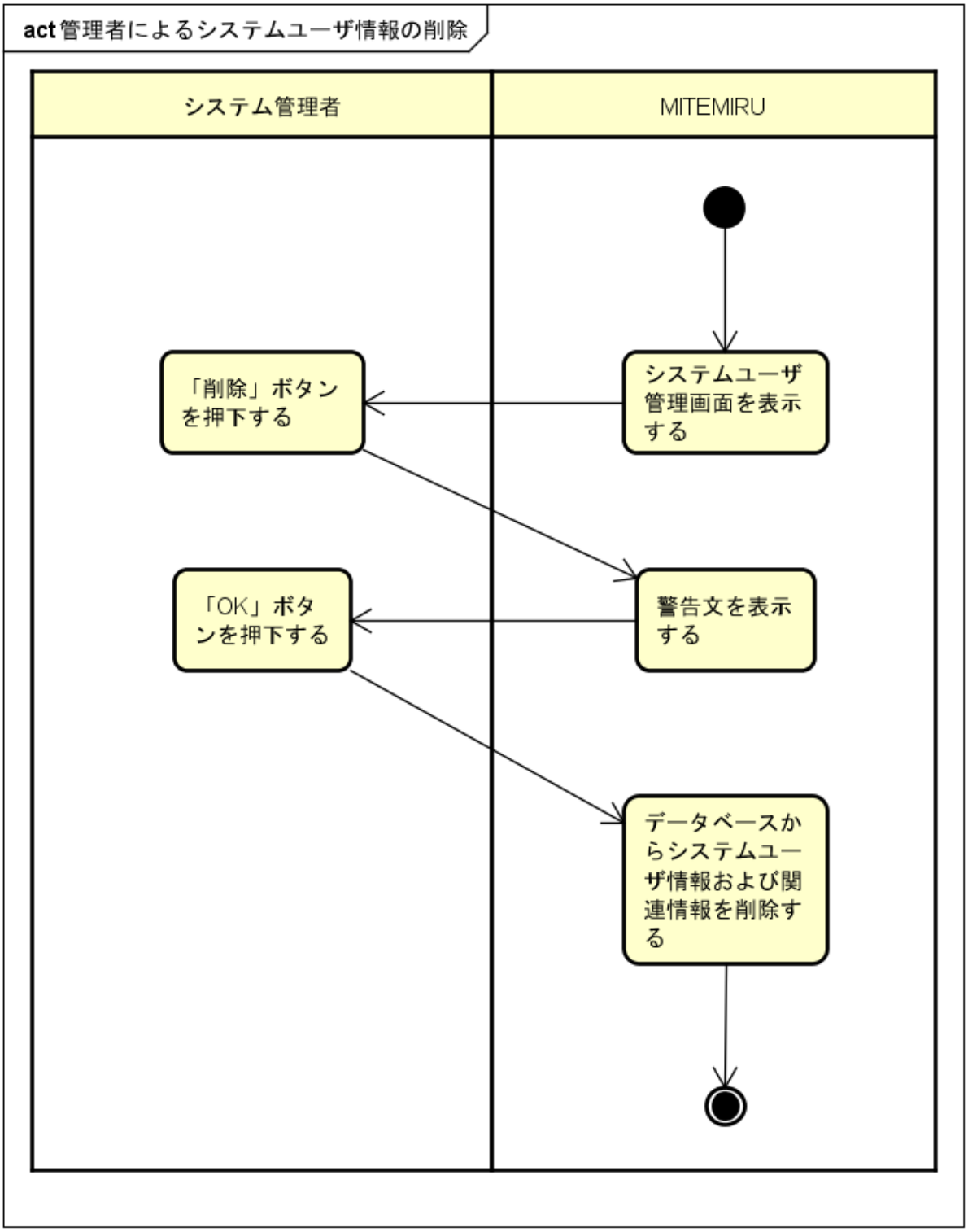

act プロジェクト情報の削除
システムユーザ
MITEMIRU
「削除」ボタンを押下する
プロジェクト管理画面を表示する
「OK」ボタンを押下する
警告文を表示する
データベースからプロジェクト情報および関連情報を削除する


actリポジトリ認証情報の追加登録
システムユーザ
MITEMIRU
リポジトリ
エラーメッセージを表示する
追加対象のリポジトリに対応する「未認証」ボタンを押下する
プロジェクト管理画面を表示する
リポジトリの認証情報を入力する
リポジトリ認証情報追加登録画面を表示する
リポジトリにアクセスできない
「認証する」ボタンを押下する
認証情報を利用してリポジトリにログインする
データベースにリポジトリ認証情報を登録する
プロジェクト管理画面を表示する


actリポジトリ認証情報の編集
システムユーザ
MITEMIRU
リポジトリ
エラーメッセージを表示する
編集対象のリポジトリに対応する「認証済み」ボタンを押下する
プロジェクト管理画面を表示する
リポジトリの認証情報を入力する
リポジトリ認証情報編集画面を表示する
リポジトリにアクセスできない
「変更する」ボタンを押下する
認証情報を利用してリポジトリにログインする
データベース上のリポジトリ認証情報を更新する
プロジェクト管理画面を表示する


actシステムユーザ情報の編集
システムユーザ
MITEMIRU
画面上部に表示されているログインIDをクリックする
システムユーザ情報編集ページを表示する
ログインIDとパスワードを入力する
「編集する」ボタンを押下する
データベース上のシステムユーザ情報を更新する
プロジェクト情報が未登録
プロジェクト情報が登録済
新規プロジェクト登録画面を表示する
メトリクスデータ閲覧画面を表示する


actシステムユーザ情報の削除
システムユーザ
MITEMIRU
画面上部に表示されているログインIDをクリックする
システムユーザ情報編集ページを表示する
「このアカウントを削除する」ボタンを押下する
警告文を表示する
「OK」ボタンを押下する
ログアウト処理を実行する
データベース上のシステムユーザ情報と関連情報を削除する
ログイン画面を表示する


act管理者によるシステムユーザ情報の削除
システム管理者
MITEMIRU
システムユーザ管理画面を表示する
「削除」ボタンを押下する
警告文を表示する
「OK」ボタンを押下する
データベースからシステムユーザ情報および関連情報を削除する