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

大規模ソフトウェアの品質向上に関わる研究開発
―ユーザ権限管理機能およびデータ読み込み蓄積機能の設計と開発―

齋藤創太

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

２０１４年　３月

概要

近年システムの大規模化・複雑化に伴っていっそう品質の向上が難しくなってきている．品質向上のための一つの方法としてソフトウェアに含まれる不具合を修正することが挙げられる．その場合不具合は早く発見すればするほど修正するコストを下げられることから，なるべく速い段階で不具合を発見，修正することが求められている．一方でソフトウェアの開発過程で企業には，ソフトウェア構成管理システムのリポジトリをはじめ様々なデータが日々蓄積されている．そこで私たちは株式会社日立製作所（以下，日立）と連携し，ソフトウェアの開発過程で蓄積されていくソフトウェア構成管理システムのデータなどを一定時間ごとに分析し，品質低下に関する警告を行うと共に，品質に関連する指標を提供するシステムの研究開発を行う．今回開発するシステムは実際の企業内で使用できることを目指しており，日立にご協力頂き試運用を行うことも視野に入れている．著者は本システムを企業内で運用するにあたって必要となるユーザ権限管理機能や，開発過程で蓄積されたデータの読み込みと蓄積を行う機能の設計と開発を行った．また他には本システムが品質向上の一助となるための，ソフトウェアのメトリクス表示画面の一部開発を行った．そしてシステム設計およびシステム実装の主責任者として，チームメンバーが行った開発成果の取りまとめと開発基盤の構築の支援を行った．

目次

図目次

表目次

第1章 はじめに

1.1 背景

ソフトウェアにおける品質の向上は必要とされているが，近年ソフトウェアの大規模化に伴い品質向上が難しくなってきている．例えばソフトウェアの品質を損なうことは金融システムの障害に挙げられるように企業に信頼を失わせ致命的な損害を与えてしまい，またそれによって国に大きく経済的な損失を与えることもある[1]．しかし今日，ソフトウェアは大規模化・複雑化しており[2]，品質向上が難しくなって[5]きている．

そのための品質向上のための一つの方法としてソフトウェアに含まれる不具合を修正することが挙げられ，その場合早い段階で不具合を修正することが求められている．なぜなら，不具合は早く発見すればするほど修正するコストを下げられる[5]からである．しかし現状，不具合は開発の終盤であるテスト工程以後で発見されることが多い．

一方でソフトウェアの開発過程において企業には様々なデータが日々蓄積されている．それはソフトウェア構成管理システムのリポジトリをはじめ，チケット管理システムで管理されたバグ情報，またそれらの情報を組み合わせどのファイルにバグが混入していたかという情報を得ることもプロジェクトによって可能である．

そこで私たちは株式会社日立製作所（以下，日立）と連携し，ソフトウェアの開発過程で蓄積されていくソフトウェア構成管理システムのデータなどを一定時間ごとに分析し，品質低下に関する警告を行うと共に,品質に関連する指標を提供するシステムの研究開発を行う．

1.2 解決すべき問題

私たちは想定利用者をプロジェクトにおける開発者と管理者とし，彼らの要求である品質の統一や向上を中心とした要求を解決すべき問題として据える．本研究開発では作成するシステムが実際の企業内で使用されることを前提として開発を行うため，企業内で進められるいるプロジェクトの開発者と管理者を想定利用者とし，そのために日立から実際の開発過程で蓄積されるデータを共有して頂いたり，企業内での開発に関するヒアリングを行い，解決すべき問題を決定した．

1.2.1 想定利用者

私たちのシステムではソフトウェア開発のプロジェクトに携わるプロジェクトの管理者と開発者を想定利用者としている．プロジェクトの管理者とはプロジェクトの進捗管理や品質管理などプロジェクト全体を管理する人間である．プロジェクトの開発者とはプロジェクト管理者の指揮のもと，実際にプログラムの開発を行う人間である．私たちが分析したこれらの想定利用者の目的と要求を以下表 1.1 にまとめる．

表 1.1 想定利用者

	管理者	開発者
目的	プロジェクト全体の品質向上	開発担当者の能力向上
要求	● 品質の統一 ● 進捗の管理 ● 開発効率の把握	● 品質の向上 ● 効率の向上 ● 作業負担の軽減

1.2.2 システムの課題

本プロジェクトにおけるシステムの課題は２つ存在する．１つ目はソフトウェアの開発過程で蓄積されていくソフトウェア構成管理システムのデータなどを一定時間ごとに分析し，品質低下に関する警告を行うことである．これをテスト工程より前の開発工程で日々行うことによって，より早い段階で不具合を修正することを促す．２つ目はその際の一助となるよう品質管理に関するプロジェクトの情報を提供することである．そこではプロジェクト全体や各ファイルのメトリクスなどを提供することで,開発者が品質を管理しやすいようにする．

1.3 著者が取り組む課題

著者は前項の課題に対するシステムを研究開発するにあたり，そもそもこのシステムを企業内で提供する際に必要となる機能や，システムの課題を解決するにあたり直接必要となる機能，また開発の支援や開発成果の取りまとめといった研究開発の課題に取り組んだ．企業内のシステムとして提供する際には，それぞれの人間がどのプロジェクトに属しているか判断し，その人間が属するプロジェクトの警告や情報の提供を行う必要が出てくる．そのために著者はユーザ権限管理機能の設計と開発という課題に取り組んだ．またシステムの課題を解決するには，開発過程で蓄積されていくデータを取り扱う必要が出てくる．そこで著者はそれらのデータの読み込みと蓄積を行う機能の設計と開発という課題に取り組んだ．そして著者はその開発過程で必要とされる動作確認環境の構築や開発環境の構築支援，そして各自が開発した成果を取りまとめるという課題に取り組んだ．

第2章 問題と解決方法

本研究開発の課題に対する私たちが考えた問題とそれに対する解決の指針を記す．これらを日立と相談しレビューを頂きながら私たちがプロジェクトの当初に作成した，プロジェクトにおけるシステムの開発目的，使用者やステークホルダー，各種の要件，制約事項などを明確にするために記述した開発構想書に則って説明する．

私たちは想定利用者が抱える問題として，大規模ソフトウェアの開発過程で生成されるデータを活用し，ソフトウェアの品質を客観的な視点で把握し継続的に品質向上に役立てることが出来ていないという課題があると次の４点から分析した．１つ目はソフトウェアの開発過程で生成されるデータは一部活用されているが，品質の分析が週次や月次といったバッチ処理で行われるので，品質の分析結果を即時に知ることが出来ず品質向上に継続的に繋げていくことが難しいと考えられること．２つ目はそれらの分析結果は能動的に見に行かなければ知ることが出来ず，活用されにくいと考えられること．３つ目は品質に関する価値基準は定められていても，指標がどのような値になったらどうするといった，実際に行動する開発者の具体的な行動指針が共有されておらず，実際の品質向上の行動に結びつきにくいと思われること．また４つ目としてプロジェクトごとに採取しているデータの種類や，使用しているソフトウェア構成管理システムが異なることから，プロジェクトごとに比較したりまた過去の実績を役立てることが難しいと思われ，そのためバグ情報以外のデータが活用されにくいという問題があると考えらる．これらが想定利用者が抱える問題であるとする．

私たちは解決の指針としてソフトウェアの開発過程で蓄積されていくソフトウェア構成管理システムのデータなどを１時間など一定時間ごとに分析し，品質低下に関する警告を行うことを据えた．ここではまず私たちは日立から解決の指針として，開発過程で生成される膨大なデータを分析し警告をリアルタイムに出すことで品質の向上を図るという指針を頂いた．その指針に則って私たちは，例えば一時間ごとといった短い時間ごとにバグ情報外のデータも含む複数のデータから品質状況を分析し，明確な基準のもと品質に問題が見られ次第警告を開発者に行うことを解決の指針として据えることで，前項に挙げられる４つの問題を解決できると考えた．

第3章 開発システム

ここでは開発構想書に則って，開発システムを説明する．開発構想書とは私たちがプロジェクトの最初の段階で作成した本研究開発の進め方を記したものであり，これにはどのようなシステムを開発するかについて記されている．ここではそれに従って，システムの特徴，システムの構成，利用の流れ，システムの機能，システムで用いる技術という流れで開発システムについて説明する．

3.1 システムの特徴

このシステムの特徴は，開発過程で蓄積された複数のデータを使い，能動的に利用者へ品質低下の兆候を素早く警告することである．従来から存在する品質管理システムにSonarQube[10]が存在する．これはソースコードを中心に分析するソフトウェア品質管理システムであり，コーティング規約や単体テストなど主に７軸のメトリクスをグラフィカルに表示し，開発者にソース品質を意識させることが目的のソフトウェアである．これに対して本システムは，ソースコードだけではなく，ソフトウェア構成管理システムのリポジトリやバグ情報のデータなど，様々な角度からソフトウェア開発の品質を分析するほか，利用者に対してリアルタイムで気付きやアラームをあたえることが特徴である．またこのシステムの機能として，使用者の権限を設定し参照する情報のフィルタリングを可能にするほか，ソフトウェアの品質を１時間程度のラグで監視することが特徴として挙げられる．

またシステムの特徴を既存の研究を参考にすることで示す．既存の研究としてソフトウェアメトリクスの値の恒常性に基いて注力すべきモジュールを特定する手法の提案が行われている[5]が，今回はソフトウェアメトリクスの値そのものも含み，またソフトウェアメトリクスに限らず研究開発を行った．また不具合予測に関するメトリクスについての研究論文の系統的レビュー[7]に記されている予測のために使われたメトリクスの分類に照らし合わせると，本研究はソフトウェア開発プロジェクトにおける，ソフトウェアの特定の版を対象としたコードにおける静的解析ツールの出力履歴やプロセスにおけるコミットログを利用しており，本研究開発はその部分に位置すると考えられる．

3.2 システムの構成

システムはソフトウェア開発に関するデータを読み込み蓄積し，そのデータを用いて品質に関する警告および品質に関する情報の表示を行う．私たちは日立に提示して頂いた構成図をひな形にシステムの構成（図 3-1）を決定した．システムの流れは次の通りである．まず品質に関する各種開発データがデータアダプタを通じて読み込まれる．次に読み込まれたデータがデータ分析にかけられる．そのデータは管理者や開発者といったユーザがブラウザで表示を行う際に表示処理の部分で使用される．また処理エンジンによって統計処理されユーザに情報の表示が行われる．また同時に処理されたデータがルール解析に入力され，品質低下の要因となりうるものを発見次第ユーザへ通知する．

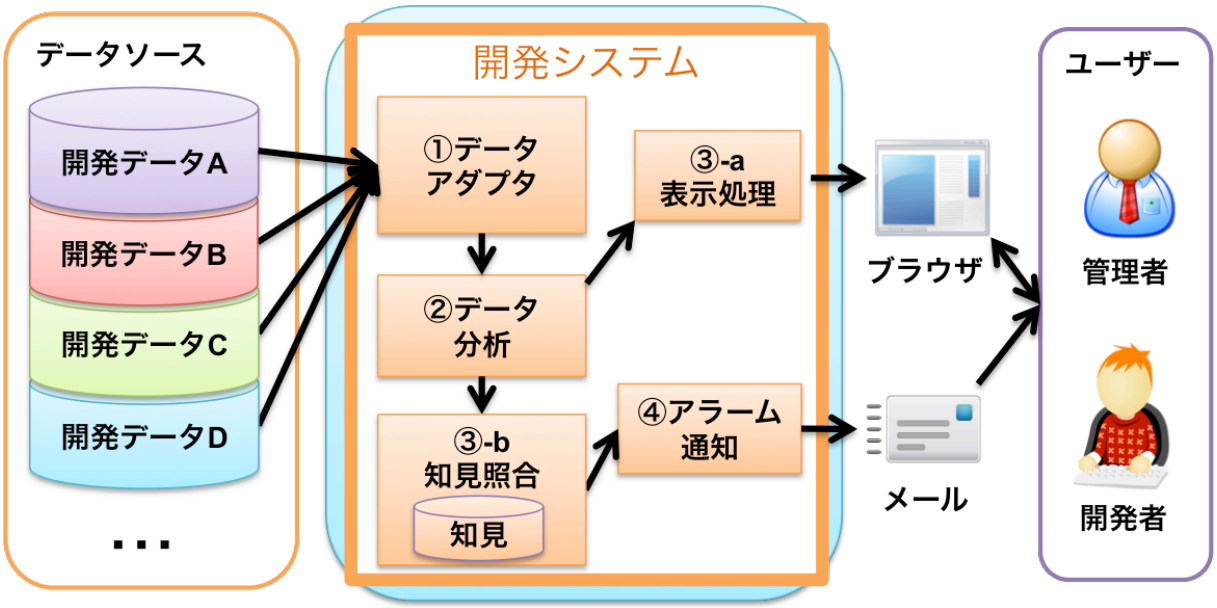


図 3-1 システム構成図

3.3 システム利用の流れ

まず利用者がソフトウェア構成管理システムやチケット管理システムなどを使いソフトウェア開発を行いそれらのデータを定期的に出力すると，システムはそれらのデータを分析し利用者に品質に関する警告を行う，利用者はそれを受け品質の低下と思われる要因に対処する．よってシステムはソフトウェアを開発するプロジェクトに導入して使用する．システムの利用の流れの一例は以下の通りである．このシステムはソフトウェアの品質に実現過程[8]において設計書とコードによる品質の作りこみをサポートするため，設計書とコードの欠陥に対するレビューに相当する部分を担う．それ以後の部分に相当するコードのソフトウェアフォールトに対するテストは行わない．

1. 日々の開発でデータが蓄積される
   (ア) 開発者が開発した成果をソフトウェア構成管理システムにコミットする
   (イ) 開発者が発生したバグをチケット管理システムに登録する
   (ウ) 開発者は解決したバグをチケット番号と共にソフトウェア構成管理システムにコミットする
2. 1 で記録されていくデータソフトウェア構成管理システムとチケット管理システムのデータを既存のソフトウェアを使い場合によっては分析し出力する
3. 2 で出力されたデータを本システムは一時間ごとに分析し，品質低下の兆候が見られ次第そのプロジェクトに属する人間に警告を行う
4. 3 の警告をプロジェクトの管理者，もしくは開発者が受け取り，品質低下の兆候を把握する
5. 4 で把握した人間が本システムを使い品質に関連する情報を閲覧する
6. 3 で警告された人間が 4 と 5 により品質低下に対処する

7. 6 により品質が向上する

3.4 システムの機能

私たちは当初このシステムで実装する必要がある機能として，大きくユーザの管理，データ収集と蓄積，そしてそのデータを使った警告と表示に分けられる機能を挙げた．個別の機能については表 3.1 のように決定した．実際の開発においてもこれらの全ての機能について開発が行われた．

表 3.1 システムの機能一覧

機能要件	詳細	優先度
インストール・アンインストール	システムをサーバーやパソコンに導入するがことできる	低
オンライン操作	ユーザが Web ブラウザを通してシステムを操作できる	高
データ収集	リアルタイムに変更・追加されたデータを読み込むことができる	中
データ分析	様々なデータから品質に関わるメトリクスを引出すことができる	中
分析結果の表示	分析したメトリクスをプラウザで表示できる	中
ルール照合	分析したメトリクスと予め得たルールを照合する	中
アラーム通知	照合した結果の警告をプッシュ配信することができる	中
データ管理	収集したデータを管理するができる	中
システム設定	システムの環境設定をオンラインで変更できる．	低
ユーザ登録	システムの管理者がユーザを登録できる．	低
ロール権限	システムの管理者がユーザのロール権限を設定できる．	低
グループ管理	ユーザグループの作成，変更，削除ができる．	低
ログイン・ログアウト	登録したユーザがシステムにログイン・ログアウトできる．	低
ロール個別の可視化設定	ユーザ毎に情報の表示項目が設定できる．	低
拡張性を実現する機能	将来の開発者が分析手法を変えたり，分析結果の表示を変更したりしやすいように設計する．	高

3.5 システムで用いる技術

システムに手を加えやすいことを念頭において用いる技術を選定した．表 3.2 に代表的な技術を示す．

表 3.2 システムで用いた主な技術

分類	技術
プログラミング言語	Ruby
ウェブアプリケーションフレームワーク	Ruby on Rails

第4章 本プロジェクトについて

4.1　　体制

本研究開発の関係者は表 4.1 の通りである．このプロジェクトでは学生四名が株式会社日立製作所（以下，日立）と連携し，また教員のご指導のもとプロジェクトを進めた．

表 4.1 関係者の一覧

所属	氏名
株式会社 日立製作所	居駒　幹夫 様
	土田　正士 様
	河合　亮 様
	河野　哲也 様
	白井　明 様
	中村　宇佑 様
筑波大学 学生	靳　松
	宮永　竜樹
	紀　冠男
	齋藤　創太（著者）
筑波大学 教員	早瀬康裕 助教
	天笠俊之 准教授

4.2　　スケジュールと実績

本研究開発では７月中旬まで企画提案を行い，その後８月中旬までデータ分析，それが終わり次第１２月中旬までシステム開発を行った（図 4-1）．

私たちはその後，データの仮説検証が長引くことが予想されたため，データの仮説検証を９月末まで延長すると共に，チーム内を２班に分け同時に開発を進めることとした．それによってリスケジュールされたスケジュールが図 4-2 となる．

ここで示したスケジュールは大きく４つに分かれているが，それぞれの期間に主に行ったことを簡単に説明する．プロジェクトが開始した５月中旬から６月上旬までは主にプロジェクト計画を行った．この期間にチームとしてプロジェクトを進めるために役割分担や全体のスケジュールの決定，チケット管理といった手法の導入，また事前知識の調査と共有を行った．６月上旬から７月上旬までは主に企画提案を行った．この期間に開発構想書の作成を行った．７月上旬から８月中旬までは主にデータ分析を行った．この期間にシステムが警告を出すルールを決定するため，どのようなデータが品質の低下を示すかという仮設の作成と検証を行った．またその検証結果を定量化しシステムで警告を出せる形に落とし込んだ．８月中旬から１２月中旬までは主にシステム開発を行った．この期間にシステムの開発とそれに付随するドキュメントの作成，また日立に試用して頂くためのシステムの導入を行った．現在は既に完成の１２月２０日を迎え試用導入を済ませており，試用導入の際に発生したバグの修正などを行っている．

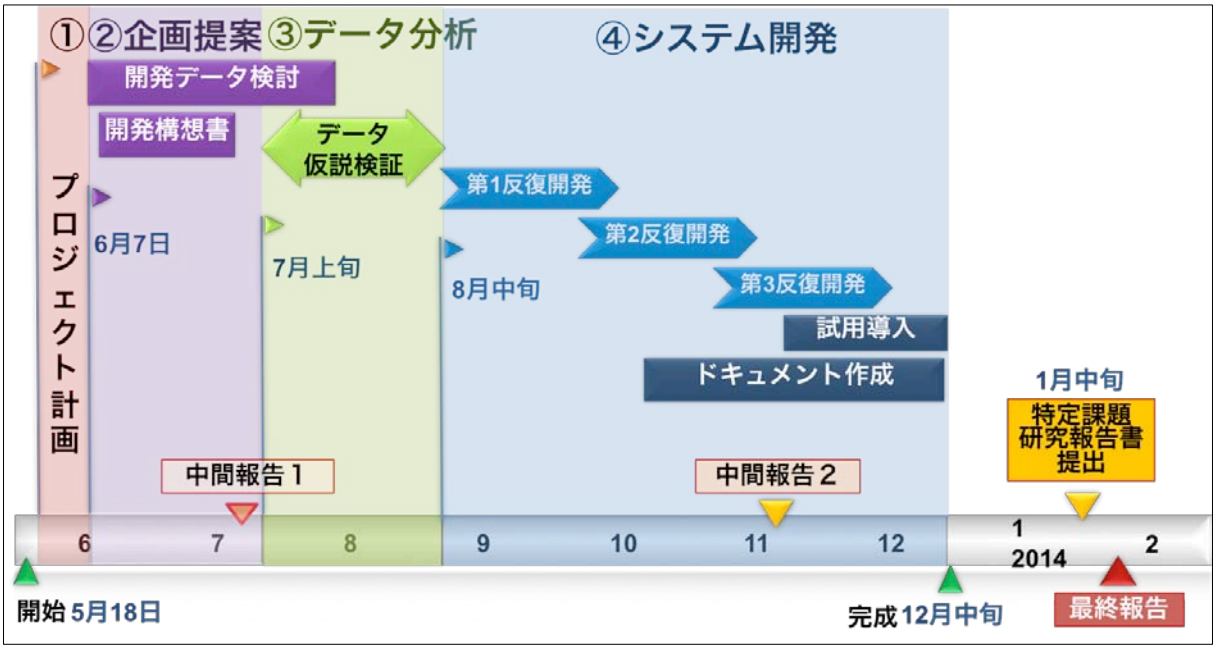


図 4-1 当初のスケジュール

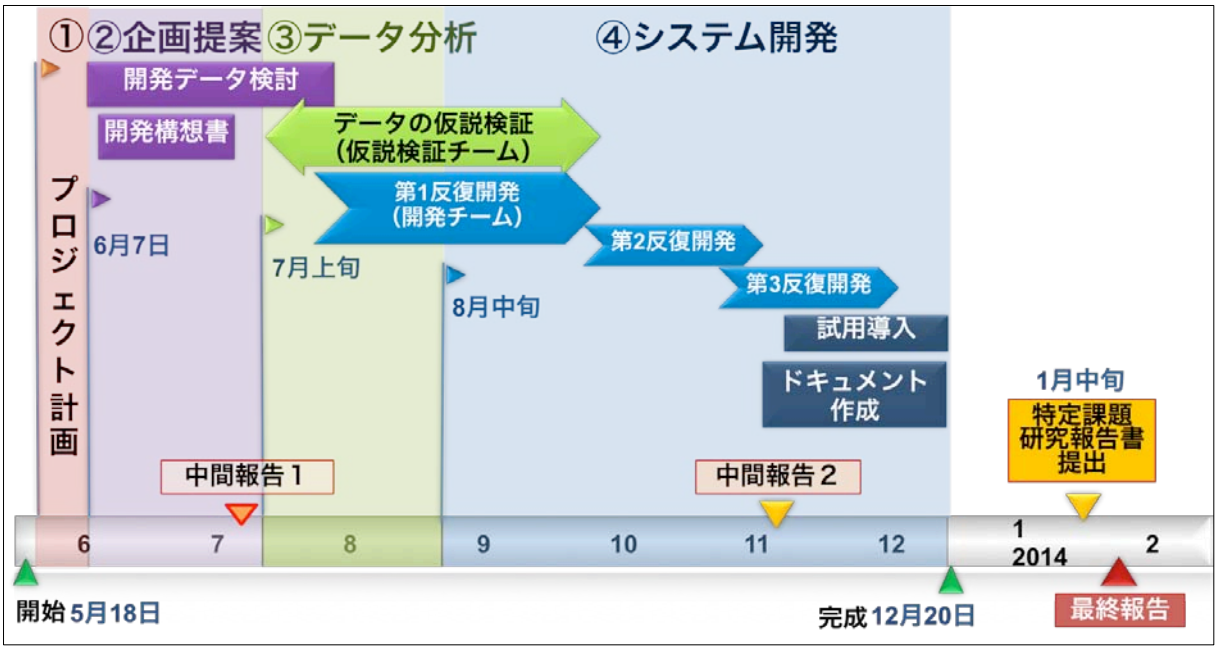


図 4-2 リスケジュールされたスケジュール

第5章 チームの立ち上げと提案活動

5.1 チームの立ち上げ

5.1.1 役割分担

チームの役割分担は主にマネジメントをキンが行い，データ分析ではキンと宮永が中心になり，システム開発では紀と齋藤が中心になった．詳細な役割について表 5.1 に示す．

表 5.1 チーム内の役割分担

	キン松	紀冠男	宮永竜樹	齋藤創太
プロジェクト計画	主責任	起案	起案	起案
マネジメント	主責任			
関連知識調査	実行	実行	実行	実行
開発環境構築				主責任
企画提案	主責任	起案	起案	起案
データ選定	起案	起案	起案	起案
データ分析	主責任		実行	
システム設計		主責任		主責任
システム実装	支援	主責任	実行	主責任
システムテスト	実行	実行	実行	実行
システムレビュー	主責任	実行	実行	実行
ドキュメント作成	主責任	実行	実行	実行
試用導入	支援	実行	支援	主責任
議事録			主責任	
起案：草案を作る 実行：作業を実行する 主責任：リーダーシップを発揮ながら責任を持って実行する 支援：仕事を順調に進められるようにサポートする				

5.1.2 手法の導入

チケット駆動型の開発を導入し，週次でチケットを発行し活動を行った．またソフトウェア構成管理システムを導入し，ソフトウェアのソースコードとドキュメントの管理を行った．チームで活動するにあたりどのようなタスクがあるか，それは誰がやるべきか，現在どのような状況か把握する必要があった．そこで私たちは Redmine[11]というチケット管理システムを導入し，週次でチケットを発行すると共に以前発行したチケットを見直すことで，本研究開発を進めていった．またソフトウェアのソースコードとドキュメントの管理にはソフトウェア構成管理システムの Git[12]を導入した．これによりソースコードを一元管理すると共に成果の統合を作業フローに取り込んだ．

5.2　　提案活動

5.2.1　　取り扱うデータと狙う効果の洗い出し

私たちはこのシステムで取り扱うデータを決めるため，ソフトウェアの開発の現場で存在すると考えられるデータとそれらのデータをどのように活用することができるかを考えた．私たちはソフトウェアの開発の現場で存在すると考えられるデータを 4M（人，機械，材料，方法）と SECI[13]の観点から開発過程で蓄積されるデータをマインドマップの形式で洗い出した（図 5-1）．次にステークホルダーの観点からシステムのメリットを洗い出した．（図 5-2）　そしてその二つをすり合わせ，それらのデータを元にシステムでどのような効果を狙えるかを洗い出した．

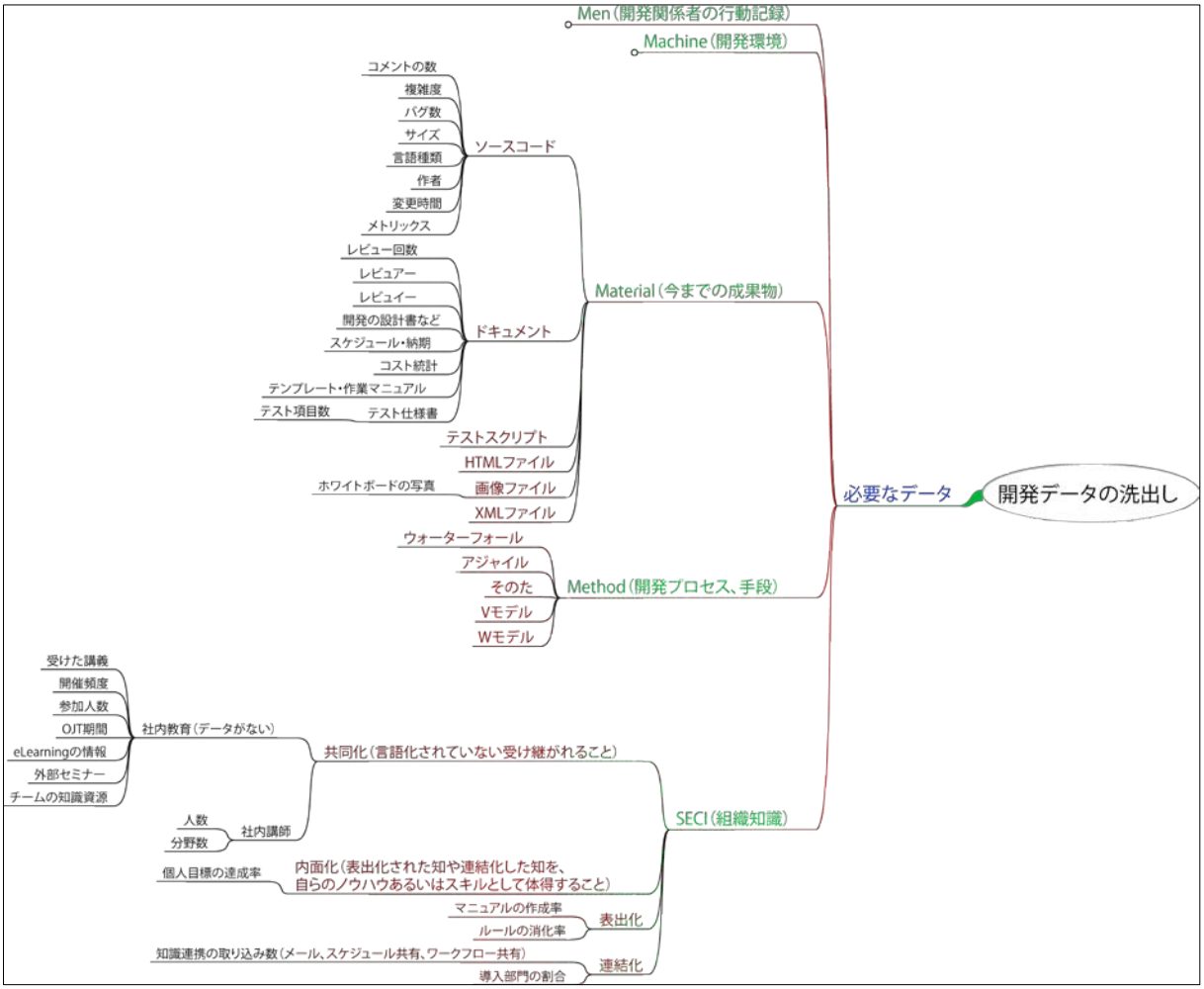


図 5-1 開発の現場に存在すると考えられるデータ

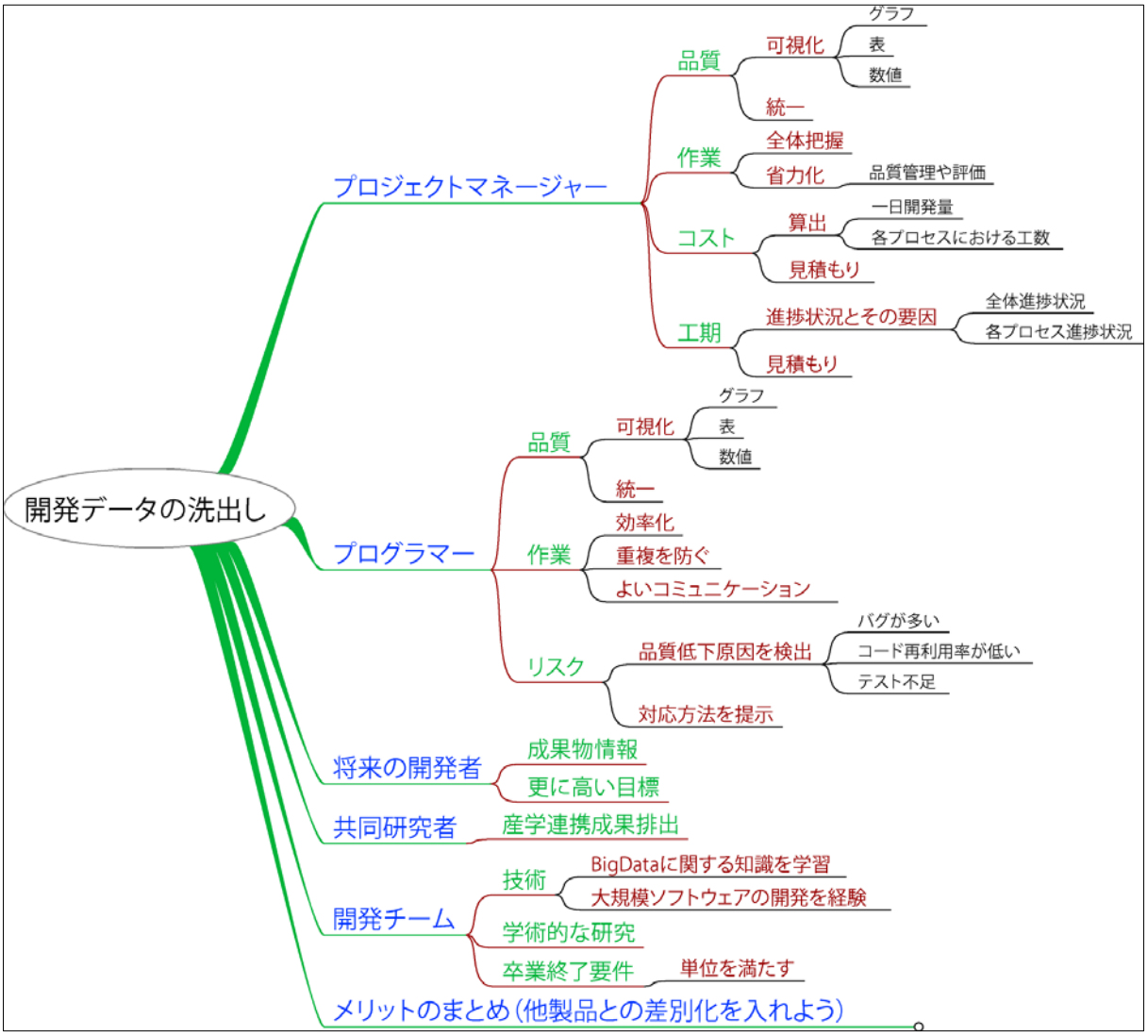


図 5-2 ステークホルダのメリット洗い出し

5.2.2 開発構想書の作成

私たちは本研究開発の目的を定める開発構想書をチームで分担し，日立にレビューを頂きながら作成した．私たちはまず前項で洗いだした結果を使い，成果物の開発目的、使用者やステークホルダー、各種の要件、制約事項などを明確にするための開発構想書をチームで分担し作成した．この間何度も日立にレビューを頂き修正を繰り返した．

第6章 研究活動と開発活動

6.1 研究活動

6.1.1 仮説の作成と検証

品質に関連する要素とその要素の変化について組織内のどのようなデータから得られそうかモデル化[4]を行った．そこから仮説を 23 項目作成し，検証が可能なものについて検証を行った．

6.1.2 検証結果の定量化

前項で得られた検証結果をアドバイザの方々と共有し，システムでどのような時に警告を出すか決定した．以下はシステムが警告を行う閾値を記したものである（表 6.1）．ファイルに対して以下のメトリクスが一つでも閾値を超えると，そのファイルには不具合が含まれている可能性が高いと判断し利用者に警告を行う．

表 6.1 システムが警告を行う閾値の一覧

メトリクス	閾値
複雑度の総和（Sum Cyclomatic）	100
複雑度の総和（Sum Essential）	100
クラスの結合度	80
インスタンス変数の数＋クラス数	50
メソッド数	110
実行可能パス数	500000
制御構造の最大ネストレベル	12

6.2 開発活動

開発活動では以下の図 6-1 に示すようにシステムが持つユースケースを三分割し反復型開発[1]を行った．反復の間には日立にシステムをレビューして頂き，方向性が誤っていないか，また改善点などのご助言を頂いた．

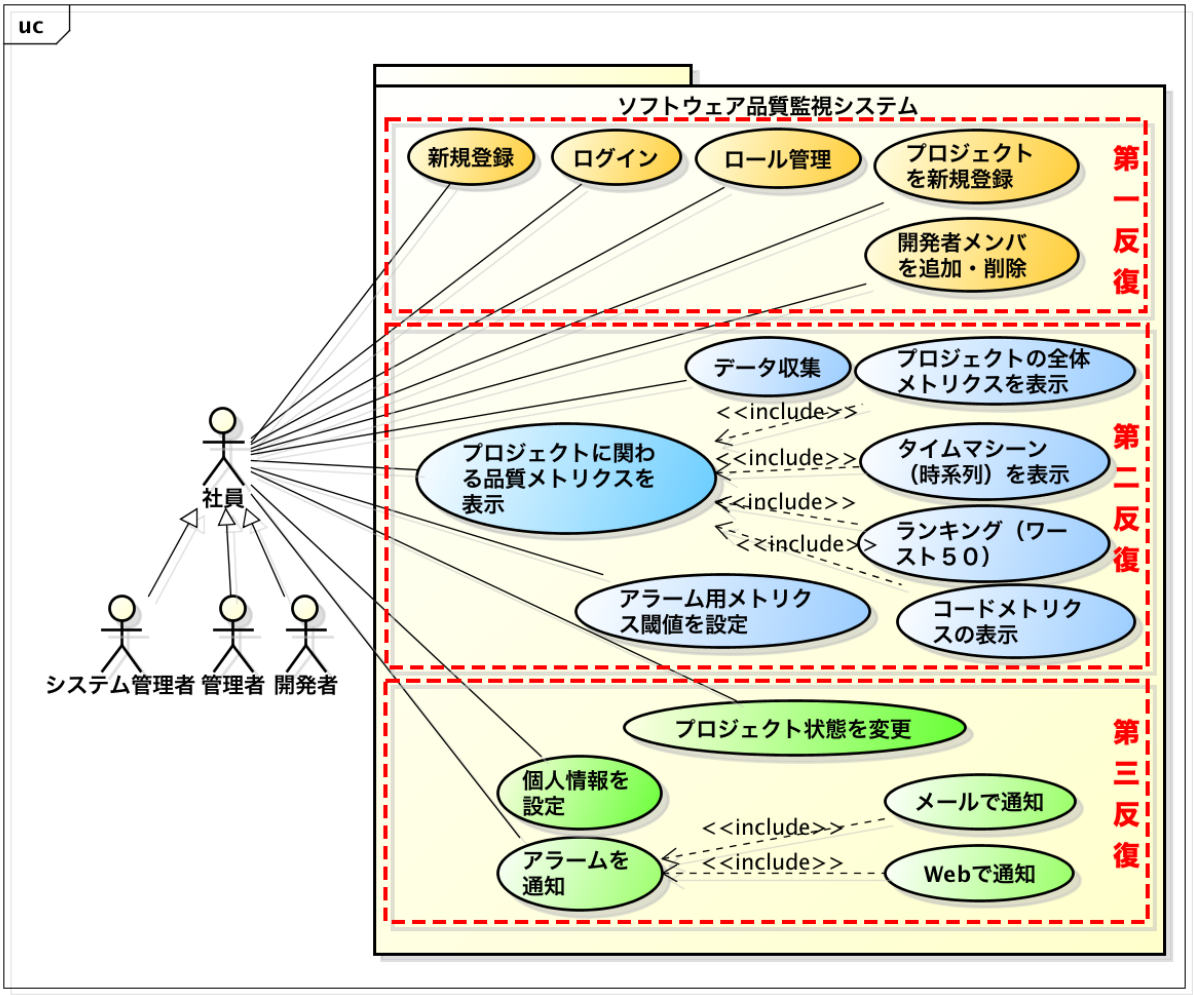


図 6-1 各反復で開発する機能

6.2.1 第一反復

第一反復ではシステムを企業内で運用していくのに必要となる，ユーザ権限管理機能とプロジェクト管理機能について開発と設計を行った．このシステムが企業内で運用され複数のプロジェクトでの使用を想定していることから，プロジェクト管理機能は必要不可欠となっている．またプロジェクトによっては非公開とし，プロジェクトメンバーのみの閲覧を許すということを日立と合意をしていたので，ユーザ権限管理機能も必要になった．第一反復では私がそれらの機能の設計と開発を行った．

6.2.2 第二反復

第二反復ではこのシステムに必須となるデータ読み込み蓄積機能について設計と開発を行った．また既存システムの一部代替となりうるよう，いくつかの観点から蓄積されたデータを閲覧できる機能，メトリクス表示画面を開発した．このシステムでは蓄積されたデータを使ってユーザへ警告を行うことから，そのためにデータをこのシステムへ読み込みおよび蓄積することが必要不可欠となっている．私はこのデータ読み込み蓄積機能の設計と開発にお

いて主導を行い，読み込む４つのデータ（表 6.2）のファイルの内 XML フォーマットのもの２つについて担当し，紀が開発した残り 2 つの CSV フォーマットのものと共に取りまとめを行った．

また日立との話し合いで SonarQube という既存システムが存在することをご教示頂き，SonarQube の一部機能を開発した．そこでは私たちのシステムを使用するに当たり SonarQube といった既存システムとの併用が必須になってしまうとその分導入や利用する手間が増えてしまうという示唆を日立に示して頂いた．そこで日立と話し合い品質を監視できるシステムとして一般に必要な機能を洗い出しそれらを開発した．それらの機能には SonarQube も存在する一部機能と重複している部分もあるが，ここでは SonarQube を併用せずとも済むというのも目的に含まれていたので，既存システムに実装されているしないに係わらず開発を行った．開発を行った機能は表 6.3 の４つである．

表 6.2 システムで使用するデータ一覧

出力に使用するプログラム	形式	内容
Understand	CSV	ソースコードのメトリクス
Redmine	CSV	チケット（ここではバグ）情報
svn log	XML	コミットログ
StatSVN	XML	コミットログの統計

表 6.3 メトリクス表示画面の一覧

画面名	機能
時系列メトリクス表示画面	一つのプロジェクトにおけるメトリクスの変動を表示する
スナップショットメトリクス表示画面	一つのプロジェクトにおける特定時点のプロジェクト全体におけるメトリクスを表示する
メトリクスランキング表示画面	一つのプロジェクトにおける特定時点のメトリクスのランキングをいくつかの指標で表示する
個別メトリクス表示画面	一つのプロジェクトにおける特定時点のクラス，ファイル，メソッド単位のメトリクスを表示する

6.2.3 第三反復

研究活動で得た知見を用いて品質低下の兆候を警告する機能を作成した．このシステムの大きな課題の一つである品質低下の兆候が見られ次第利用者に警告する機能そのものをこの反復で設計および開発を行った．このシステムでは設定された閾値を上回ると利用者に警告が行われるようになっており，閾値の設定部分を宮永が，その設定を使いデータを分析し警告データを生成する部分を齋藤が，生成された警告データを使いメールで利用者に警告を行うと共にウェブでも表示する部分を紀が行った．

第7章 著者のプロジェクトに対する貢献

ここでは著者のプロジェクトに対する貢献についてユーザ権限管理機能とデータ読み込み蓄積機能の設計と開発の観点から述べ，次にその他の貢献について記述する．

7.1 ユーザ権限管理機能の設計と開発

著者は開発活動の第一反復においてシステムを企業内で運用する際に必要となってきたユーザ権限管理機能の設計と開発を行うことで，本システムで定めた要件の達成に貢献した．ここで開発に携わった機能を図 7-1 に示す．私はサーバ側の設計と開発を主に担当し，紀が画面設計とその開発を担当した．

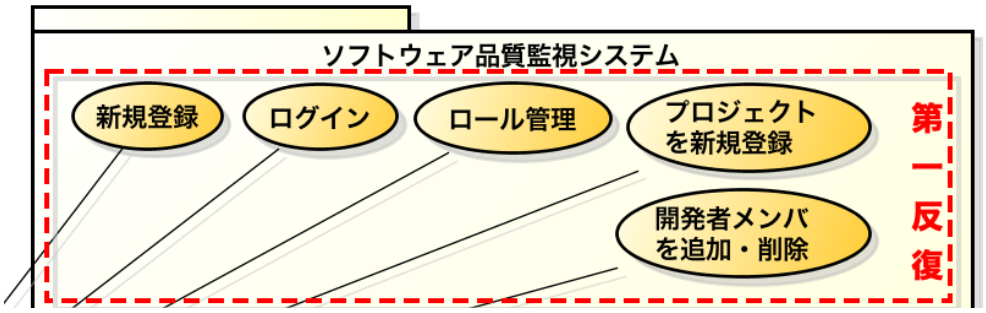


図 7-1 第一反復で携わった機能

7.1.1 提案作成

私はユーザ権限管理機能の詳細について複数案を作成し日立に提案し詳細について確定させた．次にヒアリングを行った過程について示す．

私たちはシステムを日立，ひいては一企業の社内で実際に運用できることを目指して研究開発を行っていた．それは私たちが日立に合意を取った開発構想書にも，機能要件の詳細の項にユーザ登録，ロール権限，グループ管理，ロール個別の可視化設定という形で記述されていた．しかしこれだけでは実際にどのようにユーザ権限管理機能を構築するのか詳細な部分での合意が取れていなかった．そこで私はユーザ権限管理機能についていくつかの案を作成した上で日立にヒアリングを行った．

まず私はこのシステムでどのような権限の存在が考えられるか洗いだした．このシステムに存在する要素は大きくユーザとプロジェクトが考えられる（図 7-2）．なぜなら開発構想書に記されている想定使用者としてプロジェクト管理者が挙げられ，またこのシステムは企業内で使われることを想定していた．通常，企業では複数のプロジェクトが平行して行われていると考えられるので，このシステムではプロジェクトという概念を扱えるべきだと考えた．またプロジェクトにはそれぞれ人間が所属することから，同じくユーザという概念を扱えるべきだと考えた．ここまででユーザとプロジェクトという要素がシステムには存在することとなり，これらに一つ一つ対しデータ操作における CRUD の観点から Create, Read, Update, Delete という権限の存在が考えられた．またユーザとプロジェクトの間には所属という概念が前述の様に存在する（図 7-3）．この所属についても同じく 4 つの権限が存在すると考えられる．これよりこのシステムには３×４＝１２個の権限が存在する可能性が考えられた（表

7.1)．

次にユーザが持つ役割について洗い出しを行った．ユーザが持つであろう役割の一つにプロジェクト管理者が挙げられる．これよりプロジェクトには管理者以外の人間もいることが考えられ，ここではそれをプロジェクト開発者と呼ぶ．またそれ以外に表 7.1 を考えると，ここで言うプロジェクト「そのもの」を管理する人間がいると考えられる．これをここではシステム管理者と呼ぶ. これよりそれぞれのプロジェクトに対して管理者と開発者が存在し，またこのシステムにはシステム管理者がいることが考えられる．これよりこのシステムには以下の役割が存在することが考えられる（表 7.2)．

私はこれらの要素，権限，役割を基本にし，日立に提案する案を作成した．まず案を作成するにあたり考える必要があるのは，一人のユーザは複数の役割を兼ねることができるのかということである. 兼任できるのならば例えばユーザ A がプロジェクト A の管理者とプロジェクト B の開発者を兼任する場合が考えられる．この場合であれば，ユーザが複数の役割を持つ形になる（図 7-4)．ユーザが役割を兼任できない場合を図 7-5 に示す．ユーザが役割を兼任できない場合は必ずいずれかのプロジェクトの管理者もしくは開発者にのみ属することとなる．この場合，いずれかのプロジェクトの管理者もしくは開発者というグループがユーザを複数人持ち，それらのユーザは重複しないこととなる．しかし複数のプロジェクトを兼任するのは通常あることだと著者は考えた．しかしこの点については著者の思い込みとも考えられるので，まずは兼任すると仮定して案を作成することにし，最終的には日立に確認を取ることとした．

次に考えなければならないのはプロジェクトの管理者または開発者とシステム管理者を兼任することがあるかということである．これについては実際どのように運用がなされるか推測が難しかったため，こちらも兼任すると仮定して案を作成し，改めて日立に確認を取ることとした．

私は以上よりユーザ権限の一案図 7-6 を作成した．送付する資料は穴埋め方式にすることも考えられたものの，全て先方に任せてしまうのはお手数をかけてしまうと思い，私がたたき台として全ての項目を予め埋めて送付した．この項目を埋める際に留意したのは日立の社内で使いやすい様にすることである．今回プロジェクトという概念を設けることが予め決まっており，また管理者と開発者という概念を開発構想書で取り入れていた．その中で私は秘密保持の観点から各人が最小の権限を持つように案を作成した．それが図 7-6 である．

å

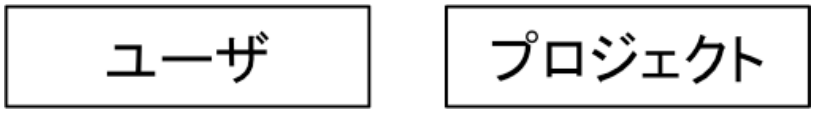


図 7-2 システムに存在すべき要素

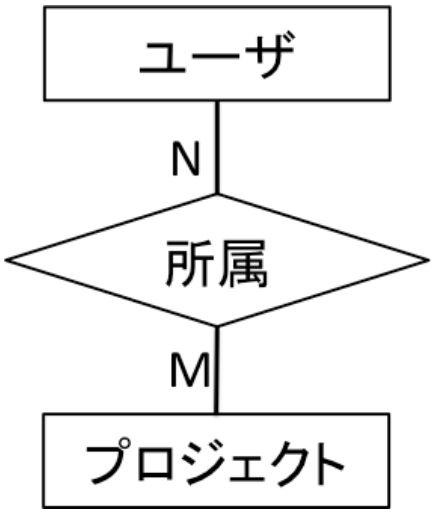


図 7-3 要素の関連

表 7.1 システムに存在すると考えられる権限

	Create	Read	Update	Delete
ユーザ				
プロジェクト				
所属				

表 7.2 プロジェクトに存在すると考えられる役割

プロジェクト A	管理者	開発者
プロジェクト B	管理者	開発者
⋮		
システム管理者		

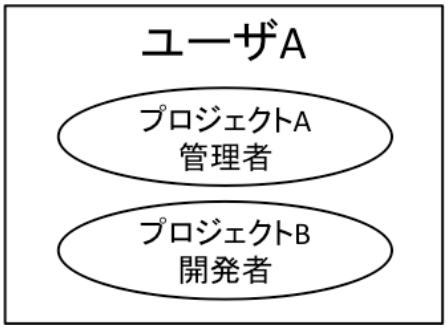


図 7-4 ユーザが役割を兼任できる場合

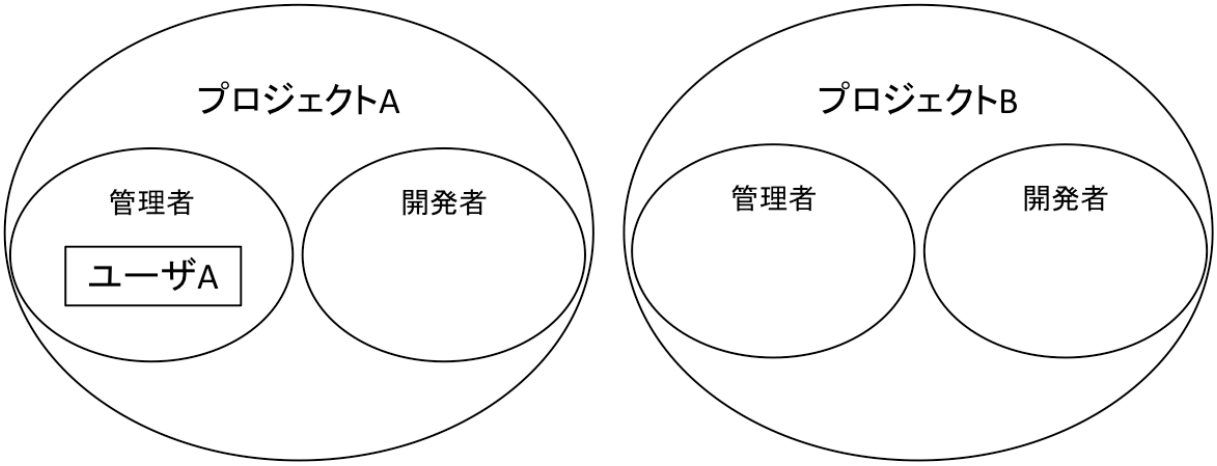


図 7-5 ユーザが役割を兼任できない場合

システムのロールとその権限について

	プロジェクト自体の		プロジェクトAの				
ロール名	追加	削除	メンバー追加	メンバー削除	全体ページ閲覧	全個人ページ閲覧	自身の個人ページ閲覧
システム管理者	○	×	○	○	×	×	×
(プロジェクトAの)管理者	×	×	○	○	○	○	×
(プロジェクトAの)開発者	×	×	×	×	×	×	○

※補足
社員は複数のロールを持つことが出来ます
なおプロジェクトの管理者と開発者については、プロジェクトごとにロールが付与されます

図 7-6 ユーザ権限の一案

7.1.2 ヒアリング

私は前項で作成したユーザ権限の一案について日立に送付しご意見を伺い，詳細を詰めたことで顧客の意図に沿う設計に貢献できたと考えている．私は前項で作成したユーザ権限の一案を日立に送付し，ミーティングにてご意見を伺った．その結果，プロジェクトやメンバーの追加や削除は案の通りだったものの，日立は基本的に全ての情報を公開することを運用の上で考えていらっしゃることが分かった．またそれ以外にも基本的には公開にするものの，プロジェクトによっては非公開にしたいということ．また管理者が非公開のプロジェクトを今後公開にすることが考えられるということを伺った．これらは日立の社内で運用する際に企業体制と絡んで発生する問題であり，ヒアリング無しに把握することは難しかったと考えられる．これによってプロジェクトにそのプロジェクトが公開か非公開かを示す属性が必要となった（図 7-7）．このヒアリングによって見切り発車で実装を進めてしまうことによる手戻りなどを防ぐことができた．

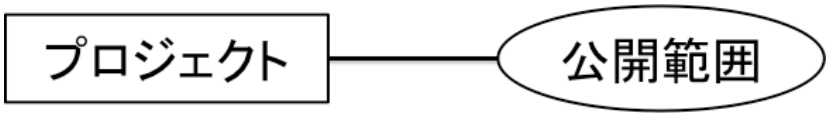


図 7-7 ヒアリングの結果新たに必要となった属性

7.1.3 設計

前項のヒアリング結果を反映したのが図 7-8 である．非公開プロジェクトを除き閲覧に関しては全てのロールに対し権限を与えることとなった．

	プロジェクト自体の		プロジェクトAの			公開プロジェクトの	非公開プロジェクトの
ロール名	追加	削除	メンバー追加	メンバー削除	閲覧	閲覧	閲覧
システム管理者	○	×	○	○	○	○	×
プロジェクトAの管理者	×	×	○	○	○	○	×
プロジェクトAの開発者	×	×	×	×	○	○	×

図 7-8 決定したユーザ権限

7.1.4 開発

前項で設計したものを実際に開発した．ここでは保守性を上げるためにいくつかの著名なライブラリを組み合わせて使用した．これによって保守性が向上したと考えている．ここで使用したライブラリは GEM Library[14]である Devise[15]と CanCan[16]である．これらのライブラリは Ruby Toolbox[17]というライブラリのダウンロード数を集計するサイトで Rails Authentication と Rails Authorization のカテゴリでそれぞれ首位を獲得しており，広く使われていると考えられたので使用した．広く使われていることでライブラリが今後保守されたり，また私たちが変更を加える際に参考となる文献などが多いことから首位であるものを選択した．また実装の際にもそれぞれの公式サイトを閲覧しその方法に則って開発を行った．これにより極端な実装を避け，保守しやすいシステムになったと考えた．

7.2 データ読み込み蓄積機能の設計と開発

私はデータ読み込み蓄積機能の設計と開発を主導し紀と共に行った．データ読み込み蓄積機能はこのシステムを稼働させる上で必ず必要となる機能である．このシステムは蓄積された情報から品質を損なうと考えられる兆候を見つけ次第，プロジェクト買付者や管理者に警告をし品質の向上を図るシステムであるが，その「蓄積された情報」を作成するのがこのデータ読み込み蓄積機能である．これは第二反復で開発を行う機能であるデータ収集に相当する（図 7-9）．

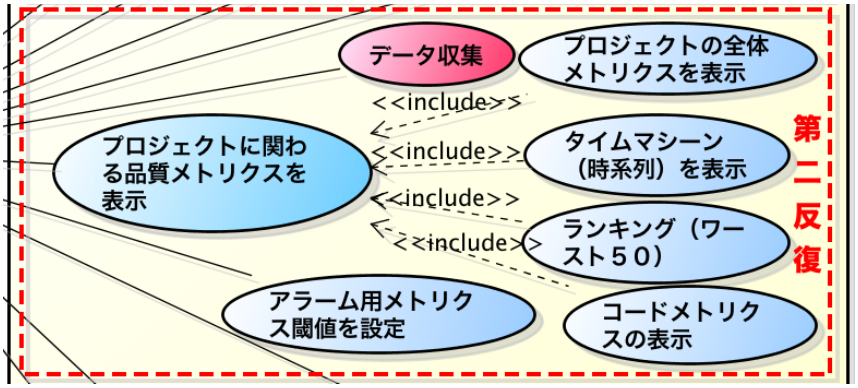


図 7-9 データ収集機能の相当する位置

7.2.1 設計

このデータ読み込み蓄積機能はこのシステムの中核となる部分であり，また今後拡張される可能性があったので，他の部分よりも設計に気をつけ設定を行った．今後考えられる拡張しては，分析手法の追加や分析データの追加などが考えれる．そのために読み込むデータの情報が失われないこと意識した．

まずここで挙げられるデータについて説明する．このシステムで取り扱うデータは表 6.2 でも示した下のデータである．

- バグ情報（チケット管理システムの Redmine より）
- ソースコードを静的分析したメトリクス情報（ソースコード静的分析ツールの Understand[18]より）
- リポジトリのログ情報（Subversio[19]リポジトリのコミットログを出力する svn log コマンドによる）
- リポジトリ分析情報（Subversion リポジトリを分析するソフトである StatSVN[20]より）

この設計で主に留意したのはデータを蓄積する際に情報が失われない点である．このデータ読み込み蓄積機能の設計時にはどのような分析を行いどのような警告を出すかが定まっておらず，またそれも今後拡張される可能性があった．よって私は入力された情報を取りこぼさないよう留意してデータベースの設計を行った．

まず具体的にデータベースの設計で留意した点は，構造化されたデータの構造を崩さずに読み込みおよび蓄積したことである．入力されるデータは CSV 形式のものと XML 形式のものがあったが，私が担当したものは XML であったので，必ずしも直列的なデータではなく一部が構造化されていた．そこで一つのデータに対し一つのテーブルではなく，一つのデータに対し複数のデータベースを用いてデータ構造を表すようにした（図 7-10）．ここで着目したのは関連の多重度である．この関連の多重度に着目して RDB のデータベースの構造を定義した．一対多の構造の場合は多の方のテーブルに一の方の主キーをもたせた．

また既存のライブラリを使用しなおかつデータを取りこぼさないようにした．リポジトリのログ情報の読み込みについては該当するライブラリがあったものの，一部の情報に対し対応する API が定義されておらず，そのまま使用するとデータを取りこぼしてしまうことが発覚した．そこで私はメタプログラミング[21]で既存のライブラリを拡張して対応を行った．これによって保守性とデータの取りこぼしの問題双方に対応できたように思う．

またこの部分については紀と共に開発するため，事前にデータを蓄積するテーブルにおいて必要な基本的な構造を考え設計した．データの読み込みと蓄積にあたっては，そのデータがどのプロジェクトのもので，またいつ分析されて蓄積されたものか記録する必要があった．そこで私は既存のプロジェクト情報を格納する projects というテーブルの他に分析した日時を記録する analyses というテーブルを作り，その間を一対多の関連で結んだ．プロジェクトと蓄積データの関連を図 7-11 に示す．私はそれに従ってデータの構造を以下の様に ER 図で書き起こした（図 7-12）．

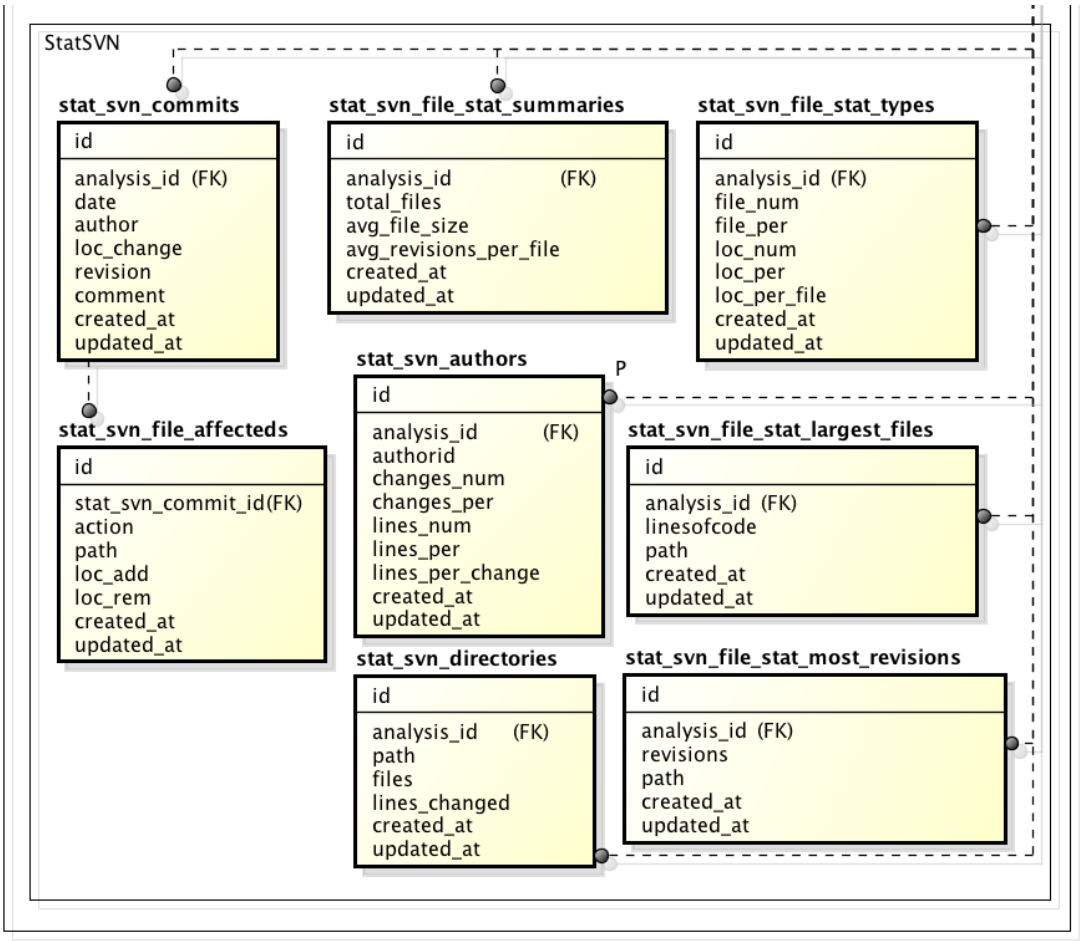


図 7-10 構造化されたデータにおける ER 図

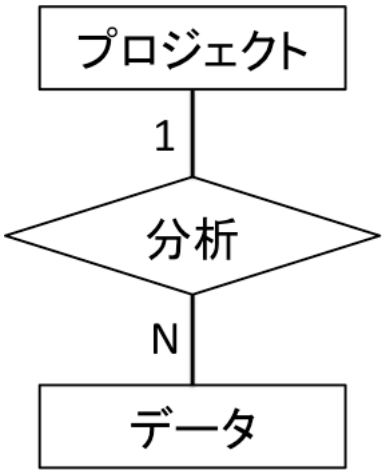


図 7-11 プロジェクトと蓄積データの関連

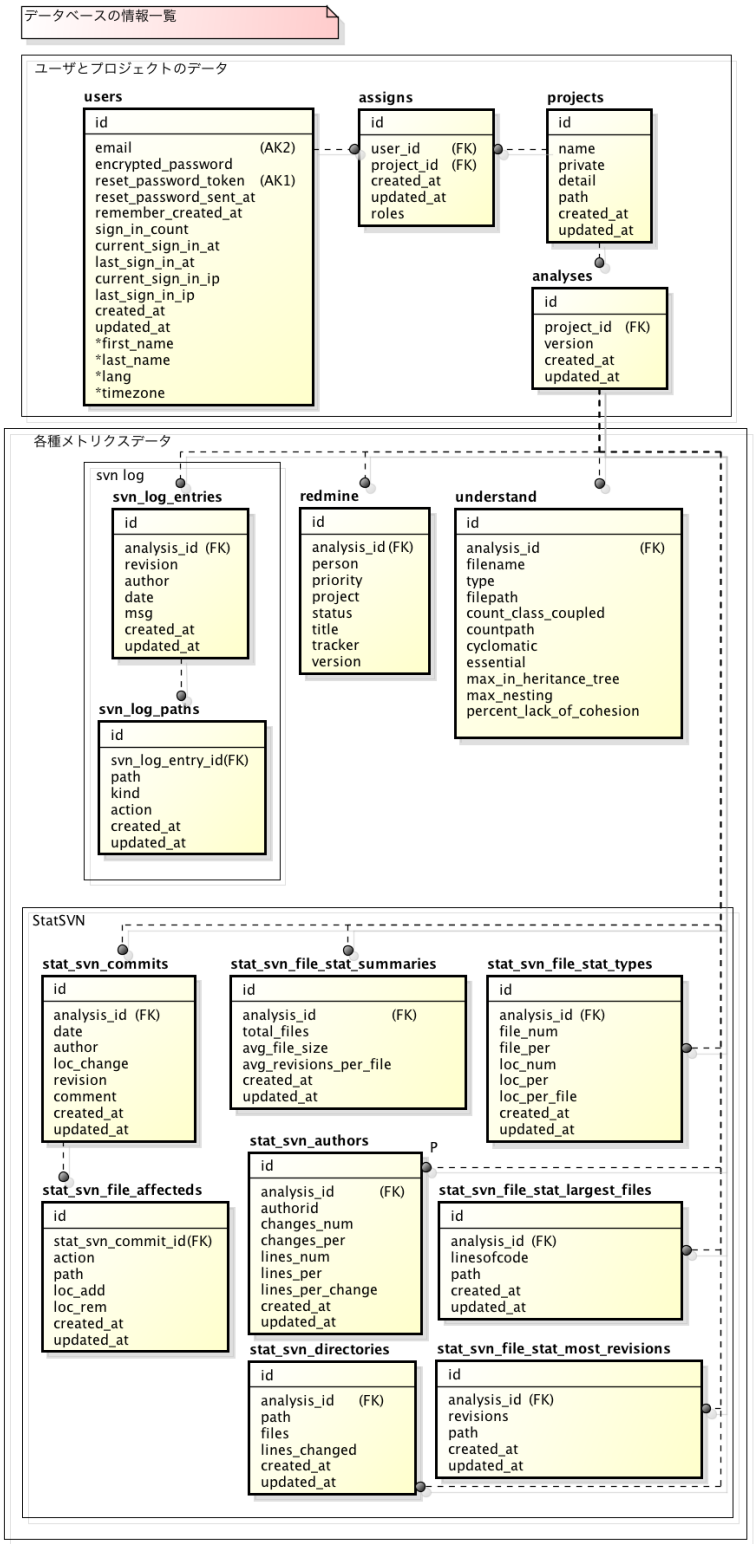

データベースの情報一覧
ユーザとプロジェクトのデータ
users
id
email (AK2)
encrypted_password
reset_password_token (AK1)
reset_password_sent_at
remember_created_at
sign_in_count
current_sign_in_at
last_sign_in_at
current_sign_in_ip
last_sign_in_ip
created_at
updated_at
*first_name
*last_name
*lang
*timezone
assigns
id
user_id (FK)
project_id (FK)
created_at
updated_at
roles
projects
id
name
private
detail
path
created_at
updated_at
analyses
id
project_id (FK)
version
created_at
updated_at
各種メトリクスデータ
svn log
svn_log_entries
id
analysis_id (FK)
revision
author
date
msg
created_at
updated_at
svn_log_paths
id
svn_log_entry_id(FK)
path
kind
action
created_at
updated_at
redmine
id
analysis_id (FK)
person
priority
project
status
title
tracker
version
understand
id
analysis_id (FK)
filename
type
filepath
count_class_coupled
countpath
cyclomatic
essential
max_in_heritance_tree
max_nesting
percent_lack_of_cohesion
StatSVN
stat_svn_commits
id
analysis_id (FK)
date
author
loc_change
revision
comment
created_at
updated_at
stat_svn_file_affecteds
id
stat_svn_commit_id(FK)
action
path
loc_add
loc_rem
created_at
updated_at
stat_svn_file_stat_summaries
id
analysis_id (FK)
total_files
avg_file_size
avg_revisions_per_file
created_at
updated_at
stat_svn_authors
id
analysis_id (FK)
authorid
changes_num
changes_per
lines_num
lines_per
lines_per_change
created_at
updated_at
stat_svn_directories
id
analysis_id (FK)
path
files
lines_changed
created_at
updated_at
stat_svn_file_stat_types
id
analysis_id (FK)
file_num
file_per
loc_num
loc_per
loc_per_file
created_at
updated_at
stat_svn_file_stat_largest_files
id
analysis_id (FK)
linesofcode
path
created_at
updated_at
stat_svn_file_stat_most_revisions
id
analysis_id (FK)
revisions
path
created_at
updated_at

図 7-12 ER 図全景

7.2.2 開発

そ私は設計を共有し，また一部について合意したあと分担し開発を行った．その際には基本的なテーブルの構造のみを合意した．これは第一反復において詳細について言語やフレームワークなどの情報を決めずに分担を行った所，それぞれの実装がバラバラになってしまい統合するのに手間がかかってしまったということに起因している．その時の反省からここではデータベースの基本的な構造を決め分担を行った．この際に取り決めたのはプロジェクトのある時点での分析を示す analysis テーブルの主キーを格納するカラムを持つということだけである．これにより統合時に再度私がそういったカラムを作成する必要がなくなり，二度手間を防ぐことができた．またそういった合意を事前に取ることで，あとで話が食い違うなど認識の相違が生まれにくくなったと考えられる．

7.2.3 取りまとめ

開発の成果を取りまとめモジュール化した．前項で設計を共有したことにより統合が容易になった．取りまとめる際には全ての蓄積機能を統一されたインターフェースで扱えるよう Rake タスク[22]の形にまとめ引数にファイルパスを取るようにした．自身のモジュールについては実際に開発を進めていく内に既にこのような形にしていたが，もう一人の開発者である紀の作成したものはコントローラに実装されていた．そこで私は後者についてもコントローラから引き剥がし Rake タスクの形に纏めた（表 7.3）．これにより同じインターフェースでデータの読み込みおよび蓄積が行えるようになり，それぞれのスクリプトが疎結合になり汎用性が向上すると共に保守性や今後の拡張が容易になったと考えられる．

表 7.3 作成した Rake タスクの一覧

タスク	コマンド
データの読み込みと蓄積	rake analysis:start
バグ情報の読み込み	rake "redmine:load[ファイルパス]"
ソースコードを静的分析したメトリクス情報の読み込み	rake "understand:load[ファイルパス]"
リポジトリのログ情報の読み込み	rake "stat_log:load[ファイルパス]"
リポジトリ分析情報の読み込み	rake "stat_svn:load[ファイルパス]"

7.3 その他の貢献

私は実装の主責任者として開発が円滑に進むように開発基盤の構築と共有をはじめとして開発の全工程に渡って必要となる部分の構築支援を行った．また実際の開発では一部分を担当し，各人が開発した成果の取りまとめの作業を担当した．他にも最終的にシステムを日立の環境で運用するために紀と共に先方へ赴いて中心的に作業を行うと共に，導入後のシステムに関するやりとりは私が中心に行っている．実装以外の部分については開発構想書の想定利用者の章を担当した．

7.3.1 開発基盤の構築と共有

私は開発の基盤となる環境の構築と共有を行い，チーム内および顧客との円滑な成果物の共有と開発期間の短縮，また実現場で活用できるシステムの開発に貢献したと考えている．

動作確認環境の構築

私は最終的に動作させる環境を想定し動作確認環境を構築することによって，顧客への納品を円滑にできたと考えている．開発にあたり私たちが使用できるコンピュータのオペレーティングシステムは Windows であったため，各自その環境を使い開発を行うことが考えられた．しかし日立への導入にあたってはオペレーティングシステムに Linux を使うことが予測された．そのように開発を行う環境と実際に動作させる環境の相違が発生すると考えられたので，私は円滑に導入を行えるよう，実際に動作させる環境に近いと思われる環境を用意しチームメンバーが随時その環境で動作の確認を行える環境を構築した．構築した環境で使用したソフトウェアは表 7.4 の通りである．

表 7.4 使用したソフトウェア

区分	ソフトウェア
オペレーティングシステム	Ubuntu 12.04.3 LTS
HTTP サーバ	nginx 1.4.1
アプリケーションサーバ	Passenger 4.0.7
プログラミング言語	Ruby 1.9.3
ウェブアプリケーションフレームワーク	Rails 3.2.13

顧客による動作確認環境構築

私は日立から逐次動作確認が行える環境（ライブデモ）を構築することで，ご助言を頂きやすい環境を作り，より目的に適ったシステムの開発に貢献できたと考えている．私は前項で作成した動作確認環境を日立からの閲覧を考慮して構築を行い，またその環境が外部者から閲覧できないようにするためシステムに Basic 認証を施した．これにより実際に顔を合わせずとも開発しているシステムの意見を貰うことが出来，より顧客が望んでいるシステムの開発を行えるようになった．また情報を秘匿することにより，未然に開発情報が流出する事態を防いだ．

開発環境の構築における支援

開発環境のひな形を構築し構築方法について共有したことにより他のチームメンバーが開発を始める支援をできたと考えている．私は開発環境のひな形を構築するためチームメンバー全員が使用しているコンピュータを初期状態に復元し，そこから必要なソフトウェアを入れていき開発環境を構築した．その際には出来るだけ情報が見つかりやすいメジャーな開発環境を構築していった．その後，必要なソフトウェアとそのソフトウェアをインストールし設定する手順を Wiki に記述することで共有した．その中で自身で調査しチームメンバーが使用する大学から貸与されたコンピュータの環境で実際に環境構築と簡単なアプリケーションの構築を行い比較検討を行った．そして環境構築の容易さと開発の効率を指標にして開発

環境を選定したことにより，それらの向上に貢献したと考えている．また私が開発基盤の構築について一手に引き受けることにより，チームメンバーは他の作業に取りかかることが出来ると共に，開発を始めようとしたその日に開発環境の構築が可能となった．これによりプロジェクトの進捗が向上したと考えられる．

開発環境整備ドキュメント on Windows

編集 ウォッチ ロック 名前変更 削除 履歴

設定済みの部分は適宜飛ばして下さい。
分かりにくい部分は聞いて下さい。

初回起動設定

- コンピュータ名を設定
  - 先輩では"（苗字のローマ字）-npc"が推奨
- 無線LANに接続
  - 3F800-2に接続（パスワードは周りの人に聞く）
- 簡単設定を使用
- Microsoftアカウントを使わない
- ローカルアカウント
- 任意のユーザ名とパスワードを設定

初回起動後

- 指定されたUSBメモリを差し込む（齋藤から借りて下さい）
- 右上にノティファイが出てくるのでクリック
  - "4HiTs開発環境"フォルダを開いてファイルを表示
    - ファイルは全てこのフォルダにあります！
- フォルダ"4HiTs開発環境"に移動
- Firefoxっぽい実行ファイルをインストール
  - ブックマークは移行しない
  - 規定のブラウザに設定する
  - この種類のリンクを開く時は毎回Firefoxに設定する
- RailsInstallerっぽい実行ファイルをインストール
  - 全てデフォルトで進めていく
    - 完了後プロンプトが表示される
      - Gitで使う名前を入力
      - Gitで使うemailアドレスを入力
  - 完了
- TortriseGitっぽい実行ファイルをインストール
  - 全てデフォルトでインストール
  - 上記のLanguage Packっぽい実行ファイルをインストール
    - エクスプローラーで右クリック
    - TortriseGit→Settingをクリック
    - Languageから日本語を選択しOK
- 秘密鍵をコピー
  - Win+Eでエクスプローラーを表示
  - C→ユーザ→（ユーザ名）→.sshに移動
  - USBからid_rsa.ppkをコピー&貼り付け
- repoリポジトリをclone
  - エクスプローラーでC→Sitesに移動
  - フォルダに移動して何も無い所を右クリックしGitクローンをクリック
  - URLに"ssh://git-user-13@vm09.sit.cs.tsukuba.ac.jp/var/git/repo.git"と入力
  - Putty認証キーの右側にある[...]をクリック
  - 前項"秘密鍵をコピー"でコピーしたファイルを選択し開く
  - OK
  - はい
  - クローズ
- web-appリポジトリ（ウェブアプリケーション用リポジトリ）をclone
  - 前項"repoリポジトリをclone"と同じようにURL"ssh://git-user-13@vm09.sit.cs.tsukuba.ac.jp/var/git/web-app.git"をクローン
- Sublimeっぽい実行ファイルをインストール
  - デフォルトでインストール
  - *.sublime-menuというファイルをフォルダ"C:¥Users¥（自分のユーザ名）¥AppData¥Roaming¥Sublime Text 2¥Packages¥Default"に上書き貼り付け

SQLite Browserをインストール

- マイドキュメントにCtrl+Alt+N"Program Files"というフォルダを作成
- 上記フォルダにsqlitebrowserっぽいフォルダをコピーして貼り付け

開発時

- Sublimeの起動とフォルダオープン
  - Win+Qを入力後suと入力してsublimeを立ち上げる
  - メニューバーからファイル→フォルダを開く
  - 前でcloneしてきたweb-appのフォルダを選択し開く
  - これでファイルの編集準備は完了
- コマンドプロンプトの起動
  - Win+Qを入力後commと入力しCommand Prompt with Ruby and Railsを開く
  - 右を入力しエンター"cd web-app"
  - 同上"bundle install"
  - 同上"rails s"
  - ブラウザを開き右のURLにアクセスする"☆http://localhost:3000"
  - Hello Rails！

Rのインストール

- Rっぽい実行ファイルをインストール
  - デフォルトで

図 7-13 開発環境構築ドキュメント

7.3.2 データフロー図の作成とシステム構成の確認

私はデータフロー図を作成し（図 7-14）日立にシステム構成を確認することによって，日立との意思の共有ひいては実際に使われるシステム作りに貢献した．日立とのミーティング

の中で全体としてデータがどのように入力されどこでそのデータが出力されるのか全体として掴みづらいというお話があった．そこでデータフロー図を作成したところ，それを元に議論を進めることができた．

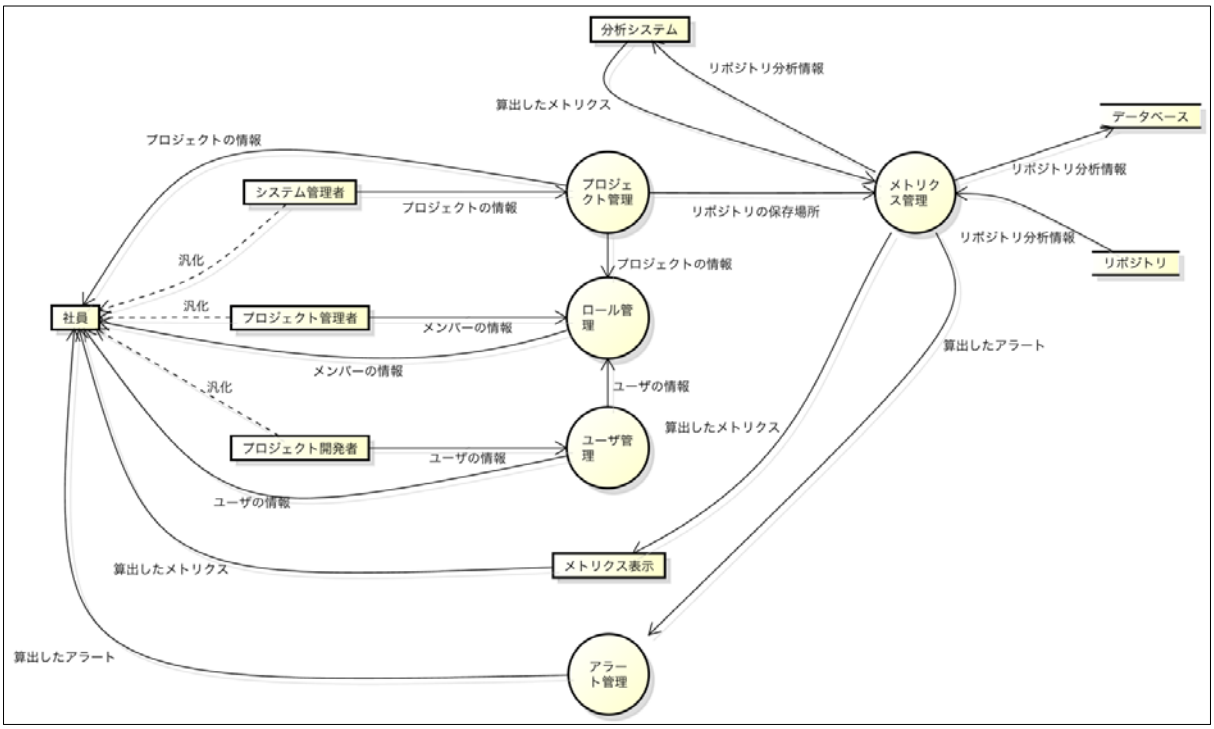


図 7-14 データフロー図

7.3.3 メトリクス表示画面の一部作成

私はメトリクス表示画面（表 6.3）の一部機能を担当及び開発することにより，顧客が望んでいるこのシステムの機能である，既存の代替手段の機能を併せ持つという要件を満たすことに貢献した．図 7-15 に私が開発を行った機能を赤で示す．

日立が望むこのシステムの要素として，既存の代替手段の代わりともなりうるという点があった．そこで挙げられていたのが既存の代替手段である SonarQube に実装されているメトリクス表示画面の一部機能である．今回の開発期間では代替手段が持っている全ての機能を実装することは難しかったので，顧客にヒアリングを行い顧客が必要としている主要な機能（表 6.3）を実装することとなった．

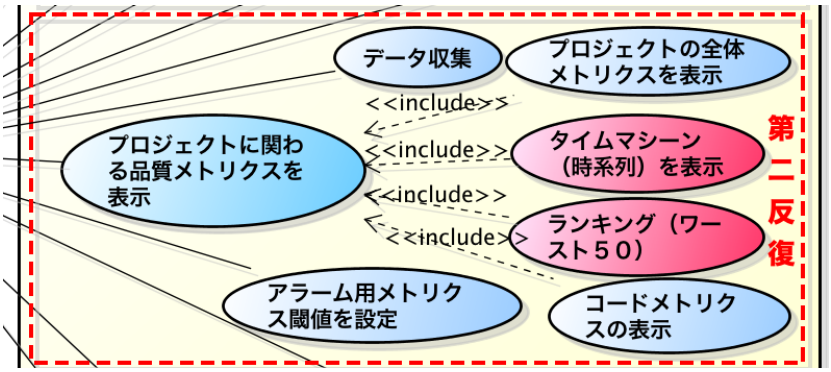


図 7-15 メトリクス表示画面にて作成を行った機能

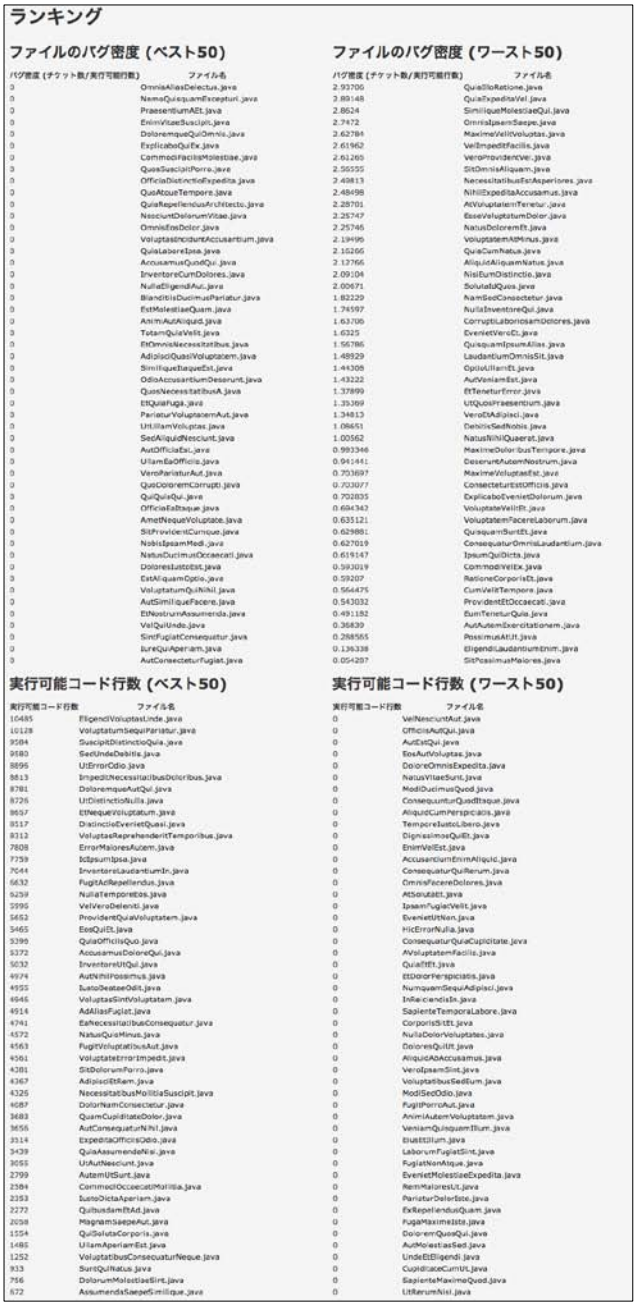
ランキング機能の実装

私は人やファイルに付随する品質に関わると思われる値をランキング形式で表示する機能を実装し，これにより品質に関して顕著な値を示すメトリクスを一覧できるようになった．サンプルデータを用いた画面例を図 7-16 と図 7-17 に示す．

私は日立との話合いの中から決まった以下のメトリクスのワースト 50 について実装を行った．

- ファイルの複雑度
- メソッドの複雑度
- 開発者がコミットした行数
- ファイルのバグ密度

上の２つについては間接的に品質に関わってくると思われることからランキング化の対象となり，３つ目については人に着目することでプロジェクトに所属する人間の興味を引くと共に新たな傾向を発見できないかということで開発の対象となった．また最後のバグ密度については品質の一つの指標として用いられていることもあり[9]対象となった．上の３つについては既に蓄積されているデータを SQL を使い絞り込み並び替えるだけで済んだが，最後の一つについては蓄積されているデータの中でもバグ情報のデータとコミットログのデータを結びつける必要があった．この部分について設計を行い実現した．



ランキング

ファイルのバグ密度 (ベスト50)

バグ密度 (チケット数/実行可能行数)	ファイル名
0	OmnisAliasDelectus.java
0	NemoQuisquamExcepturi.java
0	PraesentiumAEt.java
0	EnimVitaeSuscipit.java
0	DoloremqueQuiOmnis.java
0	ExplicaboQuiEx.java
0	CommodiFacilisMolestiae.java
0	QuasSuscipitPorro.java
0	OfficiaDistinctioExpedita.java
0	QuoAtqueTempore.java
0	QuiaRepellendusArchitecto.java
0	NesciuntDolorumVitae.java
0	OmnisEosDolor.java
0	VoluptasIncidunteAccusantium.java
0	QuiaLaboreIpsa.java
0	AccusamusQuodQui.java
0	InventoreCumDolores.java
0	NullaEligendiAut.java
0	BlanditiisDucimusPariatur.java
0	EstMolestiaeQuam.java
0	AnimiAutAliquid.java
0	TotamQuiaVelit.java
0	EtOmnisNecessitatibus.java
0	AdipisciQuasiVoluptatem.java
0	SimiliqueItaqueEst.java
0	OditoAccusantiumDeserunt.java
0	QuosNecessitatibusA.java
0	EtQuiaFuga.java
0	PariaturVoluptasemAut.java
0	UtUllamVoluptas.java
0	SedAliquidNesciunt.java
0	AutOfficiaEsc.java
0	UllamEaOfficiis.java
0	VeroPariaturAut.java
0	QuaDoloremCorrupti.java
0	QuiQuiaQui.java
0	OfficiaEaItaque.java
0	AmetNequeVoluptate.java
0	SitProvidentCumque.java
0	NobisIpsamMedi.java
0	NatusDucimusOccaecati.java
0	DoloresJustoEst.java
0	EstAliquamOptio.java
0	VoluptatumQuiNihil.java
0	AutSimiliqueFacere.java
0	EtNostrumAssumenda.java
0	ValQuiUndo.java
0	SintFugiatConsequatur.java
0	EtreQuiAperiam.java
0	AutConsecteturFugiat.java

ファイルのバグ密度 (ワースト50)

バグ密度 (チケット数/実行可能行数)	ファイル名
2.93706	QuiaEiloRatione.java
2.89148	QuiaTypeditaVel.java
2.8624	SimiliqueMolestiaeQui.java
2.7472	OmnisIpsamSaepe.java
2.62784	MaximeVelitVoluptas.java
2.61962	VelImpeditFacilis.java
2.61265	VeroProvidentVel.java
2.56555	SitOmnisAliquam.java
2.49813	NecessitatibusEtAsperiores.java
2.48498	NihilExpeditaAccusamus.java
2.28701	AtVoluptatemTeretur.java
2.25747	EsseVoluptatumDolor.java
2.25746	NatusDoloremEt.java
2.19995	VoluptasemAtMinus.java
2.15266	QuiaCumNatus.java
2.13766	AliquidAliquamNatus.java
2.09104	NisiEumDistinctio.java
2.00671	SolutaEQuos.java
1.82229	NamSedConsectetur.java
1.74597	NullaInventoreQui.java
1.63706	CorruptiLaboriosamDolores.java
1.6325	EvenietVeroEt.java
1.56786	QuisquamIpsumAlias.java
1.48929	LaudantiumOmnisSit.java
1.44308	OptioUllamEt.java
1.43222	AutVeniamEst.java
1.37899	EtTeneturError.java
1.35369	UtQuosPraesentium.java
1.34813	VeroEtAdipisci.java
1.08651	DebitisSedNobis.java
1.00562	NatusNihilQuaerat.java
0.993346	MaximeDoloribusTempore.java
0.941441	DeseruntAutemNostrum.java
0.703697	MaximeVoluptasEst.java
0.703077	ConsecteturEstOfficiis.java
0.702835	ExplicaboEvenietDolorum.java
0.694342	VoluptateVelitEt.java
0.635121	VoluptatemFacereLaborum.java
0.629881	QuisquamSuntEt.java
0.627019	ConsequaturOmnisLaudantium.java
0.619147	IpsumQuiDicta.java
0.593019	CommodiVelEx.java
0.59207	RationeCorporisEt.java
0.564475	CumVelitTempora.java
0.543032	ProvidentEtOccaecati.java
0.491182	EumTeneturQuia.java
0.36839	AutAutemExercitationem.java
0.288565	PossimusAtUt.java
0.136338	EligendiLaudantiumEnim.java
0.054297	SitPossimusMaiores.java

実行可能コード行数 (ベスト50)

実行可能コード行数	ファイル名
10485	EligendiVoluptasLinde.java
10128	VoluptatumSequiPariatur.java
9584	SuscipitDistinctioQuia.java
9180	SedUndeDebitis.java
8895	UtErrorOdio.java
8813	ImpeditNecessitatibusDoloribus.java
8781	DoloremqueAutQui.java
8726	UtDistinctioNulla.java
8657	EtNequeVoluptatum.java
8517	DistinctioEvenietQuasi.java
8312	VoluptasReprehenderitTemporibus.java
7808	ErrorMaioresAutem.java
7759	IdIpsumIpsa.java
7644	InventoreLaudantiumIn.java
6632	FugitAtRepellendus.java
5259	NullaTemporeEos.java
5195	VelVeroDeleniti.java
5152	ProvidentQuiaVoluptatem.java
5165	EosQuiEt.java
5195	QuiaOfficiisQuo.java
5172	AccusamusDoloreQui.java
5032	InventoreUtQui.java
4974	AutNihilPossimus.java
4955	IustoSeateeOdit.java
4646	VoluptasSintVoluptatem.java
4914	AdAliasFugiat.java
4741	EaNecessitatibusconsequatur.java
4572	NatusQuiaMinus.java
4563	FugitVoluptatibusAut.java
4561	VoluptateErrorImpedit.java
4381	SitDolorumPorro.java
4367	AdipisciEtRem.java
4326	NecessitatibusMollitiaSuscipit.java
4087	DolorNamConsectetur.java
3683	QuamCupiditateDolor.java
3655	AutConsequaturNihil.java
3514	ExpeditaOfficiisOdio.java
3439	QuiaAssumendeNisi.java
3055	UtAutNeociunt.java
2799	AutemUtSunt.java
2584	CommodiOccaecatiMollitia.java
2353	IustoDictaAperiam.java
2272	QuibusamEtAd.java
2038	MagnamSaepeAut.java
1554	QuiSolutaCorporis.java
1486	UllamAperiamEst.java
1252	VoluptatibusConsequaturNeque.java
933	SuntQuiNatus.java
756	DolorumMolestiaeSint.java
672	AssumendaSaepeSimilique.java

実行可能コード行数 (ワースト50)

実行可能コード行数	ファイル名
0	VelNesciuntAut.java
0	OfficiisAutQui.java
0	AutEstQui.java
0	EosAutVoluptas.java
0	DoloreOmnisExpedita.java
0	NatusVitaeSunt.java
0	ModiDucimusQuod.java
0	ConsequunturQuodItaque.java
0	AliquidCumPerspiciatis.java
0	TemporeIustoLibero.java
0	DignissimosQuiEt.java
0	EnimVelEst.java
0	AccusantiumEnimAliquid.java
0	ConsequaturQuiRerum.java
0	OmnisFacereDolores.java
0	AtSoluitatt.java
0	IpsamTugiaeVelit.java
0	EvenietUtNon.java
0	HicErrorNulla.java
0	ConsequaturQuiaCupiditate.java
0	AVoluptatemFacilis.java
0	QuiaTEt.java
0	EtDoloriesperspiciatis.java
0	NumquamSequiAdipisci.java
0	InReiciendisIn.java
0	SapienteTemporaLabore.java
0	CorporisSEt.java
0	NullaDolorVoluptates.java
0	DoloresQuiUt.java
0	AliquidAbAccusamus.java
0	VeroIpsamSint.java
0	VoluptatibusSedEum.java
0	ModiSedOdio.java
0	FugitPorroAut.java
0	AnimiAutemVoluptatem.java
0	VeniamQuisquamIllum.java
0	EiusEtEum.java
0	LaborumFugiatSint.java
0	FugiatNonAtque.java
0	EvenietMolestiaeExpedita.java
0	RemMaioresUt.java
0	PariaturDolorSste.java
0	ExRepellendusQuam.java
0	FugaMaximeista.java
0	DoloremQueaQui.java
0	AutMolestiasSed.java
0	UndeEtEligendi.java
0	CupiditateCumUt.java
0	SapienteMaximeQod.java
0	UtRerumNisi.java

図 7-16 メトリクスランキング表示画面①

ファイルの複雑度の合計 (ワースト50)

複雑度	ファイル名
11964	QuamSitQuidem.java
11883	EtQuosLaborum.java
11667	IureInventoreIpsum.java
11407	DoloremqueSitQuam.java
10906	UtUtEsse.java
10897	PariaturFugiatSit.java
10656	ItaqueTotamRerum.java
10638	QuiaVoluptatemEt.java
10489	IpsaVelQui.java
10447	HarumNihilSimilique.java
10296	IsteUtQuia.java
10262	MollitiaEtDoloremque.java
10224	DelectusCulpaAperiam.java
9963	SintAbQuibusdam.java
9949	EtAssumendaSapiente.java
9275	AsperioresRepudiandaeTempore.java
9251	NemoNostrumPorro.java
9014	DucimusConsequaturNecessitatibus.java
8422	AutEtExercitationem.java
8283	NamConsecteturMinima.java
8155	VelitDistinctioUllam.java
8055	QuoDoloresQuidem.java
7958	VitaeConsequaturVoluptas.java
7494	DoloresAutRerum.java
7474	AliasEtSit.java
7457	CumqueLaudantiumAmet.java
7402	AsperioresExcepturiQui.java
7256	VoluptatesEnimHic.java
7253	ConsecteturNamRecusandae.java
6816	FacereQuiConsectetur.java
6810	EaEumQui.java
6348	EtAccusamusSit.java
6237	EtEstExcepturi.java
5710	EstAccusantiumQuam.java
5651	DolorumEtQuo.java
5030	NatusPerferendisIure.java
5001	NonVoluptatemQui.java
4868	NihilAccusantiumNumquam.java
4635	AdipisciConsequaturEa.java
4622	MolestiaeEligendiNon.java
4082	EtSitQuo.java
3730	EnimQuiaEc.java
3729	NamMolestiaeAperiam.java
3445	EtUtSed.java
3340	VoluptasOmnisFugit.java
2877	NullaEtEligendi.java
2522	AliquidAtqueConsequatur.java
2279	NecessitatibusDolorVoluptatum.java
2075	PerferendisEarumMinima.java
2065	EaDucimusTemporibus.java

メソッドの複雑度 (ワースト50)

複雑度	メソッド名
491	numquam
481	praesentiumEiusAspernatur
480	autemVelitEst
466	maioresEosVeritatis
459	doloremBeataeSunt
443	dolorIllo
432	nesciuntNatus
426	molestiasAutem
417	necessitatibusOfficiaAut
416	voluptates
414	providentDucimus
401	qui
389	consequatur
380	quiCum
343	sequiSintEos
337	utLaborum
303	aliquidEaque
276	estVel
275	natusOfficiis
273	nonNobis
270	nonIure
238	ut
224	voluptatemUndeVero
223	repellendus
221	hicNullaEt
211	nihilAspernaturMagni
208	officiaCorporisEt
207	ipsaQui
197	quiaEosDolor
183	estConsequuntur
181	necessitatibusEt
177	cupiditateVelit
171	quas
157	totam
157	nihilRerum
149	ea
145	temporaDolor
144	repellendusNemo
139	voluptateRecusandaeUllam
118	quiaIllo
111	perferendisSoluta
106	delenitiDeserunt
77	veniamIn
68	quoDoloresQuibusdam
63	quiAutQuia
52	ametImpedit
44	iureVoluptatem
31	saepeItaqueOptio
27	facilisAut
16	praesentium

開発者がコミットした行数 (ベスト50)

コミット行数	Autho ID
29672	ea2b17e2ffef5d844a82e4b92b42245f
28674	6314b1a726e4fceb0fe8587811be5aeb
28377	89447a44aafdd0e27dabf4b10bfae83f
28335	11369f2975e17c770df8eb320c8ec0a1
28218	2767c55da3788b9465d9b1aec15f7c63f
27396	a4177339d864ec60d758086dae90e482
27003	2d842a9b8e15a2d24a43c55cbbcc8ebc
26851	4393d84b53abcc1b8afc3402724aa1bd
25566	8849313be9501c28c3e8aad5b480ae
24237	342095ad99b3bb8db68cb01565f0c306
24170	5efbd6e5a6c3ceda12ff1bbca4a865a9
23371	b0fbf8821678630e4c490f270d93fa54
23023	8529255b6402948e98a8f7cf78606719
23005	c2b97c562551d3a2a21be0acda44443
22461	595591ef3f84b10d5150df1f81e6e35733
22370	227b878eecb5632ea50397248c395dfc
21568	997235d3631d6f2acbe930686f9489afa
20832	a7a2089e132db1ba0c8e6eb40d9d2f3a
20336	8bfbda1a2ca17db7853ca65230021d0e
20122	cdf6b83ca5def9d6fe449b6bfc2beda3
19853	51ef42dd6bcbadc6c20d0df82cc6a0c9
19011	c2ab87435c4a8e93671455a2427b6324
18560	f2fec8fe74d0bf8246f2484520cbadff
18501	9bbaf2b7fef3327c0984bcdee6efbf1d
18260	9439f10a3373ef43debce52dd302910d
18144	885530638ca38e99f0dfd9a1724c3078
17979	144b2d9f1ae22bdaee52ca2f51f64c30
16501	a5f28b6216372ad55c9619c637d745c3
16014	ecd6ccda766a064b59debfcd5a9907c9
14361	ac503871a7210216d203ec0fe6777537
14169	2ec90840a99cdff983f094a3a854d78e
12488	4e830cbf77be5716232f552c4be0ac20
12438	7f4ec61c2beb6b0914a11023f0e4f9ab
10750	bd5ccc6942aa643a42b49803846d1fb1
10225	1cc870e096ef2ce8f993d9929a70c28
9563	2496c17ea8b4cf9a8713f2558e09ce7b
8442	6718c42139801fa4d7102102f3b3d2d0
7628	119bbb5e90079a3ca5e9d5678a78fbd7
5347	cc48c0bbe0766dacf2c5fE522558bc9e
4760	bb6243233523ec9968f9801f0f77574c5
4417	be299dcced8757006f0f03fd50de7727
3198	31d5cded728157dff46cb1e645c6d369
2538	d65f9fbf01abdf47a36abb9dd08a2649
1716	a151bc281791b339b4bacaba30dd02a
1225	b1d271a2427ed8f849fd39a952cea32a
1200	b17fc2b98f10d84916746c899ee72b3d
491	b58a6e9926083a24f14ababff6a26af7
388	bf2fb47a145e1bf0ddfc5d3051c7b4ae
365	7123396ca9dca70508cc642de6e8c212
306	abfa7558dbcf895aee1f190b34291b81e

図 7-17 メトリクスランキング表示画面②

ファイルのバグ密度算出部分の開発

私は携わっていないものの既にバグ密度については R[23]を使いチームメンバーが行っていたので，その方法を共有して貰った．私たちは R を使い分析した知見をシステムに活かそうとしていたので，この部分で相違が存在すると意図した分析をシステムで行えないことが考えられた．

次に R を使い行われていたバグ密度の算出方法とそれに対応する私の実装について説明する．R を使ったバグ密度の算出では一度コミットログのコメントからチケット番号を抽出し，

そのコミットにて影響があったファイル全てにバグがあると仮定していた．またコミットログに記されているファイルとメトリクス分析結果と付き合わせる際にファイル名を正規化していた．そこで私は以下の方法でバグ密度を計算した．

1. コミットログのコメントからチケット番号を抽出
2. チケット番号が抽出できたコミットで変更があったファイルにチケット番号のバグがあったと仮定し，対応するファイルにチケット番号を複数付与
3. それぞれのファイル単位でチケット番号が何種類付いているか計算し，それがそのファイルに含まれていたバグ数だと仮定
4. 前で得たバグ数をファイルの実行可能行数で割りバグ密度を算出

実際の実装では処理時間の問題で何度か改修を行った．一番初めの実装では１を予めデータを読み込む際に行いテーブルに保存し，２～４を一つの SQL を使いウェブアプリケーションのコントローラで行っていた．しかしコントローラで行うと閲覧時に時間がかかってしまうので，次の実装ではキャッシュをテーブルに保存することでそれを避けた．しかし実際の導入では上手く動かず，原因を探ってみると恐らく SQLite で実行時間がかかりすぎてしまった為のタイムアウトだということが判明した．その後の改修で２～４を全てのファイルについて同時に行うのではなく一つ一つのファイルについて行うことでタイムアウトを防ぐことができた．

メトリクス時系列表示画面の実装

私は一部のメトリクスが時系列でどのように変化しているか表示する画面を作成し，プロジェクトにおける品質に関するメトリクスがどのように変化しているのか閲覧できる画面を開発した．この開発には 7.1.4 で行ったのと同じようにライブラリを探し使用した．ここではグラフの描画に Highcharts [24]とそれを Gem に纏めた LazyHighCharts [25]を使用している．ここで使用したメトリクスは以下のとおりである．

- ファイル数
- クラス数
- メソッド数
- 複雑度とバグ数
- 複雑度とコード行数

最後の２つに関しては双方のメトリクスにおいてどのような関連があるか判断するため，一つの表に２つのグラフを載せることとなった．図 7-18 にサンプルデータを用いて描画した画面をを示す．

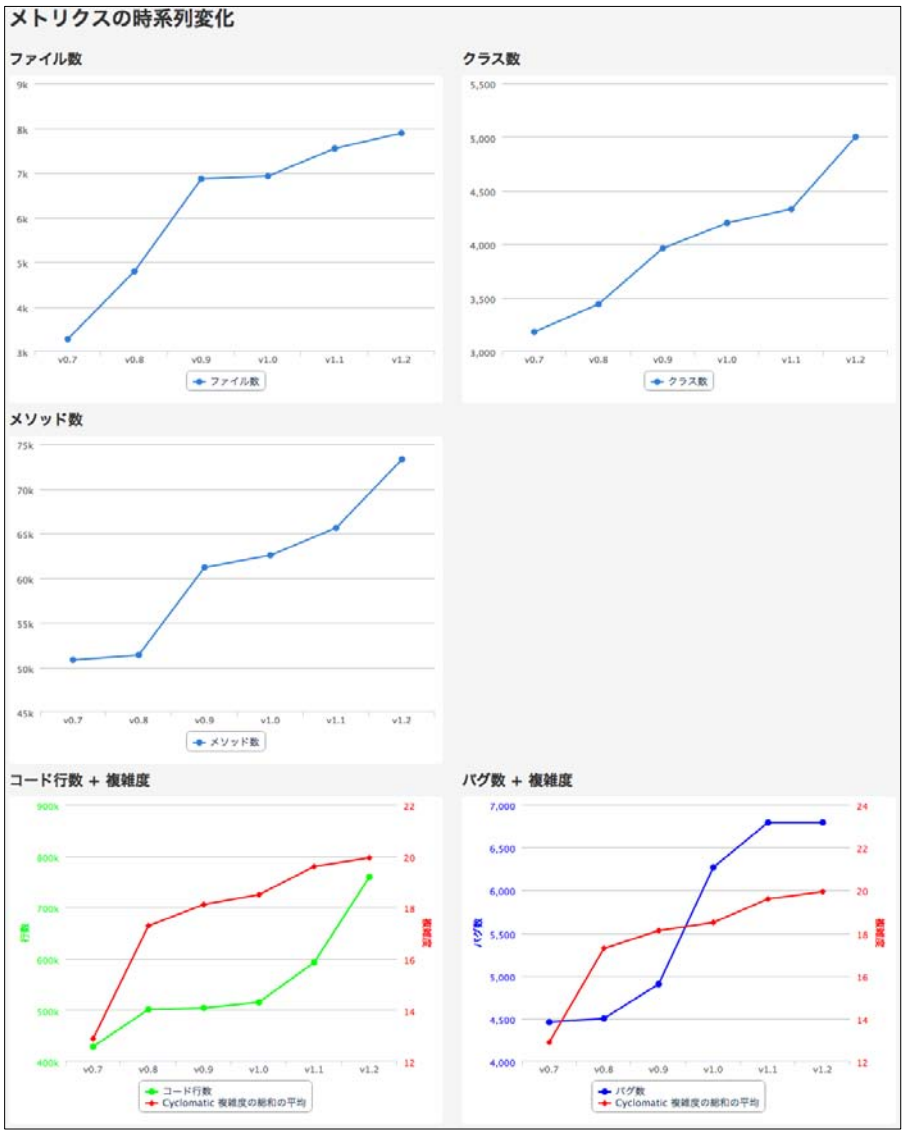


図 7-18 メトリクス時系列表示画面

取りまとめ

私がメトリクス表示画面の取りまとめを行い，一つのシステムとして結合した．ここで取りまとめたものは前述した２つ含め以下の４つである．最後の２つは私が担当したものである．

- プロジェクト全体のメトリクス表示画面
- ファイルごとのメトリクス表示画面
- メトリクスランキング表示画面
- メトリクス時系列表示画面

この作業に当たり 7.2.1 の教訓を活かし，試しにメトリクスを表示する画面を作成しこれをひな形としてチームメンバーに配布した．これによりチームメンバーはこれを改変，拡張

することで担当部分の実装ができるようになり，またこの後の結合が容易が容易となりプロジェクトを円滑に進めることができたと考える．

7.3.4 システムの導入

私はシステムの導入にあたり，対応や実際の導入を他のチームメンバーと共に主導的に行い，システムを実際に企業で使用する際の問題の解決に貢献した．

導入手順書の作成

キンの記した導入手順書のひな形を元にし，私は再度実際に動作確認環境へシステムを導入しながらその手順を細かく記し導入手順書を作成した（表 7.5）．これを日立に送付し導入の準備をして頂くことが出来た．また導入手順書の作成により今後システムのソースコードを一般に公開する際に必要となるドキュメントを作成することができた．

また私は日立に赴く前に先方から事前にできる作業をして頂けるということを伺っていたので，そのために動作確認環境へ改めてシステムを導入しその手順を書き記すと共にそれを導入手順書としてまとめあげた．以下がその目次である．

表 7.5 導入手順書の当初における目次

目次

システムの導入

私とチームメンバーである紀が日立に伺い社員の方と共にシステムの導入を行った．実際の導入の際にはいくつか問題が発生したので，それに伴ってシステムおよび導入手順書の修

正を行った．本システムの導入は日立に伺い実際にシステムを動作させる仮想マシンへ SSH で接続し作業を行った．実際にブラウザを立ち上げシステムへ接続することも可能であり，逐次ブラウザを立ち上げ実際の動作を確認しながら導入を行った．

その際に発生した問題は多きく２つ存在し，１つは Gem のインストール，もう１つはメール送信である．前者については社内から外部へアクセスするには設定が必要だったためであり，そのための設定を行うことで正常に動作するようになった．後者についてはメール送信のための送信用アカウントの用意が出来ていないということだった．この時点のシステムはメール送信に失敗してしまうとその時点で動作を停止してしまうので，その部分についてはその場では一旦処理を行わないようにし動作を確認することができた．

導入手順書の修正

前項で行ったシステム導入にて発覚した導入手順書の不足の修正，またシステムへのパッチのあて方を追記し，実際にシステムを運用するためのドキュメント作成に貢献した．導入手順書を作成した段階では Ubuntu によりインストール作業を行っていた．しかし実際の環境では CentOS であったので，インストールの際のパスや関連ライブラリのインストール方法が異なっており，その部分を修正した．また導入時にいくつかの問題が発生したので，今後システムを修正する必要が出てきた．そこでその際に日立に修正して頂くことになったので，システムへのパッチのあて方について章を追加した．

7.3.5 開発構想書の一部作成

私は開発構想書の二章である想定使用者の章を担当し，システムが対象とする利用者を明確にして要件を決定することに貢献した．私はどのような人を対象に開発を行うかということを定めると共に，その使用者がどのような課題を抱えているか分析を行い，開発構想書に記した．

第8章 おわりに

本研究開発では株式会社日立製作所と連携し，ソフトウェアの開発過程で蓄積されるデータを用いて品質低下の兆候を警告すると共に品質に関連する指標を提供することで，品質向上の一助となりうるシステムの研究開発を行った．

著者は主にシステムを実際の企業内で使用するに必要となるユーザ権限管理機能やプロジェクト管理機能の設計と開発に加え，システムに不可欠となるデータ読み込み蓄積機能の設計開発を主導した．前者では案を作成すると共に日立からヒアリングを行い，実際に企業内で使われるためのシステムの基盤作りを行った．後者では今後の拡張や複数人での開発を見越した設計を行った．またシステム設計および開発の主責任者として，開発基盤や開発環境の構築やその支援，そして実際の開発においてもメンバーのサポートにあたった．プロジェクトの終盤では実際に日立に試用して頂くために，日立に赴き導入を行うと共に，そのための導入手順書を作成した．

プロジェクトの中で学び試行錯誤することも多く，これからはメンバーとそれぞれの担当範囲についてより共有し，意見を出し合うことが必要だったと考える．

謝辞

株式会社日立製作所の居駒幹夫様，土田正士様，河合亮様，河野哲也様，白井明様，中村宇佑様にはプロジェクトの当初から最後まで何回にも渡るミーティングをさせて頂き，ミーティングの中で，またメールにて多くのご助言やご支援を頂きましたこと，深く感謝致します．お力添えがあってこそ研究開発を進めていくことができました．

また本研究開発を含め入学当初より要所要所でご指導，ご鞭撻を頂いた指導教員の田中二郎教授には大変お世話になりました．研究開発ひいては学生生活を健康に過ごせたのもお力添えがあってのことだと存じます．

また課題担当教員の早瀬康裕助教，天笠俊之准教授にはお忙しい中でも多くのご助言，ご指導を頂きましたこと，心より感謝致します．

そして最後に，共にプロジェクトを遂行した靳松氏，宮永竜樹氏，紀冠男氏には研究開発においてとても支えられてきました．この度は本当にありがとうございます．

参考文献

[1] David Rice. Geekonomics: The real cost of insecure software. Pearson Education, 2007.

[2] Capers Jones (著), 富野壽 (監訳), and 小坂恭一 (監訳). ソフトウェア開発の定量化手法 第 3 版 ―生産性と品質の向上をめざして―. 共立, 2010.

[3] Craig Larman, and Victor R. Basili. Iterative and incremental developments. a brief history." Computer 36.6 (2003): 47-56.

[4] 野中誠, 小池利和, 小室睦. データ指向のソフトウェア品質マネジメント―メトリクス分析による「事実にもとづく管理」の実践. 日科技連, 2012.

[5] Capers Jones (著), Olivier Bonsignour (著), and 小坂恭一 (翻訳). ソフトウェア品質の経済的側面. 共立, 2013.

[6] 村尾憲治, 肥後芳樹, and 井上克郎. "ソフトウェアメトリックス値の変遷に基づいた注力すべきモジュールを特定する手法の提案" 電子情報通信学会論文誌. D, 情報・システム 91.12 (2008): 2915-2925.

[7] 畑秀明, 水野修, and 菊野亨. "不具合予測に関するメトリクスについての研究論文の系統的レビュー" コンピュータソフトウェア 29.1 (2012): 106-117.

[8] 山田茂. ソフトウェア信頼性の基礎: モデリングアプローチ. 共立出版, 2011.

[9] Subhas C. Misra, and Virendra C. Bhavsar. "Relationships between selected software measures and latent bug-density: Guidelines for improving quality." Computational Science and Its Applications—ICCSA 2003. Springer Berlin Heidelberg (2003): 724-732.

[10] "SonarSource S.A. SonarQube™" http://www.sonarqube.org/

[11] "Redmine" http://www.redmine.org/

[12] "Git" http://git-scm.com/

[13] 城川俊一. "知の創造プロセスと SECI モデル ―オープン・イノベーションによる知識創造の視点から―" 東洋大学「経済論集」 33.2 (2008): 27-37.

[14] "RubyGems.org" http://rubygems.org/

[15] "plataformatec/devise" https://github.com/plataformatec/devise

[16] "ryanb/cancan" https://github.com/ryanb/cancan

[17] "The Ruby Toolbox" https://www.ruby-toolbox.com/

[18] "Scientific Toolworks, Inc. Understand Your Code" http://www.scitools.com/

[19] " The Apache Software Foundation. Apache Subversion " http://subversion.apache.org/

[20] "StatSVN" http://sourceforge.net/projects/statsvn/

[21] Paolo Perrotta (著), and 角征典 (翻訳). メタプログラミング Ruby. アスキー・メディアワークス, 2010.

[22] "class Rake::Task" http://docs.ruby-lang.org/ja/2.1.0/class/Rake=3a=3aTask.html

[23] "The R Project for Statistical Computing" http://www.r-project.org/

[24] "Highsoft AS. Highcharts" http://www.highcharts.com/

[25] "michelson/lazy_high_charts" https://github.com/michelson/lazy_high_charts