独立行政法人情報処理推進機構 委託

2012 年度ソフトウェア工学分野の先導的研究支援事業
「ソフトウェア品質の第三者評価のための基盤技術
－ソフトウェアプロジェクトトモグラフィの開発－」
成果報告書

平成 25 年 1 月

国立大学法人奈良先端科学技術大学院大学

本報告書は独立行政法人情報処理推進機構　技術本部　ソフトウェア・エンジニアリング・センターが実施した「2012 年度ソフトウェア工学分野の先導的研究支援事業の公募による採択を受けて奈良先端科学技術大学院大学情報科学研究科（研究責任者 松本健一）が実施した研究の成果をとりまとえたものである.

目次

研究成果概要

１．研究の概要

本委託研究は，ソフトウェア品質の第三者評価の普及と高度化を推進する基盤として，

① 品質評価に必要となるソフトウェアプロジェクトデータの提供

② 提供されたデータに基づくプロジェクト理解

を容易にする技術の確立を目指すものである．そのために，２つの研究目標，

① ソフトウェアプロジェクトトモグラフィ技術の確立

② クラウド型開発管理環境の構築・評価

を設定し,その妥当性･有用性をプロトタイプシステムの実装と実証実験を通じて示した.

２．ソフトウェアプロジェクトトモグラフィ技術の確立

2.1 スナップショットの実現

本委託研究では，医療におけるコンピュータ断層撮影，いわゆる，CT（Computed Tomography）を，ソフトウェア品質の第三者評価に適したデータ構造を考える上でのモデルとする．すなわち，ソフトウェア開発プロジェクトをからだに見立て，プロジェクト開始から終了まで（あるいは，現時点まで）のいくつかの時点において，プロジェクトの状況を定量的に表すスナップショットを断面画像のように作成する．

CT において断面画像の系列から身体の 3 次元モデルが再構成できるように，スナップショットの数が十分であれば，その系列によってプロジェクトを様々な観点で可視化し，ソフトウェアやその品質が実現される過程(プロセス)を表すことも可能になると考える．スナップショットの系列によってソフトウェア開発プロジェクトを表現する手法を「ソフトウェアプロジェクトトモグラフィ（Software Project Tomography）」と命名し，解析や可視化に適したスナップショットの構造や構成要素など,基本概念を構築した(図 1 参照).

スナップショットは，ソフトウェアやその品質が実現される過程を表現（再現）するための多様な解析や可視化に向けて，必要なソフトウェアプロジェクトデータを扱いやすい形式で提供するプラットフォームである．これにより，本委託研究が目指す「① 品質評価

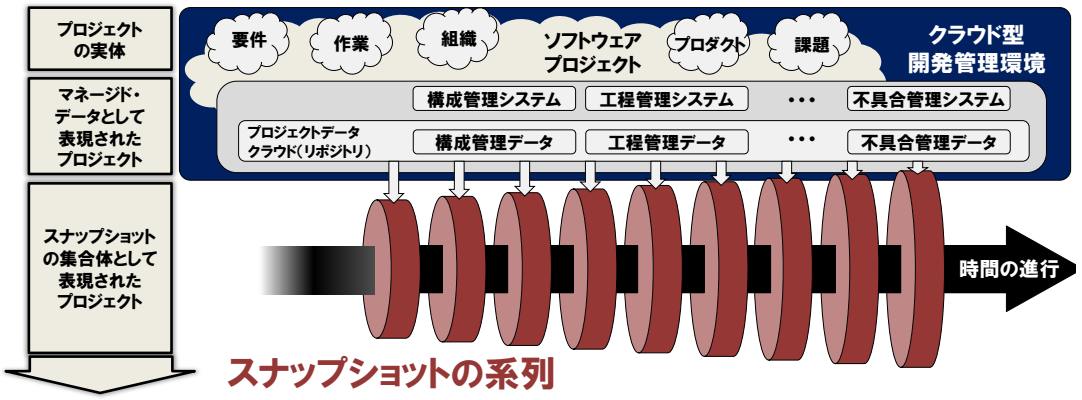


図１　ソフトウェアプロジェクトトモグラフィの基本概念

に必要となるソフトウェアプロジェクトデータの提供」が可能になると考える．なお，スナップショット自体も可視化の対象であり，多様なソフトウェアプロジェクトデータやその解析結果（一次解析結果）の集合体となる．本委託研究では，次の 5 つの観点のペインで実現した．

- 要件ペイン：プロジェクトで実現すべきソフトウェア要件（機能／非機能）の全体像，要件間の関係，各要件の実現状況などを表す．
- 作業ペイン：プロジェクト要件を実現するために実施される作業の全体像，作業間の関係，各作業の進捗状況などを表す．
- 組織ペイン：作業を実施する組織の全体像，組織構成メンバー間の関係，各メンバーの状況などを表す．
- プロダクトペイン：作業によって作成･利用されるソースコードや中間成果物といったプロダクトの全体像，各プロダクトの構成要素と要素間の関係，各構成要素の作成・利用状況などを表す．
- 課題ペイン：ソフトウェア要件が実現されていく過程で発生する不具合や仕様変更要求といった（プロジェクト終了までに解決すべき開発上の）課題の全体像，課題間の関係，各課題への対応状況などを表す．

2.2 スナップショット生成・解析・可視化システムの試作

「ソフトウェアプロジェクトトモグラフィ」の基本概念と「スナップショット」の定義に基づいて，スナップショットの生成，および，スナップショットに基づく「ソフトウェア開発プロジェクトの解析と可視化」の具体的な方式を設計し，システムを試作した．設計の概要を図 2 に示す．個別の方式の概要は表 1 の通りである．

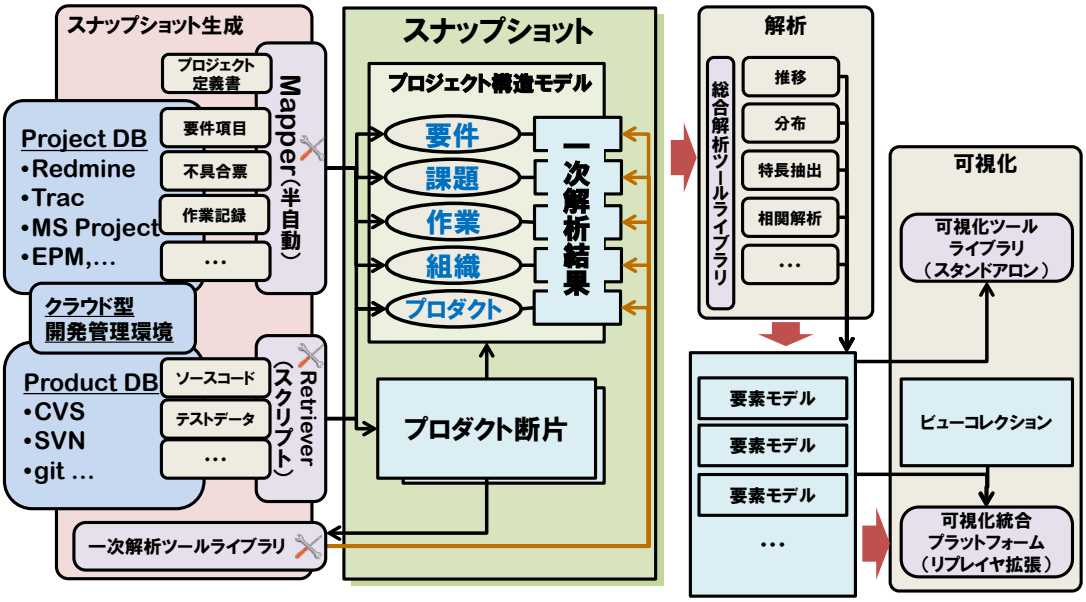


図 2　スナップショット生成・解析・可視化の設計概要

表１　スナップショット生成・解析・可視化方式の概要一覧

スナップショットの生成	
要件ペイン	FP（Request For Comment）から作成した．非機能要件評価表を作成し，RFP上の非機能要件の記述の網羅性や明確さを一次解析データとして付加した．また，非機能要件にマッピングされるキーワード間の関係を明らかして，その評価や改善を支援するため，自己組織化マップを付加した．
作業ペイン	開発作業におけるアプリケーションの実行履歴を記録するツールTaskPitを用いて作成した．
組織ペイン	本委託研究メンバーが開発してきた「アウェアネス支援システムACTION」，および，ジョージア工科大学が開発した「ソーシャルネットワーク可視化ツールVizter」を用いて作成した．
プロダクトペイン	構成管理システムが蓄積するソースコードから作成した．UCI Source Code Data Setsに蓄積されている12,191プロジェクトのデータに基づき，45個のソースコードメトリクスそれぞれについて基準値を作成し，付加した．
課題ペイン	課題管理システムが管理する課題表に基づき作成した．一次解析データとして，未対応課題数，および，未対応課題が残存するソースコードとの紐付けを行った．
スナップショットに基づく解析	
検証型解析	静的解析ツール「Understand」を用いて，スナップショットに含まれるソースコードの特性値，および，ソースコード間の依存関係を計測し，未対応課題数を多く含むスナップショットにおいて低品質なモジュールが生成された要因を解析する技術を開発した．
探索型解析	低品質モジュールをソフトウェア開発プロジェクトの初期に予測（判別）する技術を開発した．
スナップショットに基づく可視化	
検証型可視化	静的解析ツールUnderstandを用いて，スナップショットに含まれるソースコードの特性値，および，ソースコード間の依存関係を計測し，未対応課題数を多く含むスナップショットにおいて低品質なモジュールが生成された要因を解析する技術を開発した．
探索型解析	階層的クラスタ分析ツールHCE（Hierarchical Clustering Explorer）を用いて，Rank-by-feature frameworkに基づく可視化を実現した．

2.3 「ソフトウェア品質の第三者評価」の妥当性評価

「2.2 スナップショット生成・解析・可視化システムの試作」で示した方式や試作システムをオープンソースソフトウェア等の開発データに適用し，それら方式やシステムが，ソフトウェア品質の第三者評価を容易にすることを確認した．ここでは，特に，第三者評価で必要となる「プロジェクト理解」を容易にする要素技術として，

- プロジェクト全体の俯瞰
- プロジェクト構成要素（間）の傾向や関係性の顕在化・評価
- 仮説の生成・検証

が必要と考えた．これら 3 つの要素技術，および，スナップショットの 5 つのペインと，個別の評価の対応関係を図 3 にまとめる．更に，「プロジェクト構成要素（間）の傾向や関係性の顕在化・評価」と「仮説の生成・検証」に属する評価が，「ソフトウェア品質の第三者評価における評価対象」のいずれに対するものであるかを図 4 に示す．ここで，「ソフトウェア品質の第三者評価における評価対象」とは，文献[IPA]に示されている「第三者評価に求められる 3 つのスコープ」とソフトウェア品質そのものである．なお，「プロジェクト構成要素（間）の傾向や関係性の顕在化・評価」については，評価対象プロジェクト内での相対的な評価となる内的妥当性（Internal Validity）と，ベンチマーキングなどの標準値・基準値に基づく評価となる外的妥当性（External Validity）に大別されている．以下では，個別の評価事例のいくつかについて，その概要を示す．

図 3　「スナップショットのペイン」と「評価事例」の対応関係

図 4　「第三者評価のスコープ」と「評価事例」の対応関係

① コミュニケーション分析に基づく組織構造グラフ（図 5）

対象プロジェクトの全てのスナップショットに含まれる情報に基づく俯瞰の例である．オープンソースソフトウェアの Apache プロジェクトにおける開発者とユーザ間のコミュニケーションの様子を，1 ヶ月毎に示している．

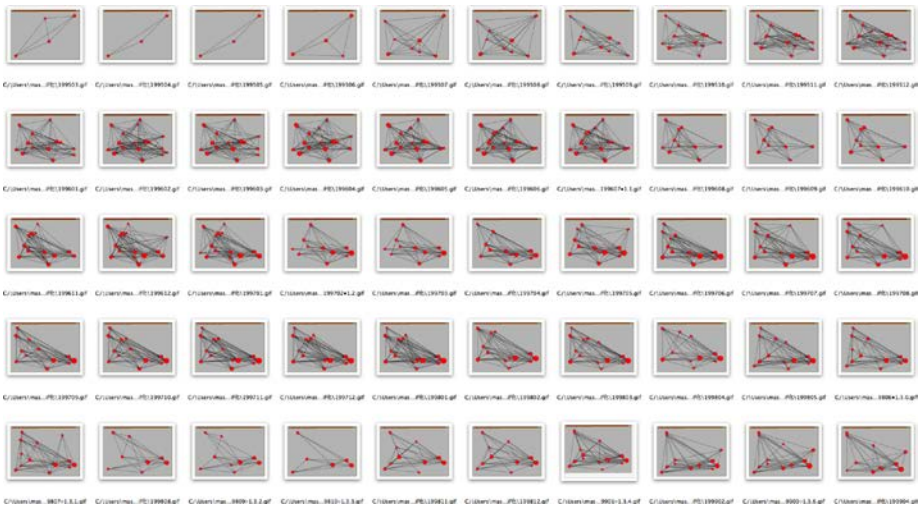


図 5　コミュニケーション分析に基づく組織構造グラフ

② コミュニケーション中心性メトリクス（図 6）

ソフトウェア品質の第三者評価のスコープの一つ「従事者の妥当性」の評価の例である．プロジェクトにおける組織構造の時間変化を図 5 で俯瞰したが，俯瞰だけで対象を理解することは難しい．そこで，組織構造の変化を定量的に表す次の２つのメトリクスを定義した．

次数中心性メトリクス：組織内コミュニケーションの活発さを表す．

媒介中心性メトリクス：メンバーが扱う情報・リソース量を表す．

２つのメトリクス値の時間変化を図 6 に示す．これにより，組織内でのコミュニケーションが活発になると，コーディネータの負担がそれに比例して増大するが，ユーザと開発者の負担は大きく変化しないことがわかる．コーディネータは，開発者とユーザの媒介者として，ユーザと開発者のコミュニケーション負荷の軽減という本来の役割を十分に果たしていることがわかる．

③ 低品質モジュール予測（図 7）

ソフトウェア品質の内的妥当性評価の例である．開発終了時に欠陥（バグ）を含むことになるソースコードモジュール（低品質モジュール）を事前に知ることは，開発管理上だけでなく，品質の第三者評価においても極めて有用である．Eclipse プロジェクトの開発データで予測モデルを構築したところ，早期（1 年プロジェクトにおける開発終了 9 カ月前）に低品質モジュールの予測を行っても，開発終了時と同等の精度（AUC と F 値のいずれにおいても）が期待できることがわかった．

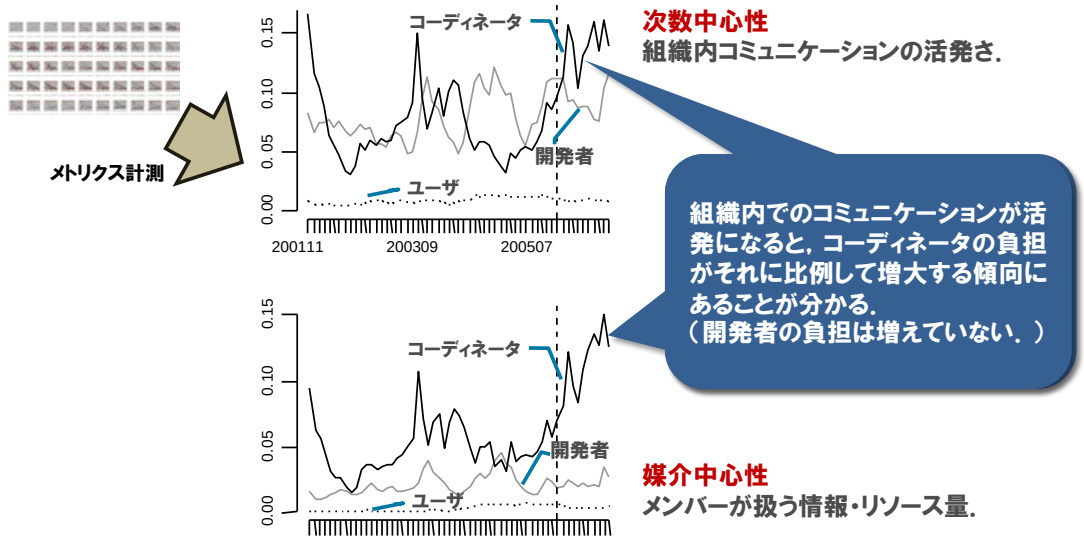


図 6　コミュニケーション中心性メトリクスによる評価

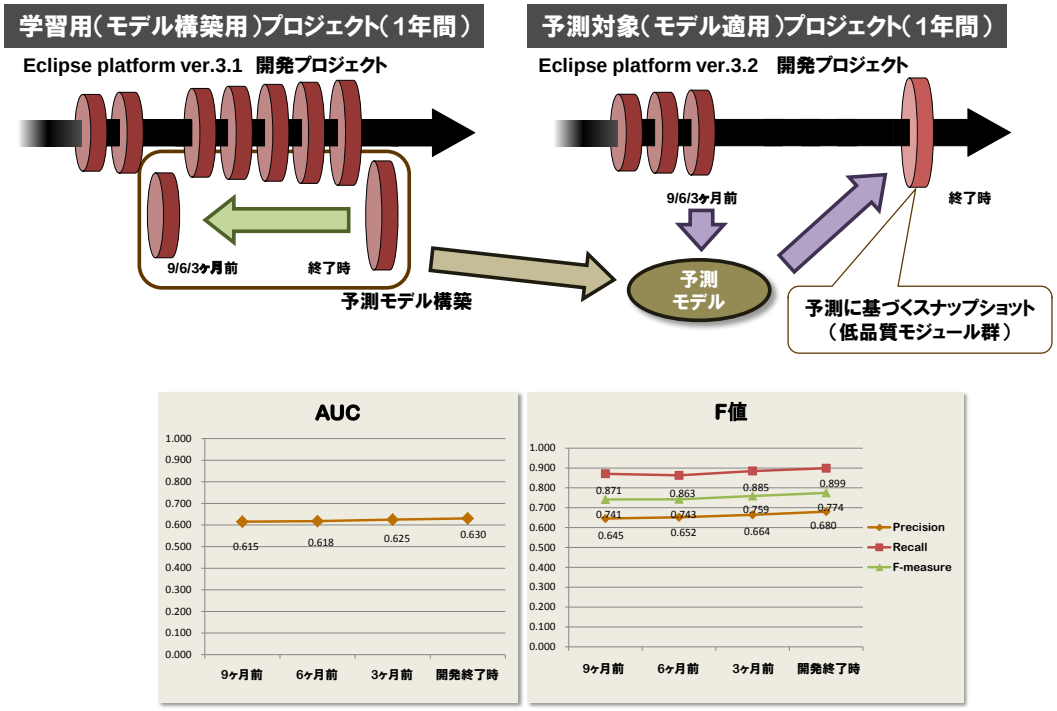


図 7　低品質モジュール予測

④ 非機能要件特性レーダーチャート（図 8）

ソフトウェア品質の外的妥当性評価の例である．大学，病院，官公庁，地方自治体，独立行政法人などが，ベンダ候補企業向けの入札情報として WWW 上で公開している 29 件を対象に，IPA「情報システム調達のための技術参照モデル（TRM）」他を参考に作成した非機能要件評価表を適用した．総合評価トップ 3 の各特性の平均評価点を基準値として，レーダーチャートを作成した．RFP 上で改善が必要な非機能要件特性がはっきりとわかる．

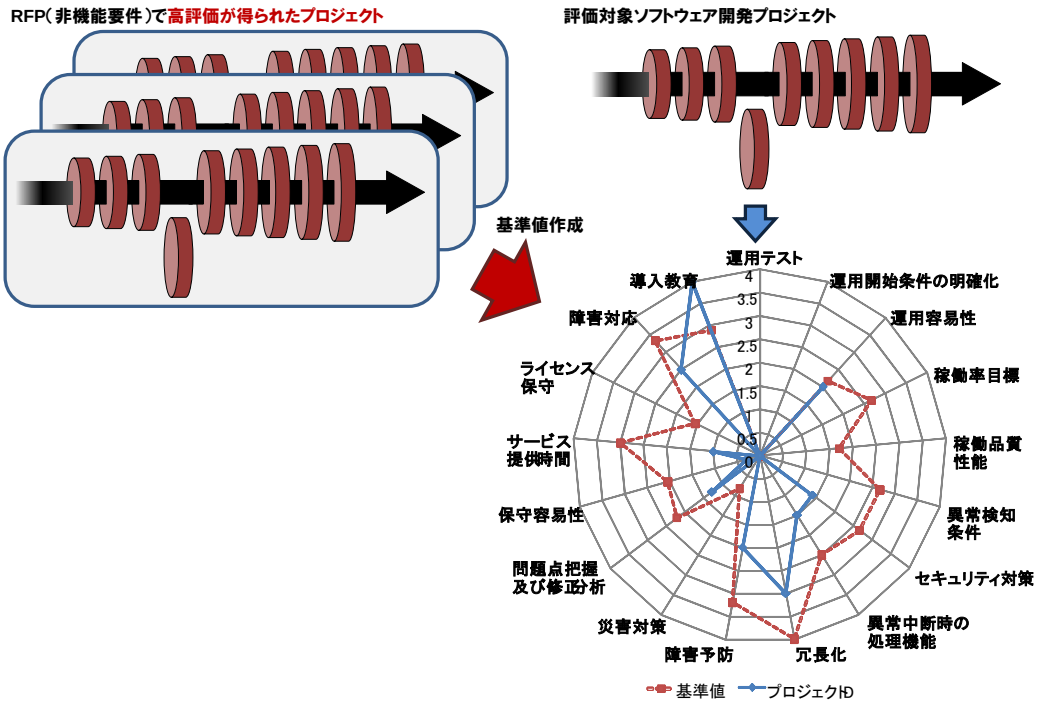


図 8　非機能要件特性レーダーチャート

⑤ ソースコードメトリクスの基準値作成（図 9）

ソフトウェア品質の外的妥当性評価の例である．UCI Source Code Data Sets[UCI]として提供されている 12,191 プロジェクトのデータに基づき，45 個のメトリクスの基準値を作成した．基準値の例としては，箱ひげ図などでもよく用いられている

上限値：第三四分位＋1.5×（第三四分位－第一四分位）

下限値：第一四分位＋1.5×（第三四分位－第一四分位）

がある．

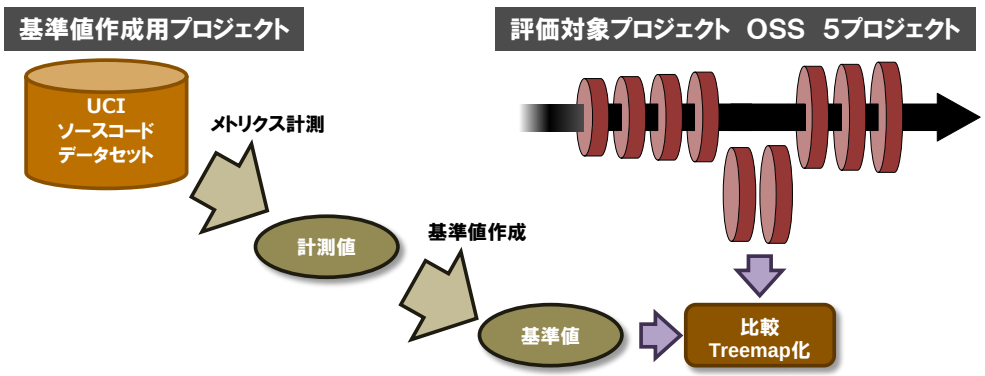


図 9　ソースコードメトリクスの基準値作成

⑥ 階層的可視化分析ツール HCE を用いた仮説生成（図 10）

階層的可視化分析ツール HCE（Hierarchical Clustering Explorer）は，入力されたスナップショットから，類似データをグルーピングする階層的クラスタリング機能，注目するクラスタに含まれる任意の 2 変数間の関係を把握するための散布図作成機能，クラスタ内のデータ分布を把握するヒストグラム作成機能などから構成されている．Eclipse Platform の不具合報告データ約 7000 件，11 項目（変数）を HCE への入力とし，オープンソースソフトウェア開発の知識を持つ被験者 3 名，持たない被験者 3 名がそれぞれ生成した仮説の量と質を比較した．なお，主要学術論文誌・国際会議で既出の法則・関係性に合致する仮説を有用とした，持ち時間 1 時間で被験者が生成した仮説は，

知識なし被験者：計 8 件．うち，有用な仮説 2 件．

知識あり被験者：計 18 件．うち，有用な仮説 4 件．

であった．生成された有用な仮説の例としては，

「課題が報告されてから担当者に割り当てられるまでの時間」と「課題が割り当てられてから解決されるまでの時間」は反比例する．

がある．対象ソフトウェアの知識が十分でない場合でも，有用な仮説を生成することが可能であることが確認された．

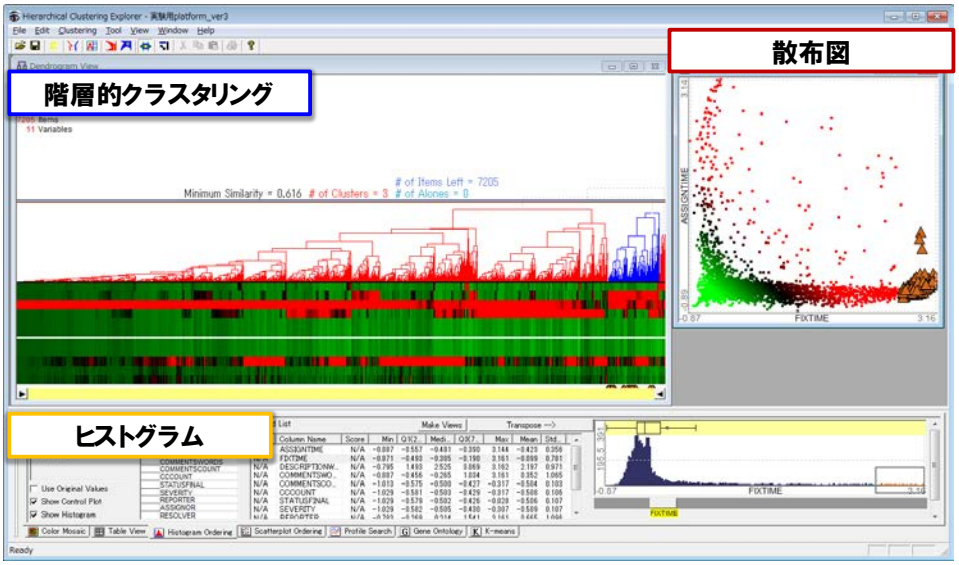


図 10　階層的可視化分析ツール HCE

2.4 まとめ

ソフトウェアプロジェクトトモグラフィ技術に関する成果は次の通りである．

① スナップショットの実現

ソフトウェアプロジェクトを表現するに十分な情報を含むスナップショットが実現可能であることを示した．

② スナップショット生成・解析・可視化システムの試作

ソフトウェアプロジェクトを５つの観点（スナップショット上の５つのペインに対応）それぞれで，「俯瞰」，「傾向・関係性の顕在化・評価」，「仮説の生成・検証」が可能であることを示した．

③ 「ソフトウェア品質の第三者評価」の妥当性評価
スナップショットに基づく解析と可視化によって，ソフトウェア品質の第三者評価の主要三対象，および，ソフトウェア品質そのものの評価が容易になることを，実験・ケーススタディにより示した．

３．クラウド型開発管理環境の構築・評価

3.1 クラウド型開発管理環境の構成・運用方式の評価

(1) 新しい統合型ソフトウェア開発管理環境の組み立てと実プロジェクトへの適用

今日，ソフトウェア開発プロジェクトにおける構成管理，不具合管理，工程管理などは自組織内で行われる「オンプレミス（on premise）型開発管理」が主流で，ソフトウェアプロジェクトデータは自組織内で収集・蓄積されている．しかし，ソフトウェアプロジェクトデータをソフトウェア品質の第三者評価に利用することを前提とするならば，データの収集や蓄積を自組織に閉じない形態で行うことも検討すべきである．本委託研究では，実際にクラウド型の統合開発管理環境を実現することで，その構成方式，運用方式の利点を実証し，課題を抽出した．

クラウド型の統合的なソフトウェア開発管理環境の構成像を図 11 に示す．その特徴は，産業界で一般的なビジネス・システム，企業のいわゆる業務情報システムと対比して考えると明らかである．産業界の業務システムでは，その事業の中のあらゆる計測可能なイベント，データがコンピュータ・システムに捕捉され，その中に格納・整理され，通信ネットワークによって運ばれて業務が進められる．いわゆる，「人，物，金」に関する状態と動きは隅々まで把握されている．本委託研究で構築した環境は，一般的なビジネス・システムと比較すると，その枠組みには何ら相違がない．ソフト開発プロセスを捕捉するイベントは個々のチケット，あるいはそれよりもさらに細かいチケットのステータスであり，プロダクトに関してはソースコードなど生産物の一行一行，あるいはコミットごとの契機を捉えている．これらの粒度，分解能はビジネス・システムにおける各種の伝票処理，あるいは物理的な生産物の流れ，サービスの提供，そして金銭の支払いや受領の契機と比べてなんら遜色がない．ソフトウェア開発に関する近年の進化した技術の集積によりこうした環境を商用のクラウド環境の中に構築し，実際の開発プロジェクトに供して評価することができた意義は，ソフトウェア開発環境の一つの姿を具体化したものとして大きい．

(2) 統合環境の機能評価に関するケーススタディ

統合環境を，すでにクラウド形態で商用提供されている課題管理システム Redmine と構成管理システム Subversion のサービスに，IPA/SEC 提供の定量的プロジェクト管理システム（EPM-X）を組み合わせルことで実現し，実開発プロジェクトに適用した．

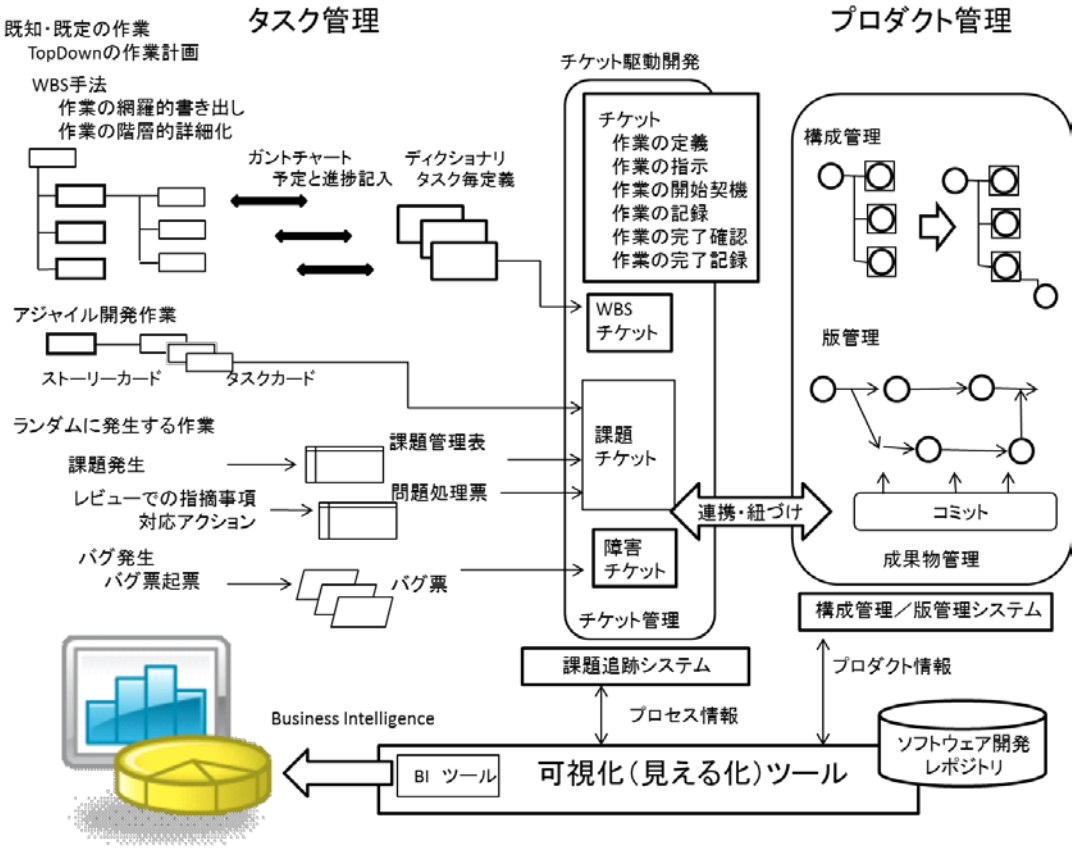


図 11　クラウド型の統合的なソフトウェア開発管理環境の構成像

EPM-X の標準サンプル・キットでは，想定するチケット構造と想定する情報項目が記入されれば，多彩な可視化情報をグラフで表示することができる．具体的には，WBS 管理，品質管理，障害管理，課題管理，要員負荷管理の 5 つの管理系で 14 種のグラフを表示できる．また，WBS，障害，課題の 3 種のチケットに記入された情報を組み合わせたグラフ，チケット記載情報と版管理システムからのソースコード行数の情報に工数の情報を組み合わせたグラフも表示可能である．

一方，本環境を適用したプロジェクトの開発者や管理者が興味を持つ情報は次の通りで有り，EPM-X の標準サンプル・キットが提供する情報とは必ずしも一致していない（図 12 参照）．

- タスクの開始・終了予定
- 進行中のタスクのステータス（例：担当者によるコード修正終了．管理者によるタスククローズ）
- 障害チケット発行時の障害状況，障害タスク遂行中のタスク状況
- 課題タスク発行時の課題の状況，課題タスク遂行中のタスク状況

以上の通り，クラウド環境上に実現された統合的なソフトウェア開発管理環境は，開発管理の枠組みとしては関係者の物理的な所在を越えて充分に機能した．しかし，実際のソ

フトウェア開発現場に対しては，サンプル・キットのままでは想定した機能は発揮されないこともわかった．

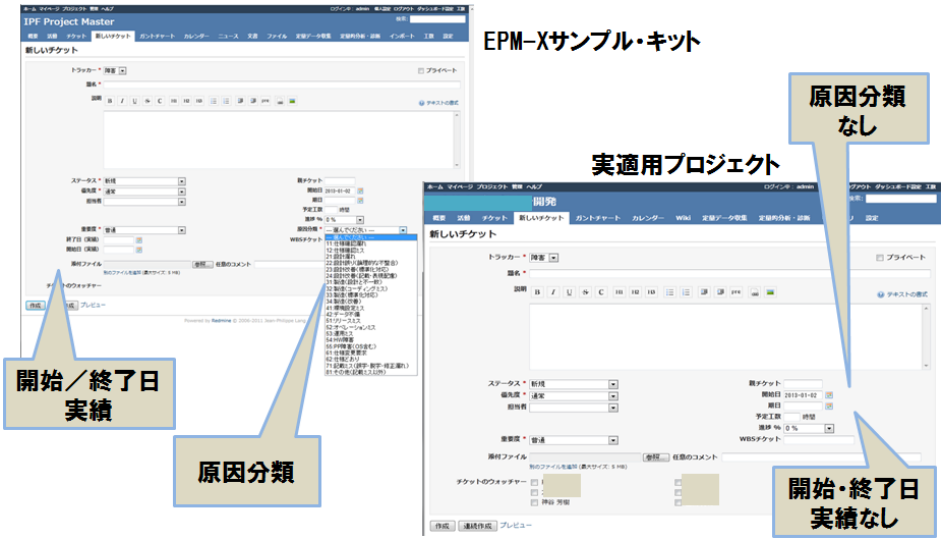


図 12　チケットの比較例

(3) 統合開発管理環境の活用法に関するケーススタディ

1 ケーススタディではあるが，実現したクラウド型開発管理環境を実プロジェクトに適用することで，その機能を想定通り発揮させるためには，個別のプロジェクト・マネジメント方式を反映したチケット構造に呼応した計測・可視化機能の設定が必要なことが明らかとなった．プロジェクト・マネジメント方式の個別性は高く，これまでのソフトウェア・エンジニアリングの成果を集積し，環境導入の容易化を図った標準サンプルのインストール・キットは今回の開発現場には受け入れられなかった．ケーススタディから得られた主な結論は次の通りである．

- チケット駆動開発，版管理・構成管理システムによるプロダクト管理，プロジェクトの自動計測と可視化機構を組み合わせた統合型開発管理システムでは，チケット構造，記入項目と計測・可視化機構を整合させた設定が必要である．
- チケットを活用したプロジェクト・マネジメントには「作業受け渡し型」と，「作業属性分類・記録型」の2つの方式がある．またマネジメントの流儀として，「WBS方式」と「フィーチャー方式」がある．統合開発管理システムの利用ではチケット構造の設定と計測・可視化機構の設定にあたって，これらの考え方を反映しておく必要がある．

3.2 組織間協業の実現可能性評価

(1) 地域コミュニティ企業群へのヒアリングの成果

組織間協業に関するクラウド環境の可能性に関しては,図 11 に示した環境の適用状況を素材として提示しながら地域企業群へのヒアリング中心の活動となった．ヒアリングは現場技術者のほかに，中堅マネージャ，管理者，経営者を含んで実施できた．結果として期待したほどネットワークを活用したサービスイメージを深めることはできなかった．

明らかになったのは,島根県を中心としたソフトウェア企業の Ruby 活用グループは非常に勉強熱心で,かつ新しいことへのチャレンジ精神に富んでいる．これを応援する人々（企業）もいる．こうしたコミュニティの当面の目標は，情報交換や切磋琢磨から地域としての技術力を高め，ブランド的に磨いて，受注につなげてゆく，ということである．

利害関係者への説明性と言うことでは，やはり得意先，受発注関係者への説明と言うことが考えられる．利害関係者が地域的に広がっている例もある．地域でも中国地方全域に広がり，また東京などからの仕事もある．ソフトフェア係争のようなことはない．組み込みソフトウェアの一部にコンシューマ製品への組み込みがあるが，これに関連したソフトウェア係争は経験がない.こうした状況で,本委託研究で提唱するようなクラウド環境は,当面は地域企業群の技術力向上に貢献し，受注増につなげてゆくのが実際的であるということが明らかになった．

(2) オープンソース連携製品の維持管理の課題

本委託研究のクラウド環境の活用に関する検討の中で一つの課題が明らかになった．今回の適用システムの開発対象ソフトウェアは,社内利用ソフトウェアの受託開発であるが,それはオープンソース製品と連携して動作し，かつ内部に別のオープンソースとして提供されている製品を取り込む方式を採用している．このように，今日の企業ソフトウェアには多様な形態でオープンソースの製品が多く含まれるようになり，オープンソースなしに企業システムの構築は考えられなくなった．企業システムに含まれるオープンソース製品の形態は様々である．コピーしてきてそのまま独自の版管理となってゆくもの，常にオープンソースの最新製品を取り込むようになっているもの，他の環境にあるオープンソース製品をサービスとして使いにゆくもの，など様々である．そしてその維持管理が大きな課題となる．

この課題に対してクラウド型のソフトウェア開発管理環境の活用が考えられる．現在,版管理システムの Git を核としたソフトウェア開発者用の共用ネットワークサービス GitHub が提供されている．提供元（GitHub 社）の本拠地は米国であるがサービスは全世界で広く利用されている．このサービスはクラウド型の Git 搭載サーバを中心に，オープンソース製品を様々な開発グループが成長させてゆく開発と提供のためのコミュニティ基盤を実現している.開発者は関心をもったオープンソース製品を GitHub センタから環境毎自分の環境にコピーし，適宜改良，追加開発を行ったのち，一定のルールでもとの GitHub センターに戻し，追加・改良部分をマージして，皆に提供することができる．

本委託研究では，こうしたオープンソース製品と企業あるいは企業グループ内に閉じて開発・維持する固有の機能との組み合わせ製品の維持管理環境，そして説明性，追跡性の要求に応えられる環境確保の課題が明らかになった．

3.3 まとめ

クラウド型開発管理環境の構築・評価に関する成果は次の通りである．

① クラウド型開発管理環境の構成・運用方式の評価

商用のクラウドサービスを用いて実現した統合的なソフトウェア開発管理環境が，説明性，追跡性の対象としての第三者を含んだ広義のソフトウェア開発関係者を結ぶ環境として有用なことがわかった．

ソフトウェア開発のマネジメント方式にはスペクトラム（構成要素分布）とも言うべき多様な幅が考えられ，クラウド型の統合的なソフトウェア開発管理環境が，それらを歩み寄らせ，より高いレベルへ導く可能性があることへの示唆が得られた．

② 組織間協業の実現可能性評価

先進的なソフトウェア開発管理環境には，技術力向上を通した地域産業の競争力，ブランド化への貢献が期待されることがわかった．また，地域産業レベルでは，差し迫った説明性・追跡性への要求は強くないことも判明した．

１．研究の背景及び目的

1.1 背景

2010 年に米国で発生したトヨタ車の急加速問題では，当初，エンジンの電子制御ハードウェア・ソフトウェアの不具合が疑われた．全米高速道路交通安全委員会（NHTSA）の依頼を受けた NASA 工学・安全センター（NESC）は，当該ソフトウェアの品質，および，品質が実現される過程を評価した．その結果，ソフトウェアの設計や実装において，急加速につながるような不具合は発見されず，急加速による事故のほとんどが運転手のミスと確認された．

NESC による評価は，専門知識を有する中立的立場の第三者が，現在のソフトウェア工学技術を用いることで，ソフトウェア品質をどのように，どこまで定量的に評価することができるのかを示した．学術的インパクトは大きく，特に，ソフトウェア工学教育の分野では，貴重なケーススタディと位置付けられ，注目を集めている[Ardis]．ソフトウェア企業にとっても，自らが開発したソフトウェアの品質を，利用者や市場にきちんと説明することの必要性，重要性を再認識するきっかけとなった．

ただし，NESC による評価には 10 か月を要した．原因の一つには，評価に必要となるデータを提供することの煩雑さがある．ソフトウェアの構成や不具合，工程等に関するデータ（ソフトウェアプロジェクトデータ）は，開発を効率よく行うために収集，蓄積されているのであり，第三者への開示に備えてのものではない．開発が終われば一部は不要となり，保存や更新がおろそかにされる場合も多い．開発管理がきちんと行われていれば，第三者による品質評価に必要なデータを迅速に提供できる，とは限らない．

もう一つの原因は，ソフトウェア開発プロジェクトを理解することの煩雑さである．ソフトウェア開発プロジェクトは多様で個別性が高いとされており，ソフトウェアとその品質が実現される過程は一様ではない．すなわち，ソフトウェア開発は数多くの作業で構成され，手法・ツールも様々である．開発プロセスは組織によって異なり，要件変更や不具合発生など不測の事態や課題の発生は避けられない．品質評価を行う第三者は，ソフトウェア評価技術はもちろんのこと，開発技術にも精通していると考えられるが，対象プロジェクトの実態に即した評価を行うためには，まず，対象プロジェクトの全容を理解することが必要となる．しかし，ソフトウェア開発プロジェクトの理解が必ずしも容易でないことは，遅延プロジェクトへの要員追加が開発速度の向上に直結しないことや，オープンソースソフトウェアコミュニティへの途中参画に多大な労力を要することからも明らかである．

1.2 研究課題

本委託研究では，ソフトウェア品質の第三者評価の普及と高度化を推進する基盤として，「(1)品質評価に必要となるソフトウェアプロジェクトデータの提供」，および，「(2)提供されたデータに基づくプロジェクト理解」を容易にする技術の確立を目指す．具体的には，ソフトウェアプロジェクトトモグラフィと名付けた新しい概念を提案し，ソフトウェアプロジェクトデータを，その概念にしたがって再構成し，解析・可視化することで(1)(2)が容易になることを，プロトタイプシステムの実装と実証実験を通じて示す．また，(1)をよ

り容易にするために，ソフトウェアプロジェクトデータをクラウド化する環境（クラウド型開発管理環境）を実現し，その有効性を同じく実証的に示す．具体的な研究課題は，次の①～⑤のとおりである．

① ソフトウェアプロジェクトトモグラフィ（Software Project Tomography）の概念設計

本委託研究では，医療におけるコンピュータ断層撮影，いわゆる，CT（Computed Tomography）を，ソフトウェア品質の第三者評価に適したデータ構造を考える上でのモデルとする．すなわち，ソフトウェア開発プロジェクトをからだに見立て，プロジェクト開始から終了まで（あるいは，現時点まで）のいくつかの時点において，プロジェクトの状況を定量的に表すスナップショットを断面画像のように作成する．スナップショットは，次の 3 つの要素で構成される．

- プロジェクト構造モデル：プロジェクトの全体構造と構成要素間の関係を，要件，作業，組織，プロダクト，課題，の各観点でそれぞれ表したモデル．ソフトウェアプロジェクトデータをそれぞれの観点で再構成することによって得られる．
- 一次解析結果：ソフトウェアプロジェクトデータの静的・動的解析の結果．プロジェクト構造モデルの各構成要素にマッピングされる．
- プロダクト断片：要件定義書，ソースコード，不具合票などの断片．一次解析結果の検証や解釈を容易にするためにプロジェクト構成モデルの各構成要素にマッピングされる．

CT において断面画像の系列から身体の 3 次元モデルが再構成できるように，スナップショットの数が十分であれば，その系列によってプロジェクトを様々な観点で可視化し，ソフトウェアやその品質が実現される過程(プロセス)を表すことも可能になると考える．スナップショットの系列によってソフトウェア開発プロジェクトを表現する手法を「ソフトウェアプロジェクトトモグラフィ（Software Project Tomography）」と命名し，解析や可視化に適したスナップショットの構造や構成要素など，基本概念を構築する（図 1-1 参照）．

② プロジェクトデータクラウド技術の開発

今日，ソフトウェア開発プロジェクトにおける構成管理，不具合管理，工程管理などは自組織内で行われる「オンプレミス（on premise）型開発管理」が主流で，ソフトウェアプロジェクトデータは自組織内で収集・蓄積されている．しかし，ソフトウェアプロジェクトデータをソフトウェア品質の第三者評価に利用することを前提とするならば，データの収集や蓄積を自組織に閉じない形態で行うことも検討すべきである．本委託研究では，開発管理のクラウドサービスと定量的プロジェクト管理ツールを組み合わせることでクラウド型開発管理環境を実現し，ソフトウェアプロジェクトデータのクラウド化に向けた基盤技術を開発する．

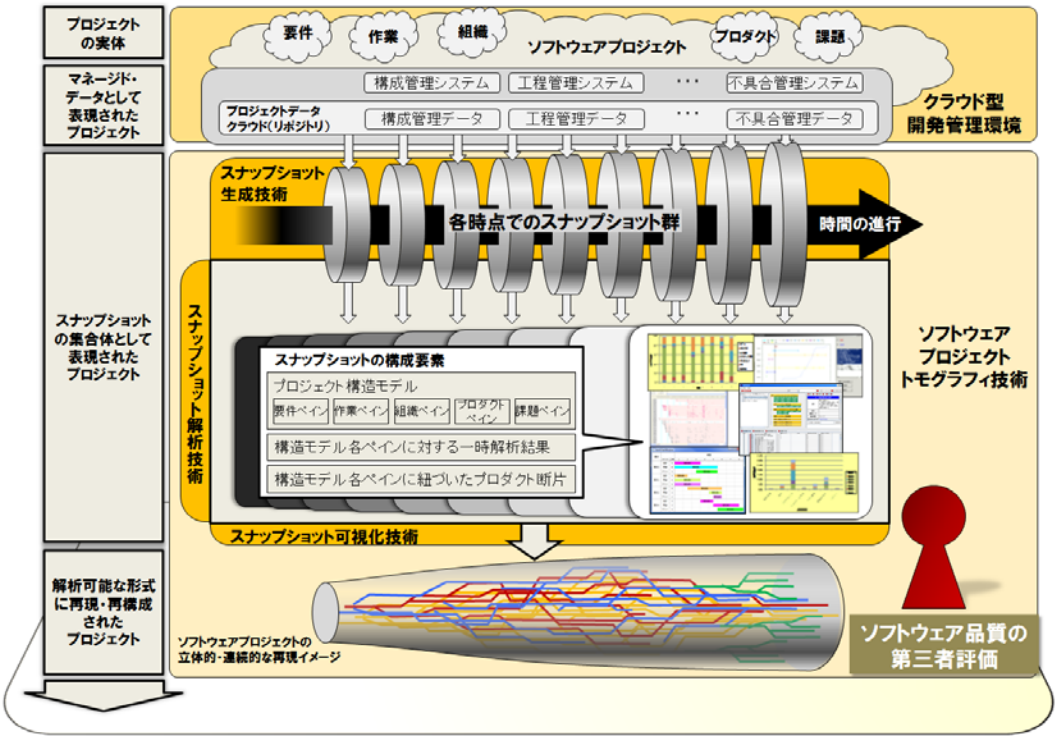


図 1-1　ソフトウェアプロジェクトトモグラフィの概念図

③ スナップショット生成技術の開発

ソフトウェアプロジェクトトモグラフィにおけるスナップショットを，プロジェクトデータクラウドに蓄積されたソフトウェアプロジェクトデータから生成する技術を開発する．生成技術は次の 3 つで構成される．

1）構造抽出技術：ソフトウェアプロジェクトデータからプロジェクト構造を抽出し，プロジェクト構造モデルとして再構成すると共に，モデルの構成要素やスナップショット間のリンクを形成する．

2）一次解析技術：ソフトウェアプロジェクトデータを静的・動的に解析し，プロジェクト構造モデルの構成要素に定量的な特性値を付与する．

3）プロダクト断片化技術：プロダクトを断片化し，プロジェクト構造モデルの構成要素に付与する．

本委託研究では，提案するスナップショットに，ソースコード，不具合票，工程表といったプロダクトを原本のまま取り込むことは考えていない．これは，解析や可視化における認知的負荷を軽減すると共に，第三者によるスナップショット利用を容認されやすくするためである．ソフトウェア品質の第三者評価では，スナップショットは組織の外に出ていくことになる．解析や可視化において，組織・プロジェクト間でのスナップショット比較も行われる．権限を持つ限られた第三者のみがスナップショットを利用するのだとしても，プロダクトそのものが外部に出ていくことには，大きな制度的・心理的抵抗が予想される．

ただし，プロダクト情報の欠落は，解析や可視化，さらには，ソフトウェア品質の第三者評価の水準や範囲を限定してしまうことになる．特に，後述する「探索型可視化」にお

いては，一次解析で得られた数値だけでは，可視化の目的が果たせない可能性が高い．そこで，プロジェクト構造モデルの構成要素（の一部）については，対応するプロダクトの断片を付与する．断片化の方法としては，例えば，ソースコードであれば，バグを除去するために変更が加えられた行やその行とクローン関係にある行（クローンペア）のみを抜き出す，不具合票の自然言語記述部分であれば，形態素解析などにより特徴語（キーワード）だけを抜き出す，工程表なら，個人名や日付などを符号化した上で抜き出す，などが考えられる．

なお，断片化したとしても，スナップショット全体を丹念に調べ，断片を集めれば，原本（の一部）を再構成したり，推測したりすることは不可能ではない．それを防ぐためには，断片そのものを暗号化するなどの対策が必要となるが，そうした対策については，将来の研究課題とし，本委託研究では行わないこととする．

④ スナップショットに基づくプロジェクト解析技術の開発

スナップショットの系列から，ソフトウェアやその品質が実現される過程を解析する技術（二次解析技術）の開発を行う．複数のスナップショットにまたがる解析（プロセス解析）は，原則，この二次解析で行われる．二次解析技術は次の 2 つで構成される．

1）検証型解析技術（Goal-driven 解析技術）

ソフトウェアやその品質が実現される過程のどこをどのように評価すればよいかが明確になっている場合の解析技術である．評価の対象や方法が明確になっているとは，「ISO/IEC 15939:2007, Software engineering -Software measurement process」における「解釈モデルや判断基準」，「指標」，および，「分析モデル」が与えられていることを意味する．検証型解析とは，分析モデルのパラメータとなる「導出測定量」もしくは「基本測定量」をスナップショット（の系列）から見つけ出し，分析モデルにしたがって指標を算出し，解釈モデルや判断基準に照らして結論を得ることと言える．

2）探索型解析技術（Data-driven 解析技術）

背景で述べた通り，品質評価を行うために，まず，ソフトウェアやその品質が実現される過程を理解するところから始めなければならない場合も多い．同じく ISO/IEC 15939:2007 上で例えるならば，「解釈モデルや判断基準」，「指標」，もしくは，「分析モデル」が与えられておらず，「基本測定量」や「導出測定量」の候補ばかりが，スナップショット（の系列）中に数多く存在する，といった状況である．探索型解析に求められるのは，（プロジェクトを跨いで）スナップショットを比較し，スナップショットやその構成要素の間に見られる特徴，傾向，規則，特異点などを発見し，ソフトウェアやその品質が実現される過程との因果関係を明らかにすることである．その結果，「解釈モデルや判断基準」，「指標」，および，「分析モデル」が構築され，検証型解析が可能となる．

⑤ スナップショットと解析結果に基づくプロジェクト可視化技術の開発

スナップショットの系列やその二次解析の結果から，ソフトウェアやその品質が実現される過程を可視化する技術の開発を行う．可視化技術は次の 2 つで構成される．

1）検証型可視化技術

検証型解析の結果を，解釈モデルや判断基準に基づいて指標をわかりやすく表示し，ソフトウェアやその品質が実現される過程の検証を容易にする．

2）探索型可視化技術

スナップショットの系列やその構成要素を多面的に表示し，ソフトウェアやその品質が実現される過程に関する仮説を立てるための材料を提供し，仮説の真偽や適用範囲の確認を容易にする．

1.3 研究の意義

(1) ソフトウェア品質に関する「情報の非対称性」の解消

現在のソフトウェア開発では，ソフトウェア品質に大きな影響を与える開発過程の情報の多くは，開発者（ベンダ）のみが知り得るものである．すなわち，ベンダと他のステークホルダー，特に，利用者（ユーザ）との間には，ソフトウェア品質に関する情報量に大きな隔たりがある．こうした「情報の非対称性」は，

- 逆選択：ユーザは低品質なソフトウェアしか選べなくなる．
- モラルハザード：ベンダが手抜きや虚偽報告をしがちになる．

といった問題を引き起こし，市場の健全な発展を阻害するとされている．対策としては，

- シグナリング：ベンダがユーザに積極的に情報発信する．
- スクリーニング：ユーザがいくつかの案をベンダに示し，選択させることを通じて情報を読み取る．
- 制度と組織：中立的立場の第三者が情報の非対称性の解消を手助けできるよう，認証や監視などのための制度や組織を整備する．

などが知られている．本委託研究で実現を目指すソフトウェアプロジェクトトモグラフィ技術は，ソフトウェア開発プロジェクトの透明性を高め，ソフトウェア品質に関する「シグナリング」の手段を提供することになる．ソフトウェア品質監査制度といった「制度や組織」と相まって，市場をより健全化し，ベンダには競争力を，ユーザには安心感を与えることの意義は大きい．

(2) ソフトウェア品質説明力の強化

日本製ソフトウェアの品質は決して低くないが，その高い品質を具体的に利用者や市場に説明する力（品質説明力）が日本企業に十分備わっているとは必ずしも言えない，あるいは，その力は備わっているが説明の必要性を強く感じるに至っていない．しかし，国際市場では，欧米流の品質保証の考え方に基づき「証拠を示してユーザに信頼感を与える．」ことが重視される[Yasuda]．長年の実績を有し，きちんと開発管理を行っていれば，品質に関する説明は不要とする日本流の品質保証は通用しない．

本委託研究の成果である「ソフトウェアプロジェクトトモグラフィ技術」は，日本のソフトウェア企業の品質説明力を高めることになる．また，「クラウド型開発管理環境」は，品質の説明に要するコストを低減すると共に，ソフトウェア品質を企業がオープンに議論する契機を与えることになり，第三者評価の普及，高度化にもつながる．日本製ソフトウェ

アの品質に対する正当な評価システムが提供されることになり，我が国のソフトウェア産業の国際競争力の維持・強化に大きく貢献することが期待される．

２．実施内容

2.1 研究アプローチ

2.1.1 研究の全体像

研究全体は次の手順で進める．研究課題間の関連を保ちながら，各課題を「準備→設計→実装→適用実験→評価」の順に進め，必要に応じて設計や実装の見直しも行う．

① 準備（既存の研究・システム・ツール等の調査），クラウド型開発管理環境の設計
② ソフトウェアプロジェクトトモグラフィの概念設計，クラウド型開発管理環境の実装
③ スナップショット生成，解析，可視化の方式設計，クラウド型開発管理環境の評価観点の設定と評価手法の策定
(中間報告)
④ スナップショット生成，解析，可視化のプロトタイプシステムの実装
⑤ クラウド型開発管理環境下で収集,蓄積されたソフトウェアプロジェクトデータへのプロトタイプシステムの適用実験
⑥ 適用実験の結果に基づく「クラウド型開発管理環境下での組織間協業」の実現可能性評価，および，ソフトウェアプロジェクトトモグラフィによる「ソフトウェア品質の第三者評価」の妥当性評価
⑦ 成果のとりまとめ，研究の将来展望・フォローアップの検討.

各研究課題におけるアプローチは次の通り.

① ソフトウェアプロジェクトトモグラフィ（Software Project Tomography）の概念設計

スナップショットは，ソフトウェアやその品質が実現される過程を表現（再現）するための多様な解析や可視化に向けて，必要なソフトウェアプロジェクトデータを扱いやすい形式で提供するプラットフォームである．スナップショット自体も可視化の対象であり，多様なソフトウェアプロジェクトデータやその解析結果(一次解析結果)の集合体となる．少なくとも次の５つの観点のペインで構成される.

- 要件ペイン（Requirement Pane）：プロジェクトで実現すべきソフトウェア要件（機能／非機能）の全体像，要件間の関係，各要件の実現状況などを表す.
- 作業ペイン（Task/Activity Pane）：プロジェクト要件を実現するために実施される作業の全体像，作業間の関係，各作業の進捗状況などを表す．例えば，作業全体をガントチャートで表され，各タスクに工数データ，進捗率などが付記されている.
- 組織ペイン（Organization Pane）：作業を実施する組織の全体像，組織構成メンバー間の関係，各メンバーの状況などを表す．例えば，ステークホルダー間のコミュニケーション関係がネットワーク図で表され，各人に役割，累積工数，累積生産量などが付記されている.
- プロダクトペイン（Product Pane）：作業によって作成・利用されるソースコードや中間成果物といったプロダクトの全体像，各プロダクトの構成要素と要素間の関係，

各構成要素の作成・利用状況などを表す．例えば，ソースコード全体が階層構造図で表され，ソースコードを構成する各モジュールに複雑さ，変更量，クローン値などが付記されている．

- 課題ペイン（Issue Pane）：ソフトウェア要件が実現されていく過程で発生する不具合や仕様変更要求といった（プロジェクト終了までに解決すべき開発上の）課題の全体像，課題間の関係，各課題への対応状況などを表す．

スナップショットの粒度（生成間隔）も，解析や可視化の利便性に大きく影響する．少なくとも次の 3 つの粒度について検討し，必要であれば併用するアプローチとする．

- Daily/Weekly/Monthly Snapshot
  日，週，月などのカレンダー時間に基づき等間隔でスナップショットを構成する．スナップショットの構成枚数は開発期間に比例する．直感的でわかりやすく，プロジェクトの開発管理の最小単位とも一致する．
- Percentile Snapshot
  開発プロジェクトの開始から終了までの経過時間（開発期間）を100分割し，開始時点と合わせて101枚のスナップショットで構成する．開発期間の50%時点での比較などが容易である．ただし，開発期間や規模が大きく異なる場合には，単純に比較してよいかどうかの検討は必要になる．
- Contextual Snapshot
  開発工程など，開発作業の節目に基づいてスナップショットを構成する．スナップショットの構成枚数は不定で，等間隔とは限らない．開発コンテキストに基づく解析や可視化は容易となるが，当然のことながら，工程の切れ目など，開発コンテキストの情報が必要となる．開発コンテキストスキームが異なるプロジェクト間での解析結果の比較や可視化は難しくなる．

② プロジェクトデータクラウド技術の開発

一般的な情報処理システムの構成法として評価が高まりつつあるクラウド環境の「ソフトウェア開発環境としての利点」に焦点を合わせ、次の 2 つの視点からアプローチする．

1）即効性・実用性の高いクラウド型のソフトウェア開発管理環境構成法の実証

プロジェクトのタスク管理，進捗管理，情報共有を可能とする Redmine，および，構成管理システム Subversion のクラウドサービスを提供する企業との連携により実証環境を開発する．さらに，独立行政法人情報処理推進機構（IPA）が新たに提供した定量的プロジェクト管理ツール（EPM-X）を利用することで，セキュリティに十分配慮しながらプロジェクトデータを安定して提供できる，高い可視性を備えた実用性の高いクラウド型開発管理環境を実現し，実プロジェクトへの適用と評価を通じてその有用性の実証を目指す．

2）クラウド方式の利点を活用した組織間協業環境の実現法の検討

クラウド方式には，広域・多岐にわたる組織・人間間での情報共有・共同作業の支援に優れた特質がある．既に，一般情報処理の分野においても，この特質を活用した新しいタイプの企業間連携の成功例が報告されている．ここでは，クラウド方式のこのような特質を活用した組織間協業，特に現場の狭義の開発関連組織だけでなく，産業構造を反映した共同開発，そして，ソフトウェア開発プロジェクトの説明や評価に関わる幅広い組織間の

協業を容易にする環境の構成法を探る．Redmine や Subversion が提供する役割（role）定義機能，および，wiki 等のコミュニケーション・ツールが提供する機能と，定量的プロジェクト管理ツール（EPM-X）によって可視化されるデータの組み合わせ評価を通じて，実用的な協業環境の実現と有用性の実証を目指す．

③ スナップショット生成技術の開発

スナップショットを構成する各ペインの生成技術を，次のような手法やツールを基に開発する．

- 要件ペイン（Requirement Pane）：本委託研究メンバーが開発してきている「Request For Proposal(RFP)における保守・運用要件指標の抽出と評価」の手法を応用する．要件間の関係の抽出には，テキストマイニングツールKH Coderを用いる．
- 作業ペイン（Task/Activity Pane）：本委託研究メンバーが開発してきている「開発タスク計測システムTaskpit」の手法を応用する．
- 組織ペイン（Organization Pane）：本委託研究メンバーが開発してきている「アウェアネス支援システムACTION」の手法を応用する．
- プロダクトペイン（Product Pane）：本委託研究実施機関の現有設備である「インテンシブ解析サーバシステム（Xeonプロセッサ X7550(8コア、2GHZ)×4，主記憶256GB，SSD 480GB)」と同システム上で実行可能な７つのソフトウェア静的解析ソフト（PG Relief C/C++ 2011,PG Relief J2011,QAC++ 2.5.1J,QAC 7.2.3J,Klocwork Insight, RSM windows, Understand 2.5）を利用する．また,「コードクローンツールCCFinder」を用いた重複コードの解析も行う．
- 課題ペイン（Issue Pane）：本委託研究メンバーが開発してきている「バグトラッキングシステムにおけるデータマイニング」の手法を応用する．

④ スナップショットに基づくプロジェクト解析技術の開発

スナップショットに対する二次解析技術を，本委託研究メンバーが開発してきている次のような手法やツールを基に開発する．

1）検証型解析技術

プロジェクト再現ツール Project Replayer，低品質モジュールに対するテスト工数割り当ての最適化．

2）探索型解析技術

類似プロジェクトのスナップショットに基づく低品質モジュールの予測，検証の前倒しシミュレーション

⑤ スナップショットと解析結果に基づくプロジェクト可視化技術の開発

1）検証型可視化技術

本委託研究メンバーが開発してきているツール 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）や重複コードの時系列分析の手法を基に開発する．

2）探索型可視化技術

スナップショットは，多次元・多変量データの集合体であり，情報可視化における従来の原則「Overview first, zoom and filter, then details on demand」は必ずしもあてはまらない．本委託研究では，多数の変数間の関係を網羅的に表し，かつ，分析者の認知的負荷の軽減にもつながる，できるだけシンプルな提示方法の実現を目指す．具体的には，Seo らが提案するフレームワーク Rank-by-feature framework を基に開発する[Seo2005][Seo2007]．

Rank-by-feature framework では，探索型可視化を行うユーザが，多次元・多変量データから興味深い仮説を導き出せるよう，１次元（ヒストグラムや箱髭図等），あるいは，２次元（散布図等）でデータが表現される．また，複数の視点からデータの全体像を素早く把握し，系統立った可視化が可能となるよう，データへの順位付けが行われる．順位付けには，データ分布の正規性や均一性，外れ値や特異値など，データの特徴（feature)を表す統計量が利用される．その一方で，3 次元表現など，ユーザの認知的負荷が大きく，解釈やデータ操作がしばしば困難となる表現形式は使用されない．

本委託研究では，スナップショットの探索型可視化において，ソフトウェアプロジェクトトモグラフィを構成する５つの観点(要求,作業,組織,プロダクト,課題)を特徴(feature)と考え，それらを表す統計量によってスナップショット（の構成要素）やその解析結果の順位付けを行うことを想定している．例えば，「プロダクト」の観点であれば，モジュールの複雑さ，変更量，クローン値などに基づいてスナップショット（の構成要素）やその解析結果を順位付けすることになる．特徴として用いる５つの観点は，ソフトウェア開発者・管理者にとってなじみ深いものである．従って，可視化結果の直感的な理解や解釈を可能にし，仮説の真偽や適用範囲の確認も容易になると考えられる．

なお，可視化結果の表示装置としては，多数の解析結果を同時に表示可能な高解像度・大画面モニター，開発者等の個人は常時利用可能な PC モニター，更には，スマートフォンなどの小型情報端末，を想定する．このうち，高解像度・大画面モニターとしては，本委託研究実施機関の現有設備である「解析結果表示装置（ハイビジョン 60 インチ×６面）」を利用する．

2.1.2 関連するこれまでの研究について

(1) 本委託研究以前に実施されていた委託研究に関連する研究について

① ソフトウェアトモグラフィ

Bowring らは，ソフトウェアトモグラフィという概念を提案している[Bowring]．これは，多数のユーザのもとで稼働しているソフトウェアの振る舞いを，ユーザに負荷をかけず表示・監視する技術である．表示・監視用のプローブやタスクを細切れにし，多数のソフトウェアで実行し，その結果を再構成するところから，トモグラフィという用語が用いられているが，本委託研究とは，対象も目的も大きく異なる．

② UCI Source Code Data Sets

UCI Source Code Data Sets [Lopes]（以下 UCI Datasets と表記する）は，カリフォルニア大学アーバイン校から提供されているソースコード集合である．UCI Datasets は約 18,000 のオープンソース Java プロジェクトを含んでいる．これらのデータはオープンソースリポジトリや開発者のリポジトリから収集された．UCI Datasets に含まれるコンテンツの概要を表 2-1 に示す．実際には 18,000 を超える Java プロジェクトが含まれているが，1 行以上の Java ソースコードが記述されたファイルが存在するプロジェクトは 13,000 程度である．

UCI Datasets は.tar 形式で配布されているが，現在配布されているものは 2010 年 4 月 22 日にアーカイブされたものである．UCI Datasets の取得は以下の手順で行う．まず，ダウンロードページ[UCI]上のフォームに氏名，Email アドレス，所属組織，利用目的を記載して送信する．次に，送信後取得できるファイルリストにしたがって分割された tar.gz ファイル（合計サイズ約 390GB）をダウンロードする．最後に分割されたファイルを結合して得られる tar.gz ファイルを解凍する．なお，ファイルを結合した時点で得られた tar.gz ファイルに md5 チェックサムを行い，フォーム送信後に得られるハッシュ値と比較することでファイル破損の有無等を確認できる．

表 2-1 UCI Datasets の概要[Ossher]

Originating Repository	Projects	Filtered Projects	Files	Filtered Files
Apache	84	83	559,140	50,222
Java.net	3,412	1,892	286,310	273,812
Google Code	5,361	4,632	402,070	368,096
Sourceforge	9,969	6,632	1,983,044	1,167,122
Other	2	2	7,346	7,346
Total	18,828	13,241	3,237,910	1,866,598

③ Understand

Understand はアメリカの SciTools[SCITOOLS1]が開発した有償のソフトウェア構造解析ツールである．日本語版はテクマトリックス[TECHMTX1]が提供している．Understand は，C，C#，Java をはじめとしたソースコードを解析してソフトウェアの構造を可視化する．ソースコード解析の機能として複雑度，ボリューム，オブジェクト指向に関するメトリクスなど約 70 種類のコードメトリクス[SCITOOLS2][TECHMTX2]を分析できる．可視化機能としては，分析したメトリクスのグラフ化（図 2-1），マップ化（図 2-2）や依存グラフ（図 2-3）などを描画できる．また，Understand は Perl や C などで扱うことのできる API（Understand API）を備えている．Understand API を利用したスクリプトを作成することで，Understand データベース（ソースコードの解析結果）へアクセスできる．ユーザは

Understand API を利用することで任意の形式でレポートが作成でき，ソースコードの解析からレポート作成までのプロセスの自動化も可能となる．

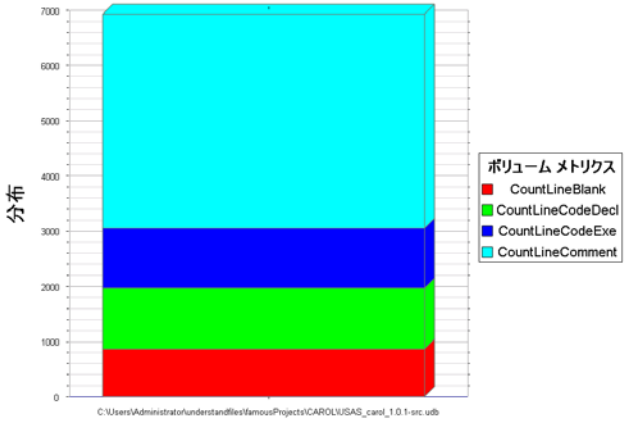


図 2-1　Understand によるボリュームメトリクスのグラフ化

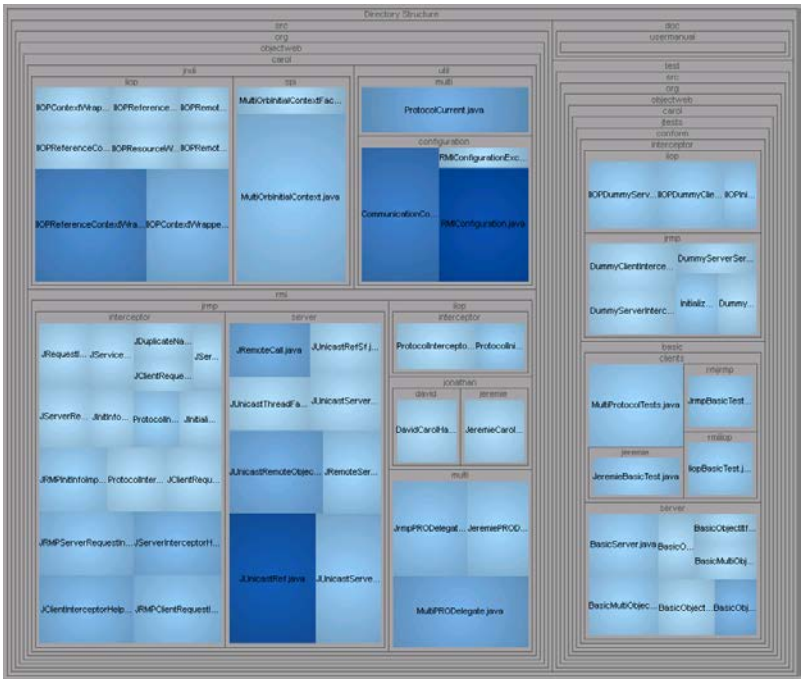


図 2-2　Understand で作成したメトリクスツリーマップ

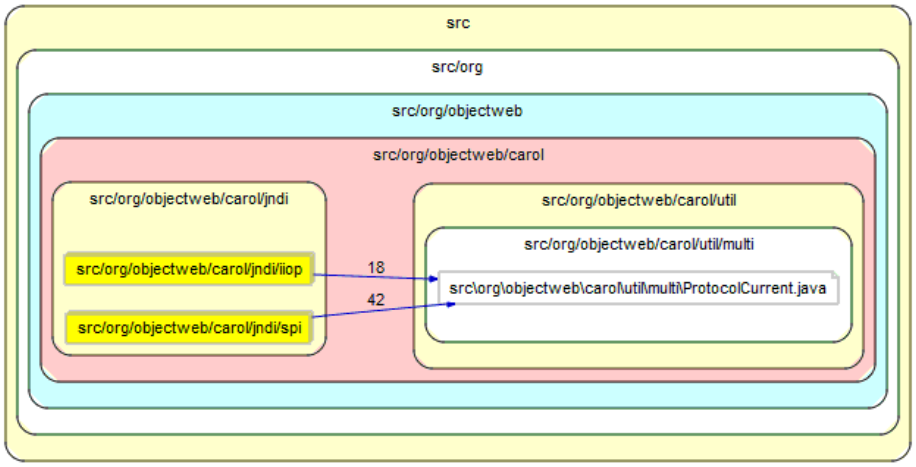


図 2-3　Understand で作成した依存関係表すグラフ

④　コードクローン分析

コードクローン[Higo]とはソースコード内の互いに類似または一致するコード片である．コードクローンが生じる理由としてコピー&ペースト，定型処理，意図的な繰り返し，偶然，プログラミング言語に適切な機能がない等があげられる．コードクローンは以下の３つに分類される．

タイプ１：空白やタブ，コメントの有無を除いて完全に一致するコードクローン

タイプ２：変数名や関数名のどのユーザ定義名や型などの一部の予約語のみが異なるコードクローン

タイプ３：タイプ２の違いに加えて文の削除，追加，修正が行われたコードクローン

一般的にコードクローンはソフトウェアの保証を困難にすると言われている．コードクローンの一つを修正した場合に，対応する全てのコードクローンに対して同様の修正を行うべきか考慮しなければならないためである．コードクローンへの対策としては以下の２つがある．

・　コードクローンの一貫した修正を行う

前述した通りコードクローンに対して修正が行われる度にコードクローンの一貫した修正を行わなければならない．

・　リファクタリングによるコードクローンの集約を行う

リファクタリングを行う事でコードクローンが１つのモジュールとしてまとめられ，将来にかるソースコードの修正に必要なコストを削減できる．

ただし，全てのコードクローンがリファクタリングによって集約するべきコードクローンであるというわけではなく，一貫した修正の必要のない無害なコードクローンも存在する．

ソースコードが大規模になるにつれて，対応する全てのコードクローンを手動で探し出す事は現実的に不可能である．コードクローンを自動的に検出するツールを用いる事が現

実的なコードクーンを探し出す方法である．コードクローンを検出する手法として以下の手法があげられる

- 行単位の検出
  ソースコードを行ごとに比較して，しきい値以上一致していればコードクローンとして検出する手法である．高速な検出が可能だが，空白やタブ，コメントの有無も一致していないものとみなすので，タイプ１のクローンの中にも検出できないものがある．
- 字句単位の検出
  ソースコードを字句単位で比較し，しきい値以上一致していればコードクローンとして検出する手法である．字句解析を用いた手法によって検出されるコードクローンはプログラミング言語の構造と一致しているわけではないのでリファクタリングを行いづらいコードクローンが検出される欠点があるが，利用しづらいコードクローンが検出されるが，抽象構文木やプログラム依存グラフを用いた手法と比べて高い計算コストを必要としないため，高速な検出が可能であるという利点がある．
- 抽象構文木による解析
  前処理としてソースコードに構文解析を行い，抽象構文木を構築した上で，抽象構文木の同形の部分木をコードクローンとして検出する手法である．抽象構文木の構築に計算コストがかかってしまう欠点があるが，抽象構文木上の一致部分がコードクローンとなるため，プログラムの構造を考慮したコードクローンを検出できる利点がある．
- データ依存グラフによる解析
  前処理としてソースコードに意味解析を行い，データ依存グラフを構築した上で，グラフ上の同形の部分グラフをコードクローンとして検出する手法である．同形の部分グラフの特定に高い計算コストがかかる欠点があるが，意味的に等しい処理を行っているコードクローンを検出できる利点がある．

ここでは字句解析を用いたコードクローン検出を行うツールである CCFinder について詳しく説明していく．CCFinder[Kamiya]によるコードクローンの検出は，以下の手順で行われる．

A) ソースコードを入力し，トークンの列にする
   ソースコードを字句単位に分割し，トークンの列にする
B) 変形ルールにより，トークン列を変形する
   変形ルールは対象のソースコードの言語により異なる．例えば，Javaの場合はパッケージ名，可視性キーワード（protected, public, synchronized等）の除去等がある．
C) パラメータ置換を行う
   変数，定数，クラス，メソッド，型名などのパラメータを全て同一のトークンに置換する．
D) マッチングアルゴリズムによりコードクローンを検出する
   トークンの列に対して接尾辞木を用いてマッチングを行う．
E) コードクローンの位置（ファイル，行，カラム）を出力する
   コードクローンの位置情報をファイル出力する．

CCFinderX というGUIツールも存在する．コードクローンであるコード片を色づけして表示する機能に加え，スキャタープロットによるコードクローンの位置を俯瞰する機能，コードクローンのメトリクスも計測し，メトリクスによる絞り込みの機能等も実装されている[GemX]．

⑤　プログラムスライス

プログラムスライスとは，ソースコード内の特定の処理に関わるステートメントの集合を，各ステートメント間の依存関係に基づいて表したものである[Weiser]．この各ステートメント間の依存関係には，コントロール依存とデータ依存が存在する．図2-5は，図2-4のソースコードを簡単なプログラム依存グラフで表したものであり，各ノードが各ステートメントを表しており，各エッジがステートメント間の依存関係を表している．この時，データ依存とは03行目と05行目の間にあるような依存関係を指す．05行目の変数ｓの計算結果は，03行目の代入文にて変数ｓに代入された値に依存しているからである．またコントロール依存とは，04行目と05行目の間にあるような依存関係を指す．05行目が実行されるか否かは，04行目の判定結果(ここではi<=a)に依存しているからである．さらに04行目と05行目には，変数ｉによるデータ依存が存在している．これは05行目の計算に用いられている変数ｉが，04行目で定義・代入されているためである．このように，2つのステートメント間に複数の依存関係が存在することもある．

プログラムスライスを得るためには，スライス基点というものを定義する必要がある．スライス基点は，一般にステートメント+変数で定義され，そのステートメントと変数は任意に決めることができる．図2-4はソースコードと，それに含まれるプログラムスライスの一例である．青色で示されている配列ｆに関するプログラムスライスは，10行目+配列ｆと定義されているスライス基点から得られたものである．このプログラムスライス(02,07,08，09，10行目)は，10行目の配列ｆの計算結果に関わるステートメントの集合となっている．

赤色で示されている変数ｓに関するプログラムスライスのスライス基点は，05行目+変数ｓとして定義されている．緑色で示されている変数ｉに関するプログラムスライスのスライス基点は，05行目+変数ｉとして定義されている．この変数ｓと変数ｉに関するプログラムスライスの例のように，スライス基点として定義したステートメントが同一であっても，同時に定義する変数が異なると，得られるプログラムスライスが変わる．

またプログラムスライスには，バックワードスライスとフォワードスライスの二種類が存在している．同一のスライス基点から，この二種類のプログラムスライスを得ることができる．バックワードスライスとは，スライス基点として定義したステートメント+変数の，計算・実行結果に影響を与えるステートメントの集合である．フォワードスライスとは，スライス基点として定義したステートメント+変数の，計算・実行結果が影響を与えるステートメントの集合である．本稿では，メトリクスの計算にバックワードスライスを用いている．

i	s	f	
			01: public void compute(){
I	I	I	02: int a = 100;
	I		03: s = 0;
I	I		04: for(int i = 0; i <= a; i++){
I	I		05: s = s + i;
			06: }
		I	07: f[0] = 1;
		I	08: f[1] = 1;
		I	09: for(int i = 2; i <= a; i++){
		I	10: f[i] = f[i-1] + f[i-2];
			11: }
			12: }

図 2-4　プログラムスライスの例

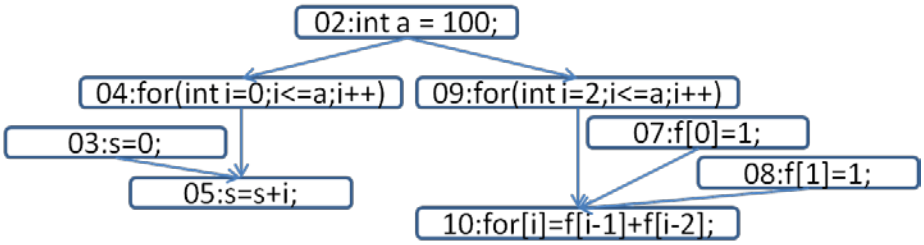


図 2-5　プログラム依存グラフ

⑥　スライスに基づいたメトリクス

凝集度メトリクスはソースコードメトリクスの一種であり，ソースコードの品質を測るためのものである．ここで紹介する凝集度メトリクスは，プログラムスライスを利用して凝集度（対象メソッドの各文が協調し合って実装されているかの尺度）を調べる．一般に凝集度が高いメソッドの各ステートメントは互いに密な依存関係を持っており，ひとつの機能を実現するためのメソッドであると考えられる．

プログラムスライスを用いた凝集度メトリクスの代表的なものに Weiser の定義した凝集度メトリクスが存在する[Weiser]．Weiser の定義した凝集度メトリクスは Tightness, Overlap, Coverage, Parallelism, Clustering の 5 種があるが，中でも Overlap, Coverage, Tightness の有用性が高いとされている[Krinke]．ここではツールで用いた Overlap, Coverage, MinCoverage, MaxCoverage, Tightness について説明する．

凝集度メトリクスの計算を行うメソッドを M，メソッドに含まれるプログラムスライスの総数を V_o，x番目のプログラムスライスの長さを SL_x，全てのスライスに共通する部分の長さを SL_{int} としたとき，

$$\mathrm{Overlap}(M) = \frac{1}{V_o}\sum_{x \in V_o}\frac{|SL_{int}|}{|SL_x|}$$

で表される．これは，対象メソッドに含まれる全てのスライスの共通部分の長さを各スライスで割ったものを，足し合わせてスライスの数で割ったものである．Overlap は各スライスに含まれる共通部分の割合を表している．Overlap が高い場合，そのメソッド内の各ステートメントは非常に高い依存関係を持っている可能性がある．

凝集度メトリクスの計算を行うメソッドをM，メソッドに含まれるプログラムスライスの総数をV_o，x番目のプログラムスライスの長さをSL_x，メソッドの長さを$length(M)$としたとき，

$$Coverage(M) = \frac{1}{V_o}\sum_{x \in V_o}\frac{|SL_x|}{length(M)}$$

で表される．これは，対象メソッドに含まれる全てのスライスの長さの平均を，対象メソッド全体の長さで割ったものである．Coverage はメソッドに対して，意味のあるスライスの割合を表している．Coverage が低い場合，そのメソッドは複数の機能を含んでいる可能性がある．また本稿では標準の Coverage に加え，Ott らによって定義された MinCoverage と MaxCoverage という凝集度メトリクスを用いた[Meyers]．これは，

$$MinCoverage(M) = \min_i \frac{|SL_i|}{length(M)}$$

$$MaxCoverage(M) = \max_i \frac{|SL_i|}{length(M)}$$

で表される．MinCoverage は，メソッド内で最も小さなスライスの長さを，対象メソッド全体の長さで割ったものである．また MaxCoverage は，メソッド内で最も大きなスライスの長さを，対象メソッド全体の長さで割ったものとなっている．

凝集度メトリクスの計算を行うメソッドをM，全てのスライスに共通する部分の長さをSL_{int}としたとき，メソッドの長さを$length(M)$としたとき，

$$Tightness(M) = \frac{|SL_{int}|}{length(M)}$$

で表される．これは，対象メソッドに含まれる全てのスライスの共通部分の長さを，対象メソッド全体の長さで割ったものである．Tightness が高い場合，そのメソッドの全てのスライスは同じ機能を実現するために実装されている可能性がある．

スライスに基づいたメトリクスを計算するためには，対象メソッドからプログラムスライスを得る必要がある．しかし，プログラムスライスを得るために必要なスライス基点として定義するステートメント+変数は任意に定めることができ，全てのスライス基点からプログラムスライスを得てしまうと，凝集度メトリクスを正しく測ることができない．そこで本委託研究では，一般にバックワードスライスのスライス基点として用いられる，メソッドの出力に相当するステートメントのみ，スライス基点として用いた[Meyers]．具体的には，Return ステートメントや，外部の変数(フィールド変数など)に代入を行うステートメントを，メソッドの出力に相当するステートメントと定義した．

図 2-4 の例では，変数 s と配列 f はフィールド変数のため，これに代入を行うステートメントが，このメソッドの出力に相当する．そのため図 2-5 に示されている，変数 s と配列 f に関するスライスのみを，凝集度メトリクスの計算に利用する．03,07,08 行目もフィールド変数へ代入を行っているステートメントだが，ここでは計算の簡略化のために省略す

る．この時，メソッドの長さを 8 として計算すると，Overlap の値は 20 分の 9，Coverage の値は 16 分の 9，Tightness の値は 8 分の 1 となる．このように図 2-4 の例では，プログラムスライスを利用した凝集度メトリクスの値は，どれも小さくなることがわかる．これはメソッド内部が，変数 s を計算する箇所と変数 f を計算する箇所に，明らかに分かれているためであり，モジュール化が正しく成されていないためであると考えられる．このように凝集度メトリクスの値は，そのメソッドに含まれる機能の協調度合いを表している．

⑦　ソフトウェアプロジェクトのインプロセス計測とフィードバック

IPA ソフトウェア・エンジニアリング・センター（SEC）では，進行中のプロジェクトの計測と可視化をテーマとして，計測プラットフォーム EPM を実開発プロジェクトに適用する大規模な実証実験が実施されてきている[SEC]．その代表例が，SEC 先進プロジェクトと SEC 国プロである．

SEC 先進プロジェクトでは，エンタープライズ系ソフトウェア開発の一つとして，交通情報分野のプローブ情報システム開発を対象とし，大規模開発型のマルチベンダプロジェクトにおいて，進行中のプロジェクト計測と可視化情報のフィードバックが行われた．

SEC 国プロでは，組み込み系ソフトウェア開発の一つとして，車載用マイコン（ECU）のミドルソフトウェアの開発を対象とし，大手自動車製造企業，大手サプライヤー，ソフトウェア企業群という産業構造の中で，進行中のプロジェクト計測とフィードバックが行われた．

他にも，一般社団法人情報サービス産業協会（JISA）で研究会が組織され，研究会メンバーの各企業が EPM を自社で装備し，実プロジェクトにおいてインプロセス計測を行い，その評価結果を持ち寄るといった活動も行われている．

(2) 研究責任者が本委託研究以前に実施していた研究と本委託研究との関係について

・ 文部科学省委託研究「次世代ＩＴ基盤構築のための研究開発：ソフトウェア構築状況の可視化技術の開発普及（エンピリカルデータに基づくソフトウェアタグ技術の開発と普及）」（平成 19 年 8 月～平成 24 年 3 月）

ソフトウェアに対するトレーサビリティの概念を普及させ，世界最高水準の安心・安全なＩＴ社会を実現するため，ソフトウェア開発が適正な手順で行われたかどうかをソフトウェア発注者によって把握･検証可能とすることを目指し,エンタープライズ系ソフトウェア，組込み系ソフトウェアを問わず，オフショアを含むマルチベンダによるソフトウェア開発に関する実証的データ（エンピリカルデータ）を収集し，「ソフトウェアタグ」としてソフトウェア製品に添付して提供する技術を開発することを目的とした委託研究プロジェクトである[MatsuATGSE][MatsuICSEC][MatsuCSPIN][MatsuSEC][MatsuSPES].研究開発項目とそれぞれの主な成果は次のとおりである．

1）ソフトウェアタグの規格化と収集技術に関する研究

ソフトウェアタグの規格化に関しては，ソフトウェアタグ規格第 1.0 版の開発・公開と国際標準化機構 ISO/IEC での標準化に向けた取り組みを行った．収集技術に関しては，ソ

フトウェアタグ規格の一部項目の自動収集機能を備えたソフトウェアタグ生成ツール CollectTag の試作と評価実験を行った.

2) ソフトウェアタグの可視化と適用に関する研究
ソフトウェアタグデータを開発プロセスの管理やベンダ・ユーザ間の情報共有等に役立つ可視化ツール群を開発した. また, タグやツールを活用するためのシナリオ群を開発した. さらに, ケーススタディ等を通じてこれら成果物の評価を行った.

3) ソフトウェア構築可視化に伴う法的諸問題に関する研究
ソフトウェア開発に関する紛争事例の分析を通じて, 紛争の多くが要件定義に明確に書かれていない仕様の解釈にあることを明らかにし, ソフトウェアの構築状況の早期の把握が紛争の減少, 解決につながることを見出した.

当該研究プロジェクトは, ソフトウェア開発が適正な手順で行われたかどうかを表す実証データを, ユーザ・ベンダ間で共有するための技術（ソフトウェアタグ技術）を開発しようとしたものである. これに対して, 本委託研究では, ソフトウェアタグ技術をベースとして, ユーザでもベンダーでもない「第三者」によるソフトウェア品質評価の技術基盤を開発しようとするものである.

2.1.3 研究目標

本委託研究では, 「ソフトウェア品質の第三者評価」の必要性, 重要性が高まりつつある現状に鑑み, その普及と高度化を推進する基盤として,

① 品質評価に必要となるソフトウェアプロジェクトデータの提供
② 提供されたデータに基づくプロジェクト理解

を容易にする技術を確立すべく, ソフトウェアタグ技術をはじめとして, 研究責任者らによるこれまでの研究成果に基づき, 次の 2 つの研究目標を設定した.

研究目標１：ソフトウェアプロジェクトトモグラフィ技術の確立
① ソフトウェアプロジェクトを表現するに十分な情報を含むスナップショットの実現
② スナップショットの生成・解析・可視化システムの試作
③ 試作システムを用いた「ソフトウェア品質の第三者評価」の妥当性評価

研究目標２：クラウド型開発管理環境の有効性の実証と課題の抽出
④ クラウド型開発管理環境の構成方式および運用方式の評価
⑤ クラウド方式の利点を活用した組織間協業の実現可能性評価

2.2 研究の活動実績・経緯

(1) 活動実績

本委託研究で設定した作業項目毎の実施スケジュールを図 2-6 に示す.

図 2-6　研究実施スケジュール

(2) 内部打合せの実施状況

研究責任者は，本委託研究メンバー（アドバイザ，研究技術員を除く）が参加する進捗報告会を毎週開催し，各メンバーから，メンバー自身，および，研究技術員による研究開発の進捗状況の報告を受けると共に，IPA からの連絡事項の伝達，中間報告会や内部レビューの実施に向けた準備・調整などを行った．それらに基づき，各メンバーに必要な指示を行うと共に，各月の進捗報告書および進捗予定管理表を作成し，IPA へ提出した．なお，メンバーの所在地が，奈良，東京，和歌山と分散しているため，進捗報告会の実施においては，必要に応じて，テレビ会議システムを利用した．また，議事，案件，成果物，報告資料等の共有と管理のため，プロジェクト管理ソフトウェア Redmine，企業向けソーシャルネットワークシステム Yammer，および，オンラインストレージサービス Dropbox を用いた．

進捗報告会とは別に，アドバイザを含む本委託研究メンバー全員が参加する「研究成果の内部レビュー」を 4 回実施した．第 1 回と第 4 回は，テレビ会議システムを利用し，参加者が奈良，もしくは，東京に会して実施した．第 2 回，第 3 回については，メール等による資料回覧とコメントフィードバックにより実施した．第 1 回は本委託研究プロジェクト開始 1 か月後の 7 月に開催し，プロジェクトの目標や実施計画などについての確認と議論を行った.第 2 回は中間報告会前に実施し,中間報告会用スライドの確認と議論を行い，議論の結果に基づいてスライドの加筆修正を行った．第 3 回は中間報告会直後に実施し，中間報告会での質疑内容などの報告と議論を行い，議論の結果に基づいてプロジェクト後半の研究内容の確認とスケジュールの調整を行った．第 4 回は成果報告会直前に実施し，成果報告会用スライドの確認と議論を行い，議論の結果に基づいてスライドの加筆修正を行った．

(3) 外部打合せの実施状況

クラウド型開発環境の研究に関して，課題管理システム Redmine と構成管理システム Subversion のクラウド形態のサービス MyRedmine，MySubversion を商用提供しているファーエンドテクノロジー社（島根県，松江市）とパートナ関係を築き，密接に連携して研究を進めた．

島根県を中心に当該サービスを利用している企業群を対象に現地調査，意見交換を進めた．

IPA/SEC 提供の「定量的プロジェクト管理ツール」の使用にあたって，SEC 研究員および製造担当のフォーレスト社（仙台）の協力・支援を得，また意見交換した．

(4) 学会参加状況とその成果

① MSR Vision 2020

開催期間： 2012 年 8 月 20 日～8 月 24 日

開催場所： Queen's University,Kingston, Canada

運営委員： Ahemd E. Hassan, Queen's university
Bram Adams, École Polytechnique de Montréal
Daniel German, Victoria university

開催概要： Mining Software Repository の将来像（2020 年における Mining Software Repository）と，その実現に向けた技術的挑戦について議論する.

参加目的： 当該分野の研究動向を知り，本委託研究における「OSS ソフトウェアプロジェクトデータの利用」や「探索型の解析・可視化技術」などについて参加者と意見交換を行う.

本委託研究組織からの参加者：伊原彰紀

主な成果：

1) 本委託研究の成果の一つである「低品質モジュールの予測」に関する発表，及び，ポスター展示を行い，参加者との意見交換を行った．発表・展示タイトルは「Understanding the distribution of bugs due to software change」.
2) 少人数グループに分かれての議論において，当該分野においては，企業との共同研究の例は極めて少なく，本委託研究の成果により「品質評価に必要となるソフトウェアプロジェクトデータの提供」が容易になることの意義を再確認することができた.
3) 少人数グループに分かれて議論において，課題票，ソースコード，メーリングリストから抽出可能，かつ有用な情報についての知見を得ることができた．これに基づき，会議参加後，「低品質モジュールの予測」に用いるメトリクスの選定を行った.

② 8th International Conference on Predictive Models in Software Engineering (PROMISE 2012)

開催期間： 2012 年 9 月 21 日～9 月 22 日

開催場所： Lund University, Sweden

プログラム委員長：Stefan Wagner, University of Stuttgart, Germany

開催概要： ソフトウェア開発における予測モデルに関する国際会議．会議主催団体が，研究者向けにソフトウェアプロジェクトデータのリポジトリを作成,公開しており，当該リポジトリのデータに基づく検証型の研究発表が数多く行われる.

参加目的： 当該分野の研究動向を知り,本委託研究における「検証型の解析・可視化技術」および「クラウド型開発管理環境」について参加者と意見交換を行う.

本委託研究組織からの参加者：神谷芳樹

主な成果：

1) 会議参加者に対して，本委託研究で対象としている「検証型の解析・可視化技術」の基本概念について紹介し，意見交換を行った．予測モデルは，検証型解析技術を実プロジェクトに展開・利用する一つのアプローチ（「解析によって検証された結果に基

づき予測のためのモデルを作成する」というアプローチ）と位置づけることができ，本委託研究の成果が，当該分野に対しても大きく貢献できることが確認できた．

2) 会議参加者に対して，本委託研究で対象としている「クラウド型開発管理環境」の基本概念と実証実験について紹介し，ソフトウェア開発における予測で用いられるベンチマークデータの収集・分析という観点から意見交換を行った．なお，IPA/SEC からは「データ白書」としてベンチマークデータの公開を行っていること，また，その一部は英訳もなされていることも，合わせて紹介した．

③ 20th International System on the Foundations of Software Engineering (FSE 2012)

開催期間： 2012 年 11 月 11 日～11 月 16 日

開催場所： Cary, North Carolina, USA

プログラム委員長：Martin Robillard, McGill University, Canada
Tevfik Bultan, University of California, Santa Barbara, USA

開催概要： ソフトウェア工学分野において，ICSE と並ぶ権威ある国際会議．ICSE よりも理論寄りの論文が多数発表される．

参加目的： 当該分野の研究動向を知り，本委託研究で提案する「スナップショットにおけるプロジェクト構造モデル」や「探索型の解析・可視化技術」などについて参加者と意見交換を行う．

本委託研究組織からの参加者：伊原彰紀

主な成果：

1) 基調講演として，IBM が開発したクイズ番組対戦用コンピュータ「WATSON」に組み込まれている自然言語処理技術に関する講演を聴講した．近年，ソフトウェア工学分野において同技術の重要性が指摘され，研究も増加しているとの指摘がなされた．本委託研究においても，スナップショットの構成要素の一つである「要件」の解析において同技術を活用しており，当該分野の動向と一致していることを再確認することができた．

2) 一般論文発表として，ソースコード・課題票からのデータ抽出，課題票と課題修正のために変更されたソースコードの紐付け，ソースコードの変更履歴の分析などに関する論文の発表を聴講し，本委託研究の内容の一つである「低品質モジュールの予測」に応用可能な知見を得た．

④ 19th Asia-Pacific Software Engineering Conference (APSEC 2012)

開催期間： 2012 年 12 月 4 日～12 月 7 日

開催場所： Hong Kong, China

プログラム委員長：
Karl Leung, The Hong Kong Institute of Vocational Education, China
Pornsiri Muenchaisri, Chulalongkorn University, Thailand

開催概要： ソフトウェア工学分野においてアジア太平洋地区で最も権威ある国際会議．ソフトウェアプロジェクトデータの解析や可視化に関する研究発表が多数行われ，当該分野の研究者が多く集まる．

参加目的： 当該分野の研究動向を知り，本委託研究で対象としている「探索型の解析・可視化技術」について参加者と意見交換を行う．

本委託研究組織からの参加者：吉田則裕

主な成果：

一般論文発表において，ソースコート解析に関する数多くの論文の発表を聴講し，本委託研究の内容の一つである「探索型の解析・可視化技術」に応用可能な知見を得た．それら論文のうち主要な２編の概要は次のとおりである．

[1] Ferdian Thung, David Lo, Lingxiao Jiang, Diffusion of Software Features: An Exploratory Study

新機能は数多くのソフトウェア・ライブラリにおいて頻繁に提案されている．これら機能は新しいメソッド，クラス，パッケージ等を含んでいる．これら機能は，多くのオープンソースソフトウェアや商用ソフトウェアにおいて使用されている．これらのうちのいくつかは，素早く適用されるが，その他は適用までに長い時間を要する．各機能は，開発，テスト，ドキュメント化に大きなリソースを要している．ライブラリの開発者やマネージャは，どの機能を優先して，次に何を開発するか決める必要がある．これらのステークホルダーを支援するために，この研究では Java Development Kit(JDK)を対象として，機能が分散している様子や，機能の適用がなされる割合を調査している．

[2] Liguo Yu, S. Ramaswamy, A. Vaidyanathan, Understanding the Effects of Code Clones on Modularity in Software Systems

モジュール性は，ソフトウェアの設計において重要な概念である．高品質のソフトウェアを設計するためには，重複コードを避ける必要がある．ソフトウェア保守を行う際，新しいコードが頻繁に追加される．コードクローンは，製品をリリースする度に増加することは明白である．そのような増加は，新しいコード断片が類似した機能やドライバと関連づいたときに手に負えなくなる．この論文では，Linux カーネルの２つのバージョンのコードクローンをモジュール性の観点から分析している．この分析において，Linux はコードクローンを削減するために労力を割いているが，多くのクローンがリリース時に追加されていることがわかった．

2.3 研究実施体制

本委託研究における研究実施体制を図 2-7 に示す．また，研究メンバーのプロフィールを表 2-2 に示す．

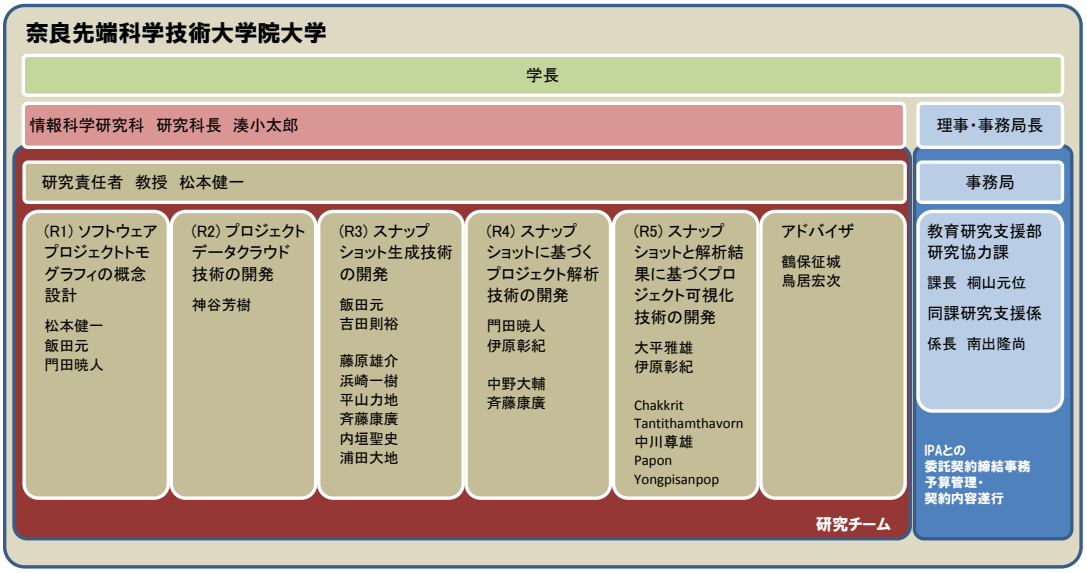


図 2-7　研究実施体制

表 2-2　研究メンバーのプロフィール

氏名	松本健一
最終学歴 主な経歴	「研究責任者のプロフィール」を参照
主な研究分野	エンピリカルソフトウェア工学 ソフトウェアメトリクス，プロジェクトデータ収集・利用支援
氏名	飯田元
最終学歴	大阪大学基礎工学研究科博士後期課程中途退学
主な経歴	平成 3 年　大阪大学基礎工学部情報工学科・助手 平成 7 年　奈良先端科学技術大学院大学・助教授 平成 17 年　同大学・教授 博士（工学）（大阪大学）
主な研究分野	ソフトウェア工学 ソフトウェア開発マネジメント支援

表 2-2　研究メンバーのプロフィール（つづき）

氏名	門田暁人
最終学歴	奈良先端科学技術大学院大学情報科学研究科博士後期課程修了
主な経歴	平成 10 年 奈良先端科学技術大学院大学情報科学研究科・助手 平成 16 年 同大学・助教授 平成 19 年 同大学・准教授 博士（工学）（奈良先端科学技術大学院大学）
主な研究分野	ソフトウェアメトリクス，ソフトウェアプロテクション， ヒューマンファクタ
氏名	大平雅雄
最終学歴	奈良先端科学技術大学院大学情報科学研究科博士後期課程修了
主な経歴	平成 15 年 同大学・産学官連携研究員 平成 16 年 同大学・助手（平成 19 年より助教） 平成 24 年 和歌山大学システム工学部・講師 博士（工学）（奈良先端科学技術大学院大学）
主な研究分野	エンピリカルソフトウェア工学 リポジトリマイニング，オープンソースソフトウェア，知的協調作業支援
氏名	吉田則裕
最終学歴	大阪大学大学院情報科学研究科博士後期課程修了
主な経歴	平成 22 年　奈良先端科学技術大学院大学・助教 博士（工学）（大阪大学）
主な研究分野	ソフトウェア工学 コードクローン分析，リポジトリマイニング
氏名	伊原彰紀
最終学歴	奈良先端科学技術大学院大学情報科学研究科博士後期課程修了
主な経歴	平成 24 年　奈良先端科学技術大学院大学・助教 博士（工学）（奈良先端科学技術大学院大学）
主な研究分野	エンピリカルソフトウェア工学 ポジトリマイニング，オープンソースソフトウェア

表 2-2　研究メンバーのプロフィール（つづき）

氏名	神谷芳樹
最終学歴	奈良先端科学技術大学院大学情報科学研究科博士後期課程修了
主な経歴	昭和 48 年 日本電信電話公社、横須賀電気通信研究所 平成 5 年　NTT ソフトウェア（株） 平成 15 年 奈良先端科学技術大学院大学・産学官連携研究員 平成 16 年(兼) IPA ソフトウェア･エンジニアリング･センター･研究員 平成 20 年 奈良先端科学技術大学院大学・非常勤講師 平成 24 年 奈良先端科学技術大学院大学・博士研究員 博士（工学）（奈良先端科学技術大学院大学）
主な研究分野	エンピリカルソフトウェア工学 ソフトウェア産業論
氏名	鶴保征城
最終学歴	大阪大学大学院工学研究科博士前期課程修了
主な経歴	平成元年　NTT ソフトウェア研究所・所長 平成 9 年　NTT ソフトウェア（株）・代表取締役社長 平成 15 年 高知工科大学工学部情報システム工学科・教授 平成 16 年 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター・所長 平成 21 年 学校法人・専門学校ＨＡＬ東京・校長
主な研究分野	ソフトウェア工学
氏名	鳥居宏次
最終学歴	大阪大学大学院工学研究科博士後期課程修了
主な経歴	昭和 50 年 通商産業省工業技術院電子技術総合研究所ソフトウエア部言語処理研究室長 昭和 59 年 大阪大学基礎工学部情報工学科・教授 平成 5 年　奈良先端科学技術大学院大学・教授 平成 13 年 同大学・学長 工学博士（大阪大学）
主な研究分野	ソフトウェア工学

・研究責任者のプロフィール

（ふりがな） 氏　名	まつもと　けんいち 松本　　健一
生年月日	昭和 37 年 8 月 21 日
所属機関・ 団体名	国立大学法人　奈良先端科学技術大学院大学
所属 （部署名）	情報科学研究科
役　職	副研究科長／教授
住　所	〒630-0192 奈良県生駒市高山町８９１６番地の５
TEL	0743-72-5310
E-mail	matumoto@is.naist.jp

【学歴（大学卒業以降）】	【職歴】
昭和 60 年 3 月　大阪大学基礎工学部卒業 昭和 62 年 3 月　大阪大学基礎工学研究科博士前期課程修了 平成元年 4 月　大阪大学基礎工学研究科博士後期課程中途退学	平成元年 4 月　大阪大学基礎工学部・助手 平成 5 年 4 月　奈良先端科学技術大学院大学情報科学研究科・助教授 平成 13 年 4 月　同・教授

【研究実績】

ソフトウェアの品質や生産性を定量的に評価するための理論的枠組と基盤技術の確立を主な研究テーマとし，2007 年からは，文部科学省次世代ＩＴ基盤構築のための研究開発「ソフトウェア構築状況の可視化技術の普及」の研究代表者として，ソフトウェアタグと呼ばれる新しい概念と技術の開発に取り組んできている．情報処理推進機構　2006 年度「SEC journal」論文発表会 最優秀賞（2006 年 10 月），同 優秀賞（2008 年 10 月），情報処理推進機構　第４回 IPA 賞（ソフトウェアエンジニアリング部門）（2006 年 10 月），電子情報通信学会論文賞（1993 年 3 月），ESEM 2007 Best Paper Award（2007 年 9 月）等を受賞．Profes 2004 組織委員長， MENSURA 2011 大会委員長，ICSE 2006 および 2012 プログラム委員，日本ソフトウェア科学会理事，情報処理学会論文誌編集委員会主査，International Journal of Empirical Software Engineering 編集委員などを歴任.

【主な論文・著書】

井上克郎，松本健一，飯田元：ソフトウェアプロセス，共立出版，東京　（平 12-3）.
井上克郎，松本健一，鶴保征城，鳥居宏次：“実証的ソフトウェア工学環境への取り組み”，情報処理，Vol.45，No.7，pp.722-728　（平 16-7）.
松本健一，“ソフトウェアの定量化と計測”　及び　“定量尺度”，情報システムのための情報技術辞典，情報システムと情報技術事典編集委員会（編），pp.4-61-4-65，培風館（平 18-6）.

・役割分担

<table>
<tr><td>実施機関名</td><td colspan="3">国立大学法人　奈良先端科学技術大学院大学情報科学研究科</td></tr>
<tr><td>研究責任者</td><td colspan="3">教授　松本　健一</td></tr>
<tr><td>連絡担当者</td><td colspan="3">教授　松本　健一</td></tr>
<tr><td rowspan="20">主な実施場所及び研究者</td><td colspan="3">国立大学法人　奈良先端科学技術大学院大学
〒630-0192　奈良県生駒市高山町８９１６番地の５</td></tr>
<tr><td>氏名</td><td>職種</td><td>担当内容</td></tr>
<tr><td>松本健一</td><td>教授</td><td>全体のとりまとめ，概念設計</td></tr>
<tr><td>飯田元</td><td>教授</td><td>概念設計,スナップショット生成技術の開発</td></tr>
<tr><td>門田暁人</td><td>准教授</td><td>概念設計,プロジェクト解析技術の開発</td></tr>
<tr><td>大平雅雄</td><td>博士研究員</td><td>プロジェクト可視化技術の開発</td></tr>
<tr><td>吉田則裕</td><td>助教</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>伊原彰紀</td><td>助教</td><td>プロジェクト解析技術･可視化技術の開発</td></tr>
<tr><td>神谷芳樹</td><td>博士研究員</td><td>プロジェクトデータクラウド技術の開発</td></tr>
<tr><td>鶴保征城</td><td>非常勤講師
(アドバイザ)</td><td>研究内容・進捗への助言，TERAS との連携</td></tr>
<tr><td>鳥居宏次</td><td>非常勤講師
(アドバイザ)</td><td>研究内容・進捗への助言</td></tr>
<tr><td>Chakkrit Tantithamthavorn</td><td>研究技術員</td><td>プロジェクト可視化技術の開発</td></tr>
<tr><td>藤原雄介</td><td>研究技術員</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>濵﨑一樹</td><td>研究技術員</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>平山力地</td><td>研究技術員</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>中川尊雄</td><td>研究技術員</td><td>プロジェクト可視化技術の開発</td></tr>
<tr><td>中野大輔</td><td>研究技術員</td><td>プロジェクト解析技術の開発</td></tr>
<tr><td>Papon Yongpisanpop</td><td>研究技術員</td><td>プロジェクト可視化技術の開発</td></tr>
<tr><td>斉藤康廣</td><td>研究技術員</td><td>スナップショット生成技術の開発,プロジェクト解析技術の開発</td></tr>
<tr><td>Thongtanunam Patanamon</td><td>研究技術員</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>内垣聖史</td><td>研究技術員</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>浦田大地</td><td>研究技術員</td><td>スナップショット生成技術の開発</td></tr>
<tr><td>経理責任者</td><td colspan="3">教育研究支援部研究協力課　課長　桐山元位</td></tr>
<tr><td rowspan="3">主な実施場所及び担当者</td><td colspan="3">国立大学法人　奈良先端科学技術大学院大学
〒630-0192 奈良県生駒市高山町８９１６番地の５</td></tr>
<tr><td>氏名</td><td>所属・役職</td><td>担当内容</td></tr>
<tr><td>南出隆尚</td><td>教育研究支援部研究協力課研究支援係・係長</td><td>IPA との委託契約締結事務
予算管理・契約内容遂行</td></tr>
</table>

３．研究成果

3.1 研究目標１「ソフトウェアプロジェクトトモグラフィ技術の確立」

3.1.1 当初の想定

(1) 想定する仮説等

①スナップショットに関する仮説等
　ソフトウェアプロジェクトを表現するに十分な情報を含むスナップショットを定義することができる.
②スナップショットの生成・解析・可視化方式に関する仮説等
　プロジェクトデータクラウドに蓄積されたデータからスナップショットを構成することができる.
③「ソフトウェア品質の第三者評価」の妥当性評価実験に関する仮説等
　スナップショットに基づく解析と可視化によって，ソフトウェア品質の第三者評価が容易になる.

(2) 当初の到達目標

①スナップショットに関する到達目標
　ソフトウェアプロジェクトを表現するに十分な情報を含むスナップショットの実現
②スナップショットの生成・解析・可視化方式に関する到達目標
　スナップショットの生成・解析・可視化システムの試作
③「ソフトウェア品質の第三者評価」の妥当性評価実験に関する到達目標
　試作システムを用いた「ソフトウェア品質の第三者評価」の妥当性評価

(3) 当初の期待される効果

① スナップショットに期待される効果
　ソフトウェアプロジェクトデータの共有や再利用の促進.
　ソフトウェアプロジェクトの解析や可視化に要するコスト低減.

② スナップショットの生成・解析・可視化方式に期待される効果
1) 生成方式
　ソフトウェアプロジェクトデータを解析し，プロジェクト構造モデルの構成要素にメトリクス値を付与することで，第三者が容易に定量化データの解析・可視化を行うための一次解析データを提供できると期待される.

[1] 要件ペイン

ソフトウェア委託開発を成功させるためには，開発初期にユーザ要件を明確にすることが求められる．その拠り所として，ユーザ要件が記載される提案依頼書（RFP）や要件定義書の品質が鍵となる．特に，非機能要件は，ソフトウェアアーキテクチャの決定や，保守コストに大きく影響するため重要である．本ペインでは，非機能要件が漏れなく，明確にRFPに記載されているか否かを評価する方法を提案し，第三者によるユーザ要件の評価を可能とする．この評価表により評価した結果は，いかなる非機能要件が要求されているかを定量的に把握できることが期待される．

[2] 作業ペイン

ソフトウェア開発者もしくは開発チームの開発プロセスの評価を行うためには，(1)どういった作業（タスク）にどの程度の時間を費やしているか，(2)どの程度の作業量を各タスクに費やしているか，(3)各タスクの成果物の量，をそれぞれ把握することが重要である．本ペインでは，このようなタスク単位のスナップショットを計測・記録することで，第三者による開発プロセスの評価が可能となる．

従来，開発の作業時間を評価するためには，作業日報が用いられてきたが，作業日報は（第三者にとって）必ずしも信頼できるとは限らない．作業をしていなくても，作業をしたと報告する可能性があるためである．本委託研究では，人手によって記録される作業日報を補うものとして，コンピュータを用いて行うソフトウェア開発関連の作業の自動計測を行うことで，より客観性の高いデータ計測と評価を可能となる．

[3] 組織ペイン

予定されていた組織構成で作業が適切に行われていたかどうかを第三者が確認するためには，開発チームもしくは開発作業を実施する組織の全体像を把握できるようにする必要がある．本ペインでは，ステークホルダー間の実際のコミュニケーション関係などをネットワーク図で表現することにより，開発開始時に想定しいた組織構成との乖離がどの程度起きているのかを評価できるようになる．

[4] プロダクトペイン

プロダクトの品質を第三者が評価するためには，プロダクトの変更履歴，および，プロダクトの規模や複雑さを表す定量的データが必要となる．本ペインでは，各スナップショットにプロダクトの規模や複雑さを表すメトリクスを紐付けることにより，プロダクトの品質，および，その時系列的変化を評価できるようになる．

[5] 課題ペイン

ソフトウェアの課題が解決されているか否かを評価するためには，課題票の対応状況やソースコードの変更内容に基づいて，課題票とソースコードを紐付けることが求められる．本ペインでは，各スナップショットにおける未対応課題数，および，未対応課題が発見されたソースコードを特定することで，低品質モジュール（課題が存在するソースコードファイル）が生成される要因を評価できるようになる．

2) 解析方式

[1] 検証型解析技術

ソフトウェアの拡張とソフトウェアの未対応課題数の時系列変化を解析することで，低品質モジュールを生成する原因を明らかにし，ソフトウェア品質の内的妥当性の評価を容易にする．具体的には，未対応課題数が増加した原因を，スナップショットデータのソースコードの特性値，及び，ソースコード間の依存関係から見つけ出す．加えて，過去の未対応課題を多く含むソースコード特性の時系列変化を学習することで，将来的に品質が低下するモジュールを評価できるようになる．

また，メトリクス値の時系列的変化を解析し，基準値と照らし合わせることで，ソフトウェア品質の外的妥当性を容易に評価できるようになると考えられる．例えば，各バージョンの複雑度の変化を解析し，大規模ソースコード集合から求めた基準値と照らし合わせる．これを行うことによって，評価対象のソフトウェアが一般的な基準値と比較して，適切な範囲の複雑度であるか，また，いつ基準値を超える複雑度を持つようになったか判断できるようになる．

[2] 探索型解析技術

ソフトウェアを構成するモジュール群のうち，低品質となるモジュールがどの程度あるのかの評価や生成要因の分析が可能となる．例えば，テスト工程に適用すれば，低品質モジュールを重点的にテストしていたかなど，テストのプロセスや戦略の評価が可能となる．

3) 可視化方式

[1] 検証型可視化技術

各スナップショットから計測した未対応課題数の推移と同時にソースコードの特徴量，ソースコード依存関係の変化を可視化することで，プロジェクト全体の変遷を俯瞰的に把握することが可能となる．開発者は，課題が発見されたソースコード（低品質モジュール）の特徴量の変化を分析することで，課題の要因分析を容易に行うことができる．また，ソースコードの依存関係が可視化されることで，低品質モジュールの偏在傾向の解析も容易になる．

[2] 探索型可視化技術

スナップショットは，多次元・多変量データ集合体であり，情報可視化における従来の原則「Overview first, zoom and filter, then details on demand」が必ずしもあてはまらない．本委託研究では，多数の変数間の関係を網羅的に表し，かつ，分析者の認知的負荷の軽減にもつながる，できるだけシンプルな提示方法を実現することで，第三者評価における探索型分析を支援する．

③ 「ソフトウェア品質の第三者評価」の妥当性評価実験に期待される効果

提案する各方式の妥当性を評価実験を通して示すことで，ソフトウェア開発企業における各方式の検討および採用を促進できると考えられる．例えば，メトリクス値の時系列変化および基準値データを用いた評価実験を行うことで，それらをソフトウェア開発企業において利用する際のシナリオを示すことができると考えられる．また，一般的なソフトウェア開発企業では，適用例が全くない方式の提案を行なったとしても，十分な理解を得られ

ることは難しく，直接的に産学連携の流れを作り出すことは難しい．よって，提案内容の適用および適用によって得られる定量的なデータの収集は，ソフトウェア開発企業において本委託研究で提案する方式の利用を促進するために必要であると考えられる．

3.1.2 研究プロセスと成果

(1) 研究プロセス

① 既存の研究・システム・ツール等の調査を行う．調査対象は，ソフトウェア工学分野，特に，ソフトウェア解析，ソーシャル分析，マイニングソフトウェアリポジトリ等に関する学術論文が掲載されている学術論文誌や国際会議予稿集・論文集，同分野の市販およびオープンソースソフトウェアとして公開されているソフトウェアツール等とする．
② ①の結果に基づき，ソフトウェアプロジェクトトモグラフィの概念設計を行う．特に，医療におけるコンピュータ断層撮影，いわゆる，CT（Computed Tomography）を，ソフトウェア品質の第三者評価に適したデータ構造を考える上でのモデルとする．
③ ①および②の結果に基づき，ソフトウェアプロジェクトトモグラフィの中核となるスナップショットの生成，解析，可視化の各方式の設計を行う．
④ 中間報告会において，①～③の成果を報告する．
⑤ ④において評価委員から頂いたコメントに基づいて，②，③の成果に必要な修正を加えた上で，スナップショット生成，解析，可視化のプロトタイプシステムの実装を行う．
⑥ ⑤で実装したプロトタイプシステムを用いて，ソフトウェアプロジェクトトモグラフィによる「ソフトウェア品質の第三者評価」の妥当性を実験的に評価する．
⑦ 成果のとりまとめ，研究の将来展望・フォローアップの検討を行う．

(2)具体的な研究成果の内容

① スナップショットに関する研究成果

スナップショットの詳細設計として，プロジェクト構造モデルの ER 図を開発した（図 3-1-1 参照）．同モデルは，ソフトウェアプロジェクトの全体構造を，5 つの構成要素「要件」，「作業」，「組織」，「プロダクト」，「課題」とその間の関係を表すものである．同モデルを構成するエンティティは次の通り．なお，図 3-1-1 では，各エンティティの属性として最低限必要と考えられるもののみを例示している．

・非依存エンティティ

要件：プロジェクトの構成要素「要件」に対応する．
作業：プロジェクトの構成要素「作業」に対応する．
組織：プロジェクトの構成要素「組織」に対応する．
プロダクト：プロジェクトの構成要素「プロダクト」に対応する．
課題：プロジェクトの構成要素「課題」に対応する．

メンバー：プロジェクトメンバーを表す．

・依存エンティティ

要件操作：「要件」に対して実施された操作を表す．一つの用件に対して複数の操作を対応させることができる．

プロダクト操作：「プロダクト」に対して実施された操作を表す．一つのプロダクトに対して複数の操作を対応させることができる．

課題操作：「課題」に対して実施された操作を表す．一つの課題に対して複数の操作を対応させることができる．

実施：「作業」の実施を表す．一つの作業に対して複数の作業を対応させることがで

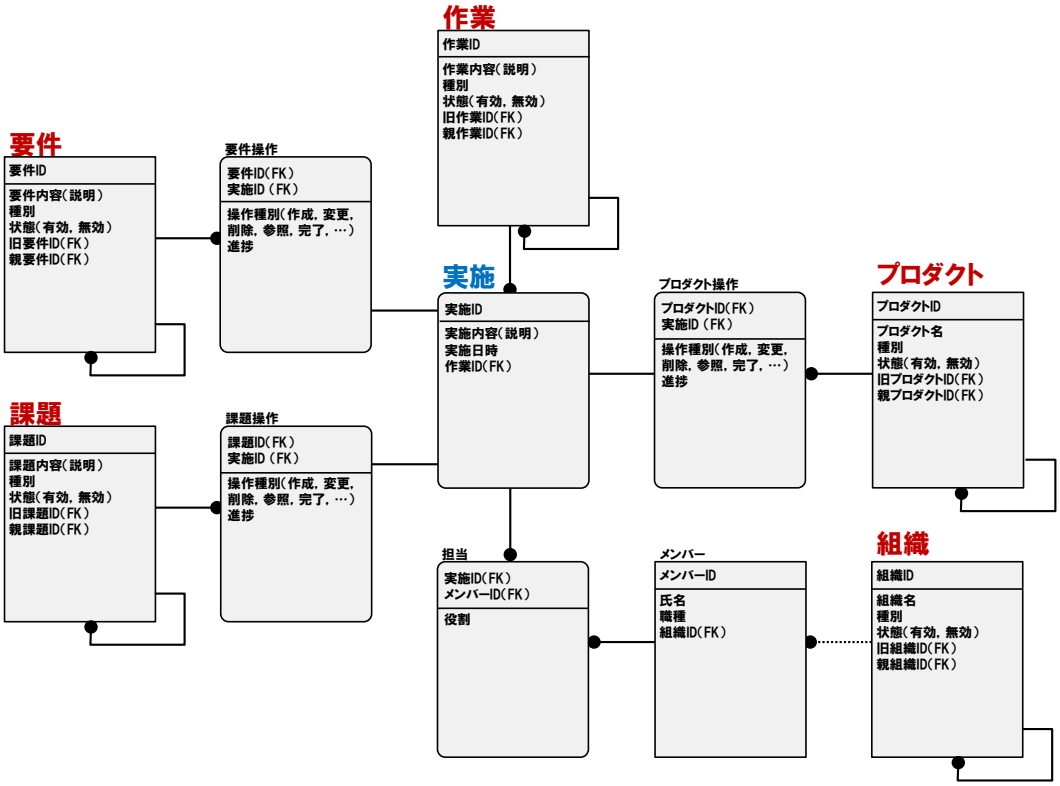


図 3-1-1　スナップショットにおけるプロジェクト構造モデル

きる．各実施には，要件，プロダクト，課題に対して実施された操作が対応付けられている．

担当：特定の役割が割り当てられた「メンバー」を表す．一つの担当に対して複数のメンバーを対応させることができる．担当は，また，「実施」の担当者を表す．一つの実施に対して複数の担当を対応させることができる．

非依存エンティティのうち「メンバー」エンティティを除く 5 つのエンティティ，すなわち，ソフトウェアプロジェクトの 5 つの構成要素に対応するエンティティは，それぞれが階層構造を成すとして自己再帰関係を持ち，親となるエンティティの識別子（ID）を属

性値として持つ．また，それらエンティティの中には，ソフトウェア開発プロジェクトにおいて，一時的に作成される実体やプロジェクトの進行に伴って削除される実体に対応するものも含まれる．しかし，解析や可視化においては，不要となった，あるいは，削除されたものも対象となる場合がある．そこで，ここでは，対応する実体が不要となったり，削除されたりしたエンティティも引き続き存在できるものとし，属性値として「状態（有効，無効）」を持たせている．より詳細な解析や可視化に対応するためには，各エンティティが有効・無効となった日時，あるいは，有効期間といった情報を属性値として追加することも考えられる．

依存エンティティのうち「要件操作」，「プロダクト操作」，「課題操作」は，その名称通り，対応するそれぞれのプロジェクト構成要素（要件，プロダクト，課題）に対する操作を表すと共に，それらをエンティティ「実施」に結びつける役割を果たしている．「担当」も同様の役割を果たしているが，「組織」ではなく別の実体である「メンバー」と「実施」との関係を表す．これは，組織がメンバーで構成されており，（作業の）「実施」を実際に担当するのはメンバーであると考えたためである．

以上の通り，本委託研究で開発した（定義した）プロジェクト構造モデルでは，非依存エンティティ「作業」に依存するエンティティ「実施」によって，プロジェクトの５つの構成要素に対応する５つの非依存エンティティが結びつけられるとしている．これは，「ソフトウェア品質の第三者評価」における解析・可視化の一義的な目標の一つは「評価対象プロジェクトにおいて何が行われたのか」であると考えた結果である．なお，「プロジェクトで行われたこと」とは，非依存エンティティ「作業」そのものではなく，そのインスタンスである．作業のインスタンスを表すものとして依存エンティティ「実施」を導入した．「作業」が中心となるという考えを別のかたちで表すとすれば，「組織（あるいはメンバー）は作業を実施するために存在しているのであり，要件は作業の実施によってプロダクトとして実現され，作業が実施される中で課題が発生・解決される．作業の実施によって，作業そのものはもちろんのこと，プロジェクトの他の構成要素も意味を成す．」ということになる．

なお，スナップショットは，プロジェクト構造モデルに加えて，「一次解析結果」と「プロダクト断片」により構成される．「一次解析結果」は，多くの場合，スカラーやベクトルとして表され，一般的なスプレッドシート上で表現・蓄積可能である．プロジェクト構造モデル上の各エンティティに，「一次解析結果が表現・蓄積されているスプレッドシート等へのポインタ」を格納するための属性を持たせることで，両者の関係を表現することができる．「プロダクト断片」も文字列や図表の集合体（複合体）であり，ファイルとして表現・蓄積可能である．各エンティティに同様の属性を持たせることで，「プロダクト断片」との関係も表現可能である．

プロジェクト構造モデルにより，「ソフトウェアプロジェクトを表現するに十分な情報を含むスナップショットの実現」という当初の到達目標が達成され，また，「ソフトウェアプロジェクトデータの共有や再利用の促進」や「ソフトウェアプロジェクトの解析や可視化に要するコスト低減」という当初期待された効果がもたらされたことは，以降に示す個々の研究成果から明らかである．

② スナップショットの生成・解析・可視化方式に関する研究成果

1) 生成方式

[1] 要件ペイン

開発の初期段階に明らかにしておくべき非機能要件が,提案依頼書(RFP)や要件定義書に十分に記述されているかどうかを第三者が評価することを目的とする．その手段として,非機能要件の記述の網羅性や明確さを数値化（スナップショット化）するための非機能要件評価表を作成した．非機能要件を評価する非機能要件評価表を生成するプロセスを図3-1-2 に示す.

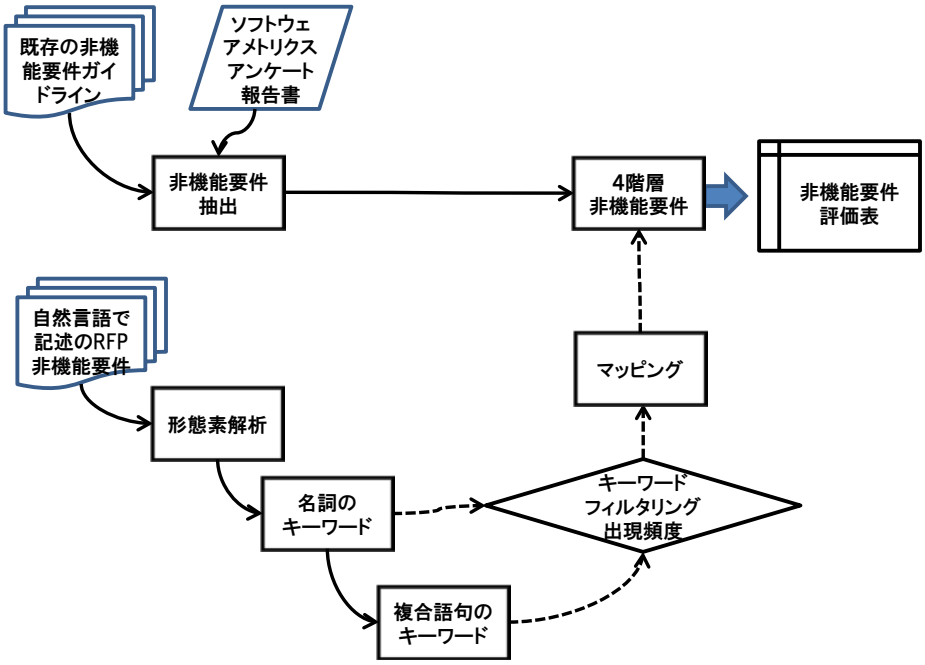


図 3-1-2　非機能要件特性表の作成プロセス

RFP の内容を評価するために， RFP をテキストマイニングにより形態素解析し，必要な非機能要件キーワードを抽出し，その出現頻度によりフィルタリングを実施した．次に、既存の非機能要件ガイドラインとマッピングし，抽象化した上位概念としての非機能要件特性による階層を定義した.また,ユーザに対するソフトウェアメトリクス調査結果から,ユーザが重要と考えている非機能要件メトリクスを回答数によりフィルタリングし，回答数が多い非機能要件メトリクスを利用して，重みづけを行い，評価結果を容易に把握できるようにした．この結果,非機能要件キーワードを最下層とし,非機能要件特性の小特性,中特性及び大特性をその上位階層とした表を構成した．RFP に記述された個別の非機能要件を評価するために 5 段階評価手法を採用した．これらの一連の手順を経て,提案する「非機能要件評価表」を生成した．RFP を評価した結果，記述が不足している非機能要件を特定したり，同じドメインの他の RFP との比較することが可能であることを確認した．

非機能要件評価表の構築の手順は次の通りである．

1. 既存の非機能要件から RFP に記述すべき非機能要件を抽出する．

2. RFP に記述されている非機能要件を形態素解析し，キーワードを抽出する.
3. 評価結果の理解を容易にするために非機能要件のカテゴライズと抽象化をおこなう.
4. 非機能要件キーワードと非機能要件特性をマッピングする.
以下，各ステップの詳細を述べる.

ステップ 1

最初に, RFP に記述されている非機能要件メトリクスを選択するにあたって, 200 以上の非機能要件メトリクスを網羅的に記載しているガイドライン[JUAS]をベースとした. ただし，本ガイドラインには多種多様な非機能要件が含まれているが，ソフトウェア開発終了後の保守・運用に関する非機能要件については必ずしも網羅されていない. RFP の段階において盛り込むべき保守･運用に関する要件を記載したガイドラインとして,文献[Nikkei]や[METI]がある. いずれも，ユーザとベンダ間でソフトウェア開発契約を行うに当たって重要となる，サービスレベルに関する合意（SLA; service level agreement）に必要な要件が示されており，保守・運用に関する非機能要件も数多く盛り込まれている. 本論文では，これらのガイドラインに記載されている非機能要件をベースとして，RFP で定義すべき非機能要件の絞り込みを行う.

ユーザ視点から様々な要件の重要性を調査した資料として，経済産業省ソフトウェアメトリクス高度化プロジェクトにおけるプロダクト品質メトリクス WG の活動報告[MRI]が公開されている. この調査資料では，非機能要件と関連づけられる様々なシステム特性について,多数のユーザ企業へのアンケートに基づくランキングが示されている.本論文では，このアンケート結果に基づいて，各非機能要件の絞り込みを行った. 具体的には，以下の基準により非機能要件メトリクスを抽出した.

- 実際に使用あるいは使用したい回答数の多い非機能要件メトリクスを選択
- ソフトウェア開発段階で管理するための非機能要件メトリクスを除外
- 契約前の要求分析段階で定義することが困難な非機能要件メトリクスを除外
- ユーザ企業内部で評価あるいは管理するための非機能要件メトリクスを除外
- ハードウェア機器管理のための非機能要件メトリクスを除外
- システムテスト以後に必要であると回答のあった非機能要件メトリクスを選択

結果として 38 個の非機能要件メトリクスを抽出した.アンケート対象となっていない非機能要件については，文献[Nikkei][METI]に基づいて，サービスレベルに関する合意に必要な非機能要件を 17 個，合計で，55 個の非機能要件を選定した.

ステップ 2

RFP の非機能要件が記述されているテキスト文を形態素解析し，解析結果の名詞および複合語の中から非機能要件キーワードを抽出した. 次に，抽出された非機能要件キーワードに絞って再び形態素解析を行い，その出現頻度でフィルタリングして採用する非機能要件キーワードを決定した.

ステップ 3

このステップでは，ステップ 1 で抽出した非機能要件メトリクスとガイドラインを参照し，その上位層としての小特性，中特性及び大特性の 3 層に構造化を定義した．次に，抽出された非機能要件メトリクスによる RFP における記述の十分性を評価するにあたって，非機能要件メトリクスを 3 つのタイプに分類した（表 3-1-1）．タイプ 1 は定性的に表現され，かつ表現の曖昧さが大きい非機能要件メトリクスであり，タイプ 2 は定性的に表現され，かつ表現の曖昧さが少ない非機能要件メトリクスであり，タイプ 3 は定量的に表現される非機能要件メトリクスである．例えば、アクセス制御メトリクスは様々な記述表現があることから 5 段階評価であるタイプ 1 とし，保守タイプは定義が明確であることから 3 段階評価であるタイプ 2 とし，応答時間は数値として記述され 2 値評価が可能なことからタイプ 1 とした．

表 3-1-1　RFP 評価尺度のタイプと評価点

ポイント	タイプ 1	タイプ 2	タイプ 3
4	明確	明確	定量的に記載
3	やや明確	N/A	N/A
2	やや不明確	不明確	N/A
1	不明確	N/A	N/A
0	記載無し	記載無し	記載なし

非機能要件メトリクスによる RFP 評価を行うにあたって，各非機能要件メトリクスの重要性には差異があることが考えられる．したがって，RFP 評価を定量的に把握するには，各非機能要件メトリクスの重要性に対して重みを考慮することが必要となる．そこで，ソフトウェアメトリクスに対するユーザ企業への調査結果をもとに，各非機能要件メトリクスの回答数による重み付を行った．各非機能要件メトリクスの 5 段階評価した結果への重み付けにより得られる評価点は，どの非機能要件が十分に記述されているか，あるいは不十分な記述であるかをより明確にかつ，定量的に理解する上で有効である．また，定量化された中特性及び大特性を他のプロジェクトと比較することより，相対的な評価を容易にし，問題点を把握することが可能となる．以下に重み付けから評価ポイントまでの計算式を示す．

プロジェクト k の各非機能要件メトリクスの評価得点は，i 番目の非機能要件メトリクス mi に対する 5 段階評価とその重み付により下記の式にて算出する．

NFR 評価得点 k（mi）= mi の 5 段階評価 mi * mi の重み

そこで，i 番目の中特性 Mi に対する評価ポイントは下記の式にて算出する．

中特評価得点 k(Mi) = $\sum_{m \in M_i} NFR$評価得点(m) / |Mi|

ここで， m∈Mi は中特性 Mi に属する非機能要件メトリクスであり，|Mi| は中特性 Mi に属する非機能要件メトリクス項目数である．同様に、i 番目の大特性 Hi は以下の式にて算出する．

大特性評価得点 k(Hi) = $\sum_{m \in H_i}$ 中特性評価得点(m) / |Hi|

次に，ドメイングループ単位での比較評価を行うために，i 番目の中特性評価得点及び i 番目の大特性評価得点の平均を下記の算出式にて算出した．

ドメイン中評価得点 D(Mi) = $\sum_{j\in D}$ 中特性評価得点$_j(Mi)$ / |D|

ドメイン大評価得点 D(Hi) = $\sum_{j\in D}$ 大特性$_j(Hi)$ / |D|

ここで，j∈D ドメイン D に属するプロジェクト数である．

ステップ 4

このステップにて，階層構造化された非機能要件特性とキーワードをマッピングすることにより，「非機能要件評価表」生成した．表 3-1-2 に非機能要件キーワード評価表の一部を示す．提案方法では，要求が記載された文書を非機能要件キーワードに基づいて小特性ごとに評価点を付け，その上位である中特性および大特性の評価点を計算することにより評価する．

非機能要件の記述の明確さの評価にあたっては，各要件とマッピングされたキーワードに着目し，RFP の中の各キーワードが出現している部位を調べる必要がある．ただし，キーワード間には依存関係があり，また，一つのキーワードが複数の非機能要件の説明に現れる場合もあるため，評価者は，キーワード間の関係を考慮しながら評価を行う必要がある．また，RFP の記述を改善するためには，不足しているキーワードを補うことも必要である．

表 3-1-2　非機能要件評価表

大特性	中特性	小特性	非機能要件キーワード		
システム運用の評価要件	操作容易特性	オペレーション指標	操作性	操作マニュアル	システム操作
			操作方法	システム運用手順書	画面遷移
			操作環境	操作を容易にする	操作手順
			操作の手順	入力ミス	操作ガイド
			簡単な操作	直感的な操作	操作が容易
			簡単に入力操作	システム操作方法	容易に操作
			運用手順書	直感的に操作	操作説明
			簡易に操作可能	運用手順	操作の利便性
			オンラインマニュアル	容易な操作	操作手順書
			操作説明書	操作が容易	
	稼働品質特性	応答性指標	平均応答	ハードディスク応答性能	平均処理応答
			ネットワーク転送容量	転送応答性	最小レスポンス
			安定的レスポンス	画面レスポンス	ターンアラウンド
			最大スループット	VPNスループット	応答性
			ハードディスク容量	メモリ使用率	応答時間
			最小応答	平均読み出し遅延	秒以内
			主記憶容量	システム応答速度	秒程度
			オンライン応答時間	秒以下	データ量
			レスポンスタイム	スループット	
		システム性能指標	ネットワークの性能	ネットワーク使用率	MPU使用率
			サーバの性能	ディスクIO負荷率	性能監視機能
			CPU使用率	%以下	SPEC
			端末性能	ネットワーク性能	演算性能
			サーバ性能	ハードウェア性能	ソフトウェア性能
			オンライン処理性能	性能目標値	バッチ処理性能
			CPU	CPU	処理性能
			十分な性能	同時接続数	システムの性能
			システム性能	以上の機能・性能	同等以上の性能
			総合演算性能	アクセス速度	以上の性能
			中央演算処理装置	計算性能	処理速度
			CPU負荷率		
		負荷バランス指標	負荷監視	CPU負荷	回線負荷
			負荷計測	最大負荷時	負荷分散性能
			ロードバランシング	負荷分散	負荷率
			負荷低減	ピーク時	ネットワーク負荷
			運用負荷	負荷予測	負荷許容範囲
			高負荷	負荷運用	最小サーバ負荷
		稼働品質指標	平均稼働率	アクセス量	アクセス頻度
			稼働率	%以上	サービス稼働率
			システム稼働率	正常稼働	安定システム
			稼動状況	安定稼働実績	稼動実績
			稼働期間	正常に稼働	24時間365日稼働
			安定稼働	システム稼働管理	ダウンタイム
			安定運用	運用スケジュール	稼動安定性
			業務稼働率	年間稼働率	システム稼動情報
			システム稼働状況	稼動実績	通信速度
			稼働監視	稼働回数	連続稼働
			トランザクション数	%以下	ネットワーク管理状況
			トラフィック量	パフォーマンス管理	連続運転

そこで,キーワード間の関係を明らかにして評価や改善を支援するために,自己組織化マップを要件ペインに付加することとした．まず，自然言語で記述されている RFP を形態素解析することにより，非機能要件メトリクスに関連するキーワードを抽出した．抽出されたキーワードを非機能要件評価表の該当する非機能要件にマッピングし，これらのキーワードを KH Corder[KHC]の「自己組織化マップツール」を用いて各非機能要件に属するキーワード間の関連性を可視化した．図 3-1-3 に非機能要件自己組織化マップの例を示す．この自己組織化マップにおけるノードのクラスタ化を用いることにより，色分けされたマップが得られる．これにより，評価対象の RFP に現れるキーワードと近い関係のあるキーワードを知ることができ，不足しているキーワードを補うなど，RFP の改善に役立てることが期待される．

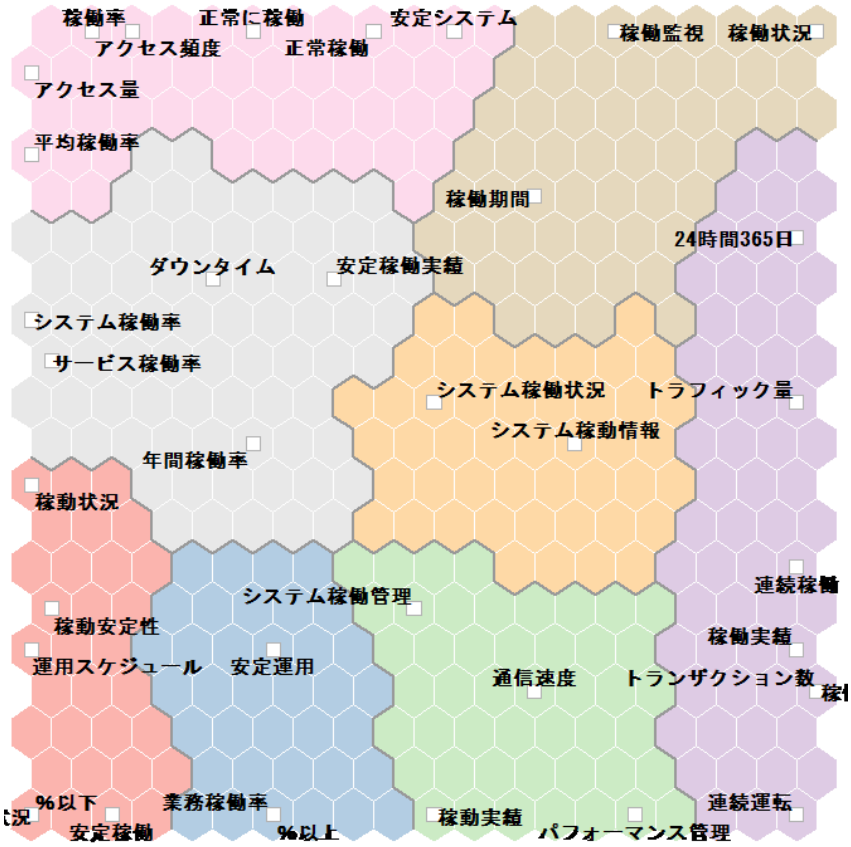


図 3-1-3　非機能要件のキーワードの自己組織化マップの例

[2] 作業ペイン

パソコン上のアプリケーションの実行履歴を記録するツールである TaskPit を用いて，開発者の実行中のアプリケーションの履歴を計測し，タスク実績シートとして記録する[Taskpit]．本委託研究では，各タスクは，それぞれ異なるアプリケーションやウィンドウ上での作業であると捉える．各アプリケーションやウィンドウ上での作業時間（ユーザが何らかの入力を与えていた期間）を記録することで，各タスクに費やした時間を計測でき

る．これによって，例えば，メーラーを使った「メールの読み書き」，Word を使った「仕様書作成」，Eclipse 上での「コーディング」といった粒度でのタスクの記録が可能となる．また，TaskPit により，各タスクの作業量は，各アプリケーションやウィンドウに対するキーストロークやマウスの操作量（回数）として記録される．さらに，各タスクの成果物は，特定のディレクトリ下にファイルとして出力されると捉えられる．ファイルサイズの増減を一定時間ごとに計測することで，各タスクの成果物の量の推移を計測できる．

本委託研究では，実際のソフトウェア開発作業において，アプリケーションおよびウィンドウ名とのマッピングを決定した．図 3-1-4 にタスクとアプリケーションのマッピングの例を示す．

```
サーバ管理= ffftp.exe|WINSCP.exe
ドキュメント作成= EmEditor.exe|TeraPad.exe|notepad.exe|
                  WINWORD.EXE|Acrobat.exe|acrodist.exe
ファイル管理=    explorer.exe|cse.exe
ブラウジング=    iexplore.exe|firefox.exe|Safari.exe|chrome.exe
プログラミング  = eclipse.exe|netbeans.exe|devenv.exe
メール =         thunderbird.exe|Outlook.exe|iexplore.exe:Gmail
```

図 3-1-4　タスクとアプリケーションのマッピング

同様に，ソフトウェア開発作業における成果物名と，それに対応する作業用ディレクトリとファイル拡張子のマッピングについても図 3-1-5 に例を示す．

タスク実績シートの例を，表 3-1-3 に示す．表に示されるように，各タスクの実施時間，マウスのクリック数，キー入力数（打鍵数）が記録される．また，成果物ごとのファイルサイズとファイル数の増減が記録される．これにより，タスクの実施状況とその結果について，客観的なデータを得ることが可能となった．

```
仕様書= "C:¥Documents and Settings¥ユーザ名¥デスクトップ¥開発¥仕様書
    ","doc,tex,txt"
設計書= "C:¥Documents and Settings¥ユーザ名¥デスクトップ¥開発¥設計書
    ","doc,tex,txt"
コード= "C:¥Documents and Settings¥ユーザ名¥デスクトップ¥開発¥コード
    ","c,cpp,java"
```

図 3-1-5　タスクとアプリケーションのマッピング

表 3-1-3　タスク実績シートの例

タスク名	時間(時:分:秒)	クリック数	打鍵数
サーバ管理	0:02:15	35	66
ドキュメント作成	0:18:10	86	385
ファイル管理	0:15:11	102	297
ブラウジング	0:58:32	3052	433
プログラミング	2:28:05	1857	25421
メール	0:25:55	202	3211
登録外	0:08:20	35	79
成果物	ファイルサイズ	ファイル数	
仕様書	+355KB	+1	
設計書	0	0	
コード	+594KB	+5	

[3] 組織ペイン

本委託研究メンバーが開発してきた「アウェアネス支援システム ACTION」，および，ジョージア工科大学が開発した「ソーシャルネットワーク可視化ツール Vizter」を用いて，開発チームの組織構造を表すペインを作成した．また，木構造として従来表現される開発開始当時のチームの組織構造と，実際の組織構造との差を容易に把握できることを確認した．具体的には，オープンソースプロジェクトの組織構造の経時変化を可視化するとともに，3 種類の中心性計測指標を用いて組織内のコミュニケーションの特徴を分析した．組織構造はプロジェクトのコミュニケーション履歴データ(メーリングリストにおける議論)から抽出したものである．以下に，分析対象プロジェクトの概要を示す．

・Apache HTTP Server

現在，世界一のシェアを誇る Web サーバ（HTTP Server）ソフトウェアである．パフォーマンスの高さや拡張性の高さ，使用が無制限であることなどから広く利用されている．また欠陥の修正パッチを継続的に多数リリースしているため，非常に高品質なソフトウェアとして広く認知されている．

・GIMP

GIMP とは GNU Image Manipulation Program の略称であり，画像の編集や加工を行うためのソフトウェアである．無料でありながら，有料かつ高額な画像編集ソフトウェアと同等の機能を有しており，多数のユーザから支持されている．

・Netscape Browser

Web ブラウザソフトウェアであり，1996 年ごろの最盛期には市場の 8 割を占めていた．Netscape コミュニティは，Internet Explorer のシェア拡大に対抗するため Raymond らの研究を受けて 1998 年にオープンソース化を実施したが，シェアを取り返すことができず衰退していった．現在は Mozilla コミュニティが Netscape の資産を引き継ぎ，Firefox などのソフトウェアを開発している．

また，3 種類の各中心性指標の定義は次の通りである．

・次数中心性

ノード vi の次数中心性 Cdegree(vi)は，ネットワーク内のノードが取りうる最大の次数によって vi の次数を正規化した値である．

$$Cdegree(vi) = deg(vi)/(n-1)$$

ここで，n はネットワーク内のノード数を，deg(vi)は vi の次数を表す．Cdegree(vi)は最小 0 から最大 1 までの値を取り，値が 1 に近いほど次数中心性が高い．次数中心性が高いノードほど，他のノードと隣接しているエッジの数が多いノードである．OSS コミュニティにおいては次数中心性が高い参加者を，多くの参加者へ情報を発信／受信している参加者とみなすことができる．

・媒介中心性

ノード vi の媒介中心性 Cbetweenness(vi)は，ネットワーク中の他の任意の 2 つノード vj，vk（$1 \leqq j < k \leqq n, j \neq i, k \neq i$）間の最短経路にノード vi が含まれる割合である．

$$Cbetweenness(vi) = \Sigma_{1 \leqq j < k \leqq n, j \neq i, k \neq i} (vi\ を含む\ vj\ から\ vk\ への最短経路の総数) / \Sigma_{j<k} (vj\ から\ vk\ への最短経路の総数)$$

ここで，n はネットワーク内のノード数を表す．Cbetweenness(vi)は最小 0 から最大 1 までの値を取り，値が 1 に近いほど媒介中心性が高い．媒介中心性が高いノードほど，他のノードの仲介者としての役割を果たすノードである．OSS コミュニティにおいて媒介中心性が高い参加者ほど，他の参加者同士を繋ぎ合わせる参加者である．つまり，媒介中心性が高い参加者が突然いなくなると，OSS コミュニティでコミュニケーションが円滑に行われなくなると考えられる．

・近接中心性

ノード vi の媒介中心性 Ccloseness(vi)は，ノード vi からネットワーク中の他の任意ノードへ vj の最短経路長の総和の理論上の最小値（n-1）（n はネットワーク内のノード数）と実際の最短経路長の総和の比である．

$$Ccloseness(vi) = (n-1) / \Sigma_{1 \leqq j \leqq n, j \neq i} (vi\ から\ vj\ への最短経路長)$$

ここで，n はネットワーク内のノード数を表す．なお，vi から vj へ到達不可能な場合，文献[Matsuo]と同様に，最短経路長は n とする．Ccloseness(vi)は最小 0 から最大 1 までの値を取り，値が 1 に近いほど近接中心性が高い．近接中心性が高いノードほど，他のノードへ近い距離で到達可能なノードと言える． OSS コミュニティにおいて近接中心性が高い参加者ほど，界隈に参加者が多い（少ない距離で他の参加者へ到達できる）参加者である．つまり，次数中心性が高い参加者が，（OSS コミュニティ全体ではなく）自身と隣接する参加者を対象とする一方で，近接中心性が高い参加者は，OSS コミュニティ全体に対して効率よく情報を伝達しているとみなすことができる．

[4] プロダクトペイン

ソフトウェア品質の外的妥当性評価のため，次の 2 種類のメトリクスに対して，それぞれの基準値を作成した．

A) Understand より得られるメトリクス

B) CCFinder を用いて計測するコードクローン含有率

メトリクス基準値はUCI Source Code Data Sets[UCI]（以下UCI Datasetsと表記する）を基に作成した．UCI Datasetsの概要を表3-1-4に示す．ただし，基準値の作成においてはツールやメトリクスの性質を考慮してUCI Datasetsの一部プロジェクト，ファイルあるいはメソッドを除外している．基準値作成の対象についての概要を表 3-1-5に示す．

表3-1-4　UCI Datasetsの概要[Ossher]

Originating Repository	Apache	Java.net	Google Code	Sourceforge	Other	Total
プロジェクト数	84	3,412	5,361	9,969	2	18,828
ファイル数	559,140	289,310	402,070	1,983,044	7,346	3,237,910

表3-1-5　基準値作成対象の概要

メトリクスの種類	Understand	クローン含有率
対象データの規模	10,087プロジェクト	12,191プロジェクト
備考	LOCが1000未満のプロジェクトを除外	除外プロジェクト無し

基準値はメトリクス値を正常値とみなす上下限値とする．基準値作成対象のすべてのデータから計測したメトリクス値の集合をMとしたとき，基準値を次式で定義する．

上限: min(q3 + 1.5 * IQR, max(M))

下限: max(q1 − 1.5 * IQR, min(M))

IQR = q3 − q1

ここで，q3およびq1はそれぞれメトリクス集合Mの第三四分位および第1四分位であり，関数min()は引数の最小値を，max()は引数の最大値を返す．上限より大きいあるいは下限未満のメトリクス値を外れ値とする．ただし，利用する基準値（上限，下限のいずれかあるいは両方）はメトリクスの性質に応じて適宜選択されるべきである．

UCI Datasetsを基に作成した基準値の一覧を表 3-1-6に示す．各メトリクスの概要については公式サイトの解説ページ[Higo][Kamiya]に詳しい．また，メトリクス名に「Count」を含むものについてはLOC（Understandにおけるメトリクス名はCountLineCode）で除した値についても基準値を作成した（表3-1-7）．これは，当該メトリクスがその性質上，最も一般的なメトリクスであるLOCと高い相関を示したためである．

表 3-1-6　Understand より得られるメトリクスの基準値一覧

メトリクス名	基準値		メトリクス名	基準値	
	上限	下限		上限	下限
AvgCyclomatic	3.5	1	CountLineCode	33259	1000
AvgCyclomaticModified	3.5	1	CountLineCodeDecl	11373.5	100
AvgCyclomaticStrict	3.5	1	CountLineCodeExe	15193.75	156
AvgEssential	1	1	CountLineComment	16543.5	0
AvgLine	358	8	CountSemicolon	18089.75	252
AvgLineBlank	46.5	0	CountStmt	23731.5	294
AvgLineCode	215	6	CountStmtDecl	10518.5	108
AvgLineComment	118.5	0	CountStmtExe	13369	134
CountDeclClass	525.5	1	Cyclomatic	6178	32
CountDeclClassMethod	234.5	0	CyclomaticModified	5938.5	32
CountDeclClassVariable	416	0	CyclomaticStrict	6570	32
CountDeclFile	370.75	1	Essential	3812.5	18
CountDeclFunction	3162.75	18	MaxCyclomatic	101.5	1
CountDeclInstanceMethod	2526	0	MaxCyclomaticModified	85.5	1
CountDeclInstanceVariable	972	0	MaxCyclomaticStrict	118.5	1
CountDeclMethod	2768	18	MaxInheritanceTree	11.5	1
CountDeclMethodAll	31361	31	MaxNesting	14	0
CountDeclMethodDefault	62.5	0	RatioCommentToCode	1.225	0
CountDeclMethodPrivate	252.5	0	SumCyclomatic	6685.5	32
CountDeclMethodProtected	149	0	SumCyclomaticModified	6443.25	32
CountDeclMethodPublic	2243.5	4	SumCyclomaticStrict	7109	32
CountLine	57788.75	1157	SumEssential	4168	18
CountLineBlank	7820.75	0	-	-	-

表 3-1-7　LOC で正規化した一部メトリクスの基準値

メトリクス名	基準値		メトリクス名	基準値	
	上限	下限		上限	下限
CountDeclClass	3.79E-02	2.83E-04	CountLineBlank	4.31E-01	4.67E-02
CountDeclClassMethod	2.05E-02	0	CountLineCode	1	1
CountDeclClassVariable	3.30E-02	0	CountLineCodeDecl	5.49E-01	1.50E-01
CountDeclFile	2.94E-02	4.14E-05	CountLineCodeExe	6.67E-01	2.22E-01
CountDeclFunction	1.73E-01	1.99E-02	CountLineComment	1.24E+00	0
CountDeclInstanceMethod	1.52E-01	8.23E-03	CountPath	1.10E+02	-6.57E+01
CountDeclInstanceVariable	7.23E-02	0	CountSemicolon	7.13E-01	3.79E-01
CountDeclMethod	1.61E-01	1.49E-02	CountStmt	9.00E-01	5.49E-01
CountDeclMethodAll	2.89E+00	2.31E-02	CountStmtDecl	5.14E-01	1.42E-01
CountDeclMethodDefault	6.42E-03	0	CountStmtExe	5.90E-01	1.90E-01
CountDeclMethodPrivate	2.29E-02	0	Cyclomatic	2.67E-01	9.69E-02
CountDeclMethodProtected	1.51E-02	0	CyclomaticModified	2.56E-01	9.77E-02
CountDeclMethodPublic	1.42E-01	1.17E-03	CyclomaticStrict	2.86E-01	9.83E-02
CountLine	2.57E+00	1.01E+00	Essential	1.93E-01	3.63E-02

UCI Datasets を用いてプロジェクトにおけるコードクローン含有率の基準値を求めた．UCI Datasets のコードクローン含有率のヒストグラムと箱ひげ図を図 3-1-6 に示す．上限は 0.6972，第三四分位数は 0.3582 であった．

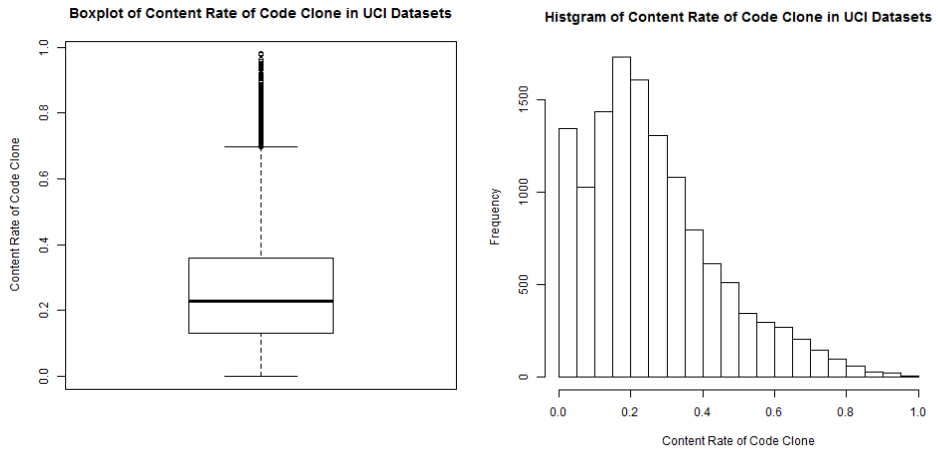


図 3-1-6　UCI Datasets のコードクローン含有率

[5] 課題ペイン

課題管理システムが管理する課題表に基づき作成した．一次解析データとして，未対応課題数，および，未対応課題が残存するソースコードとの紐付けを行った．これにより，各スナップショットにおける低品質モジュールの発見が可能となった．ソフトウェア開発中に混入した欠陥を除去するために修正されたソースコードを特定には，Sliwerski らが提案した SZZ アルゴリズム［Śliwerski]を利用した．SZZ アルゴリズムは，版管理システムに記録されたコミットログとバグ管理システムに記録されたテキスト情報を用いる.SZZ アルゴリズムの概略は次の通り．

手順 1: コミットログメッセージに各バグ票に付された ID と「ソースコードの修正」を示すキーワード（例えば，fixed や closed など）が記載されているか否かを確認する．含まれていない場合は，以降の手順をスキップし，次のコミットログメッセージを分析する．

手順 2: 手順 1 のコミットログメッセージに記載されていたバグ ID の修正進捗状況をバグ管理システムで確認し，既に修正済みか否かを確認する．修正が完了している場合，当該コミットログに対応するコミットが発生した時点で，そのバグ（欠陥）が修正されたとする．修正が完了していない場合は，以降の手順をスキップし，次のコミットログメッセージを分析する．

手順 3: 手順 2 で確認された「バグ修正時」の直前のコミットにおいて欠陥が混入したとする．

本手法を用いて「低品質モジュール可視化システム」を試作した．詳細は，本項「3) 可視化方式 ［1］検証型可視化技術 」にて後述する．

2) 解析方式

[1] 検証型解析技術

静的解析ツール「Understand」を用いて，スナップショットに含まれるソースコードの特性値，および，ソースコード間の依存関係を計測し，未対応課題数を多く含むスナップショットにおいて低品質なモジュール（リリース後に欠陥が発見されるモジュール（ソースコードファイル））が生成された要因を解析する技術を開発した．ソフトウェア開発では，過去の開発を振り返り，将来のプロジェクトを成功に導くための教訓を得ることが重要である．ただし，数多くの開発データから教訓を導き出すことは容易ではない．ソフトウェアリポジトリマイニングの分野でも，開発中に発見された課題の偏在傾向の評価[Antoniol][Geipel]，および，課題発生原因の把握[Livshits][Pan]を支援する研究が数多く発表されている．本技術は，同一プロジェクトで生成されたスナップショットを対象としたものであり，ソフトウェア品質の内的妥当性（Internal Validity）を解析する技術と位置づけることができる．具体的な解析手法は，次の通りである．

手順 1: 一定間隔のスナップショットを取得し，各スナップショットの未対応課題数を計測する．ここで，未対応課題数とは，課題管理システムに登録されている課題票の中で解決されていない課題票数とする．

手順 2: 未対応課題数が多いスナップショットにおけるソースコード群の中から低品質モジュールを特定する．低品質モジュールの特定には SZZ アルゴリズムを用いる．

手順 3: 低品質モジュールが生成された要因を解析するため，低品質モジュールと因果関係を持つモジュール，および，低品質モジュールのソースコード特性値を計測する．

[2] 探索型解析技術

低品質モジュールを，スナップショットを利用して，ソフトウェア開発プロジェクトの初期に予測（判別）する技術を開発した．

開発した技術では，まず，評価対象ソフトウェアの過去バージョンのスナップショットに基づいて「低品質モジュール判別モデル」を構築する．同モデルにおける説明変数は「開発終了 N ヵ月前のスナップショットに含まれる各モジュールの特性値」であり，目的変数は「それら各モジュールにおける，開発終了時点での欠陥の有無」である．構築したモデルに，「評価対象ソフトウェアの現バージョンのスナップショットに含まれる各モジュールの特性値」を入力として与えることで，それらモジュールが（開発終了時に）低品質モジュールとなるかどうかを予測することができる（図 3-1-7 参照）．

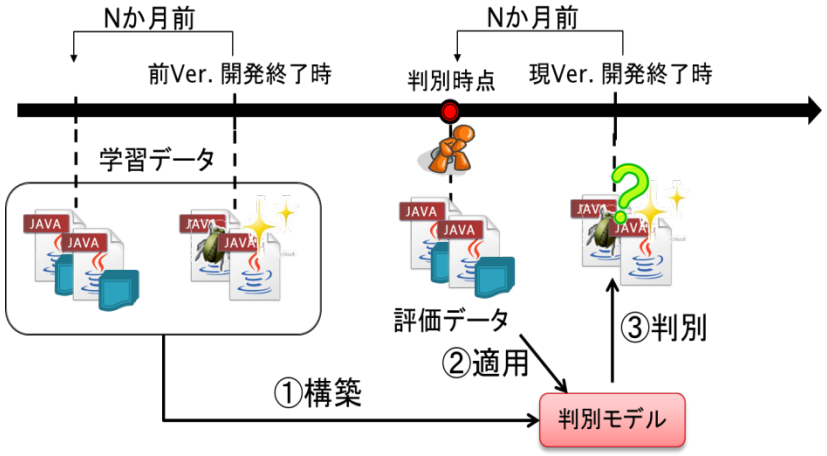


図 3-1-7 低品質モジュール予測

3) 可視化方式

[1] 検証型可視化技術

低品質モジュールを可視化するシステムを試作した．開発中に発見された課題の偏在傾向や課題の発生原因を把握するための研究は，これまでにも行われている[Beyer]．しかし，開発の進行とともに変化するソースコードの依存関係とソースコードの特徴量の時間推移を可視化するためのシステムは実現されていない．試作したシステムに対する要件は次の通りである．

・各スナップショットの未対応課題数を計測する.

ソフトウェアの品質を評価するために発見課題数が利用されることが多い. 開発者は, 課題管理システムに登録されている課題の進捗状況を確認し, 未対応課題数を計測する. 特に課題が多く発見されている時期は重点的に分析することが求められる.

・低品質モジュールを発見する.

開発者は, 課題対応のためにソースコードを変更する際, 課題票の ID などのコメントを手動で付与する. しかし, 全ての変更に ID が付されているわけではない. 低品質モジュールを正確に特定するためには, 課題票と課題対応のために変更されたソースコードを正確に紐付ける必要がある.

・低品質モジュールが生成された要因を分析する.

低品質モジュールの原因を把握するため, 低品質モジュールの特性とモジュール間の依存関係の時間変化の提示, および, モジュールの特性値の計測を行う必要がある.

試作システムの構成を図 3-1-8 に示す. 本システムは, 静的解析ツール「Understand」を用いてスナップショットの解析を行う Source Files Dependency Analyzer と Metics Analyzer, SZZ アルゴリズムを用いて低品質モジュールを特定する Low Quality Module Analyzer, ソースコードの依存関係, 及び, 特性値, 低品質モジュールの出力結果を開発者に提示する User Interface から構成されている. 各構成要素の概要は次の通り.

・Source Codes Dependency Analyzer

各スナップショットにおけるソースコードを Understand に入力し, ソースコード間の呼び出し関係を出力する.繰り返し行われるソースコードの追加/変更によって呼び出し関係のあるソースコードが同時に低品質モジュールとなる可能性が高い. 従って, 呼び出し関係を分析することは, 低品質モジュールが生成された要因を見つける一つの手段になる.

・Metrics Analyzer

各スナップショットにおけるソースコードを Understand に入力し,各ソースコードの特性値を出力する. 繰り返し行われるソースコードの追加/変更によって可読性が悪くなり, 低品質モジュールを生成する可能性が高くなる. 従って, ソースコードの特性値を分析することは, 低品質モジュールが生成された要因を見つける一つの手段になる.

・Low Quality Module Analyzer

各スナップショットにおけるソースコード, 及び, 課題票を用いて, 課題修正のために変更されたソースコード, 及び, 修正期間を分析する. 課題修正のために変更されたソースコードを SZZ アルゴリズム[Śliwerski]を用いて特定する.

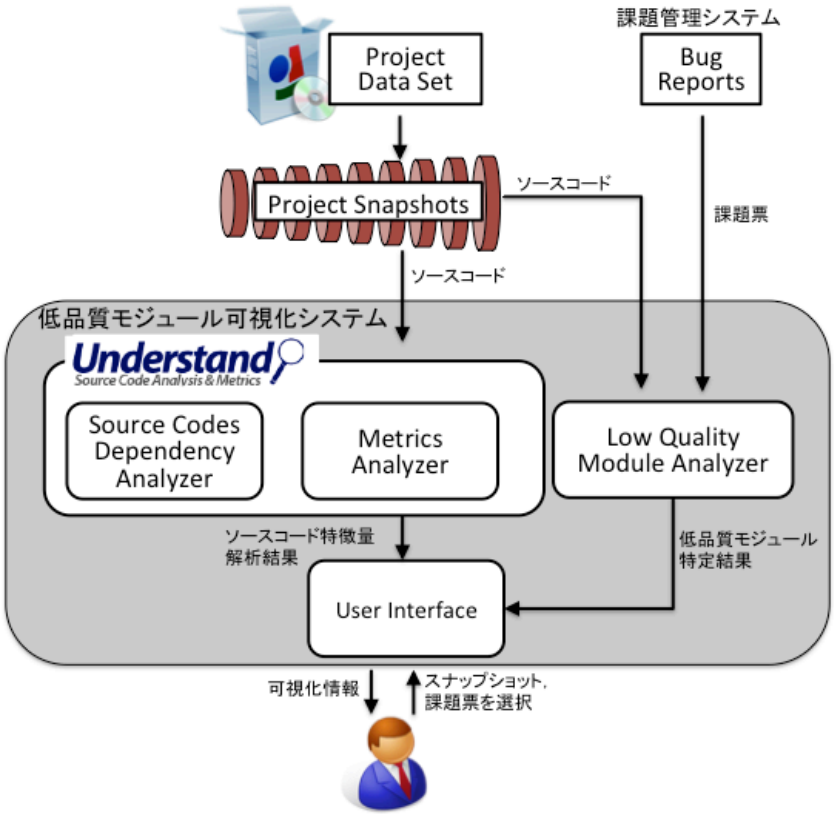


図 3-1-8　低品質モジュール可視化システムの構成

・User Interface

次に示す 5 つの View によって構成されている（図 3-1-9 参照）．

1. Bug Count View: 評価対象スナップショット中の未対応課題数が提示される．開発者がプロジェクトに報告された課題の処理状況を俯瞰的に把握することが可能である．本委託研究では，低品質モジュールを見つけ出すために使用する．
2. Time bar: タイムバーをスライドさせることで，スナップショットを変更する．開発者が着目したモジュールの変化を分析することが可能である．
3. Bugs List View: 課題管理システムに登録されている課題票の中で，Bug Count view で分析対象となったスナップショットの未対応課題が提示される．開発者が着目したスナップショットにおいて，修正中の不具合の種類を把握することが可能である．
4. Bug Prediction Locator View: 分析対象となったスナップショットのモジュールをノードとし，モジュール間の依存関係をエッジとして表示され，選択した課題に関連するソースコードにズームインされる．各ノードの色は，Metricsl List View で選択した特徴量に従って異なる．課題の依存関係とソースコードの特徴量に基づき，低品質モジュールである原因を分析することが可能である．

5. Metrics List View: 静的解析ソフト「Understand」によって計測可能なメトリクスを提示する.開発者は，用意された X 種類の特徴量を変えて,低品質モジュールである原因を分析することが可能である.

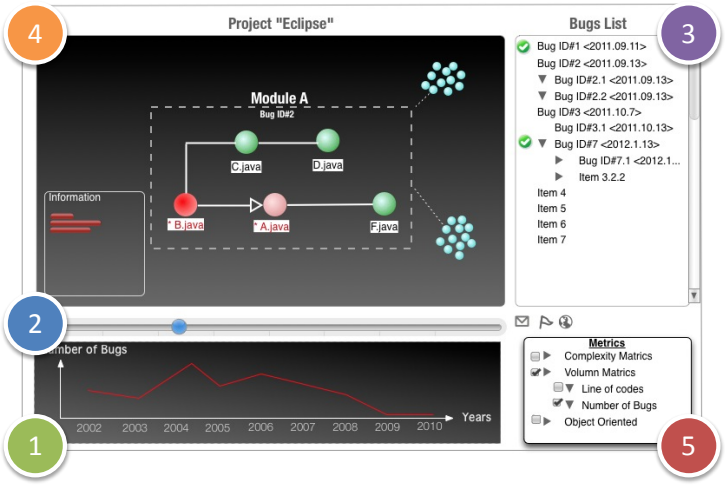


図 3-1-9　低品質モジュール可視化システムの UI

[2] 探索型可視化技術

Rank-by-feature framework に基づいた探索型可視化方式を採用した．特に，ソフトウェアプロジェクトトモグラフィを構成する 5 つの観点（要求，作業，組織，プロダクト，課題）のうち，プロダクトおよび課題を特徴（feature）とする統計量に基づいて，スナップショットを探索的に可視化／分析できるかどうかを検証した．本方式によって，ドメイン知識にかかわらず仮説を立案できること，立案した仮説の真偽を容易に確認できることが分かった.具体的には,階層的クラスタ分析ツール HCE(Hierarchical Clustering Explorer)[Seo2005][Seo2007]を用いて，オープンソースプロジェクト Eclipse Platform の課題（不具合）データから仮説生成を行えるかどうかの検証実験を行った（図 3-1-10 参照）.

HCE ツールは，入力されたスナップショットデータから，類似するデータ同士をグルーピングする階層的クラスタリング機能，分析者が注目したクラスタの任意の 2 変数間の関係を把握するための散布図作成機能，クラスタ内のデータの分布を把握するヒストグラム作成機能，などから構成されている．これら豊富な探索型可視化分析機能により，HCE ツールは，生物学分野における遺伝子解析やプロテオーム解析（タンパク質の機能・構造の大規模解析）など[Cluzet][Tsai]で数多くの利用実績がある.

分析者はまず，階層的クラスタリング機能を用いていくつのクラスタを作成するかを指定する．分析対象データは通常，多変数データであるが，変数間の類似性が階層図の下側に表示されている（図 3-1-10 では緑色と赤色）ため，容易にデータのまとまりを識別できるようになっている．注目するクラスタを選択（図 3-1-10 中，青色の階層図）すると同時に，選択したクラスタに対応するデータの散布図やヒストグラムが自動的に表示されるため，データの特徴や変数間の関係をさらに調べていくことが可能となる．さらに，本ツールには，散布図作成機能やヒストグラム作成機能のほかに，すべての変数間の相関関係を

求める機能や，平行座標プロット機能など，一画面の中で探索的にデータを分析する機能が備わっている．特に，相関関係の強い（または弱い）2 変数をランク付けすることで，重点的に分析すべき対象を分析対象に精通していない，あるいは，分析のエキスパートでない分析者にも容易に選択できるようにしている点に大きな特徴がある．

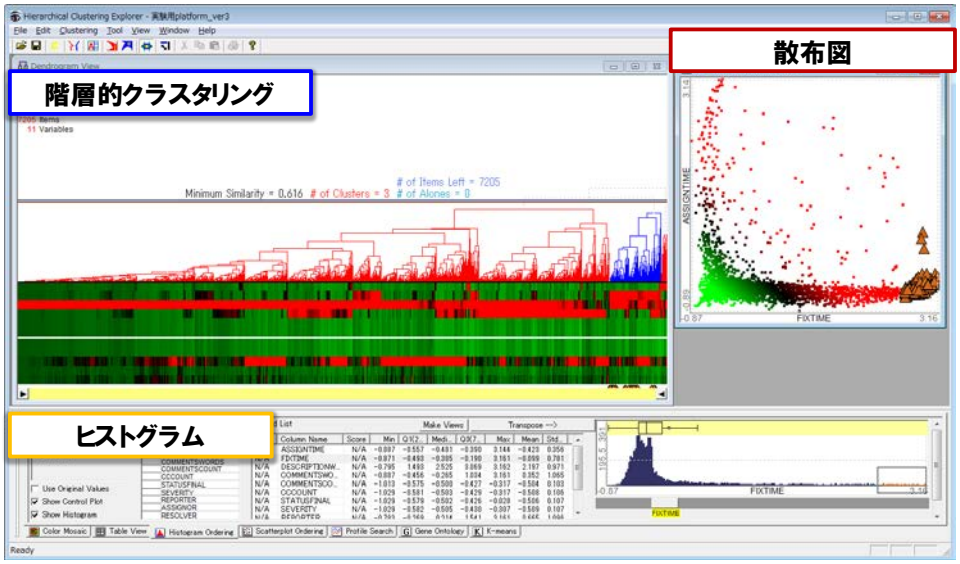


図 3-1-10　HCE ツールの概観

③「ソフトウェア品質の第三者評価」の妥当性評価実験に関する研究成果

本項「②スナップショットの生成・解析・可視化方式に関する研究成果」で示した方式や試作システムをオープンソースソフトウェア等の開発データに適用し，それら方式やシステムが，ソフトウェア品質の第三者評価を容易にすることを確認した．

1）生成方式

[1] 要件ペイン

表 3-1-2 に示した非機能要件評価表を用いて，29 件の RFP の非機能要件の評価を行った．これら RFP は，大学，病院，官公庁，地方自治体，独立行政法人などがベンダ候補企業向けの入札情報として WWW 上で公開しているものであり，5 つのドメイン（図書館情報システム，病院管理情報システム，　大学情報システム，政府機関情報システム，及び地方自治体情報システム）に分類できる．

評価者は，システム発注・開発に 10 年以上携わってきたエキスパート 1 名である．評価者は，各 RFP を熟読し，各小特性の評価点を決定した．評価に要した時間は，RFP 1 件あたり 30 分から 1 時間程度であった．このことから，経験を持った技術者であれば，現実的な時間で評価が行えることが分かった．

図 3-1-11 は，各ドメインの総合評価点の分布を，箱ひげ図で表したものである．政府情報システムと病院情報システムの 2 つのプロジェクトがわずかに 60 点を超えていたが，残りのプロジェクトは 50 点未満であった．総合評価点は 100 点に近いことが望ましいことを考えると，29 件の RFP はいずれも非機能要件の記述が不十分であることが分かった．特に，図書情報システムは総合得点の中央値が 10 点未満であり，改善の余地が大きいことが分かった．

RFP がどの程度よく書けているかを評価するためには，何らかの基準値との比較を行うことが望ましい．ここでは，ケーススタディとして，評価点の高かった 3 つのプロジェクトの評価点の平均値を基準値として採用し，比較による評価の例を示す．評価対象として，総合評価点が中央値であったプロジェクトD（県立図書館情報システム）を取り上げる．図 3-1-12 は，プロジェクト D と基準値との比較結果を示す．図 3-1-12 より，「導入教育」に関しては基準値を上回っているが，他の特性は基準値に満たないことが分かる．特に，稼働品質性能，異常検知条件，災害対策，保守容易性，サービス提供時間，ライセンス保守に関しては，基準値を大幅に下回っており，RFP の改善が必要なことが分かる．このように，基準値と比較することで，特に改善が必要な特性を明らかにできることが分かった．

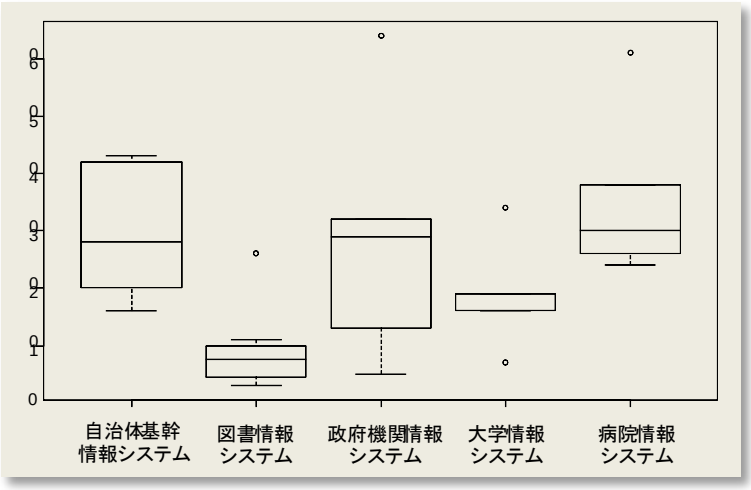


図 3-1-11　総合評価点による評価

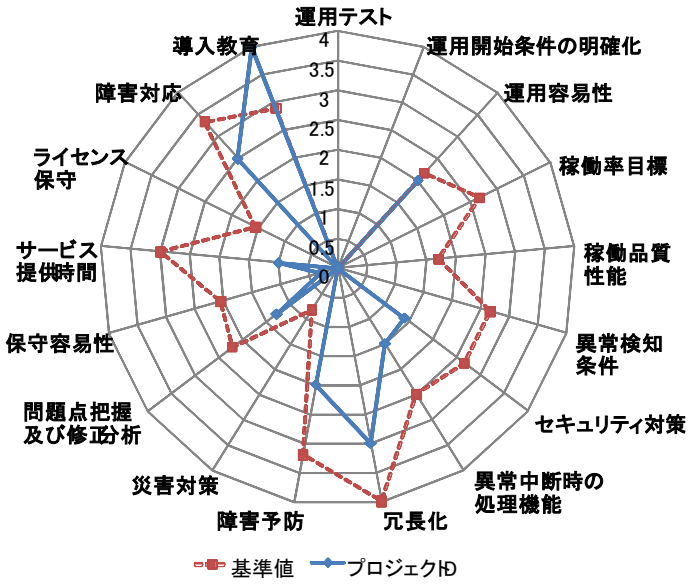


図 3-1-12　基準値との比較

[2] 作業ペイン

ソフトウェア開発タスクの自動計測の効果を実証実験により評価した．あるソフトウェア開発企業において主に Web 系プログラミングに従事する 1 名の開発者を対象として，約２週間にわたって計測を行った．計測に先立って，TaskPit の設定ファイルの調整と予備計測を 3 日間行った．各計測日について，計測を開始する前に，各タスクに従事する時間の計画値を決めてもらい，その日の計測終了後に計測結果を見る前に，各タスクに従事した時間の推定値を報告してもらった．計測結果を図 3-1-13 に示す．図 3-1-13 において，X 軸は経過日数であり，Y 軸はタスクの実施時間数である．

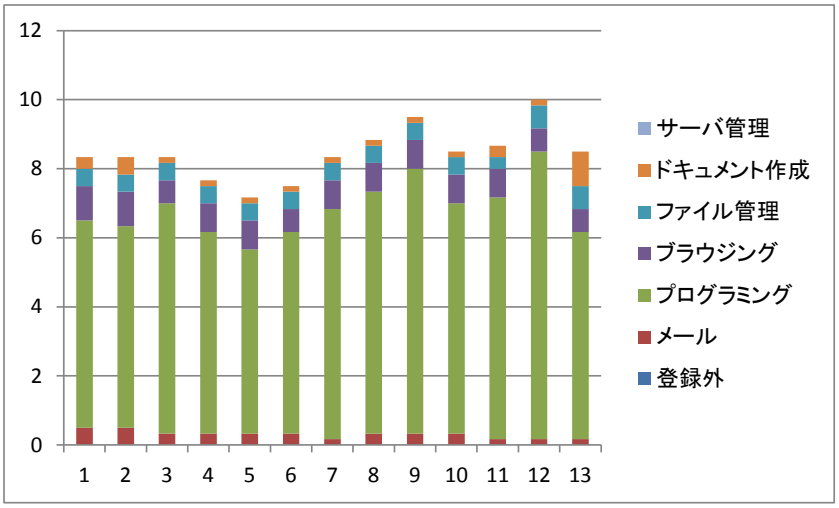


図 3-1-13　開発タスク計測における実績値の推移

図 3-1-14 に，実測値と推定値の差（実測値-推定値）の推移を示す．図 3-1-14 より，推定値については，TaskPit を使い始めた当初は過大見積もりしていたが，次第に実測値に近づくようになった．このことから，主観的と比べて実際の作業時間は小さくなりがちなこと，および，タスクの自動計測を継続することで開発者自身も自らの作業量を正しく把握できるようになることが分かった．この結果より，ソフトウェア開発作業の第三者を行うにあたっては，開発タスクの自動計測により得られる客観的なデータが重要となることが示された．

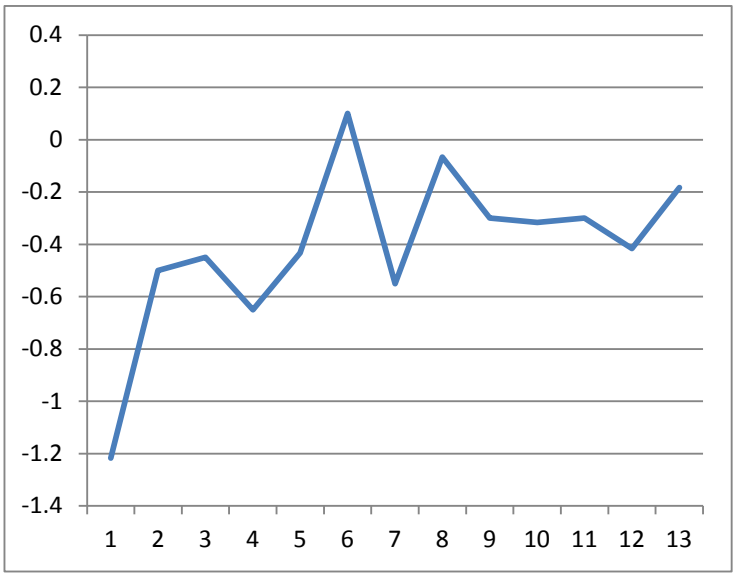


図 3-1-14　実績値－推定値の推移

[3] 組織ペイン

プロジェクト開始当初に規定した組織構造がプロジェクトの進行とともに変化する様子をスナップショットデータから捉えることができるかを検証するために，オープンソースプロジェクトの組織構造の経時変化を可視化するとともに，3 種類の中心性計測指標を用いて組織内のコミュニケーションの特徴を分析した．組織構造はプロジェクトのコミュニケーション履歴データ（メーリングリストにおける議論）から抽出したものである．分析対象プロジェクトの基本統計量を表 3-1-8 に示す．

表 3-1-8　分析対象 OSS コミュニティの統計量

	分析対象期間			総リリース数		総参加人数		総メッセージ数	
	開始年月	終了年月	期間(月)	メジャー	マイナー	開発者	ユーザ	開発者	ユーザ
Apache	2001/11	2006/09	59	1	22	1,619	9,818	32,985	68,495
GIMP	1999/10	2003/09	50	2	46	1,120	1,519	15,846	6,638
Netscape	1999/09	2007/02	92	3	11	8,161	8,002	29,417	61,946

次に，3 つのプロジェクトの組織構造をそれぞれ，分析対象期間中のある時点で可視化した結果を図 3-1-15 に示す．図 3-1-15 において，コーディネータとは，開発者コミュニティとユーザコミュニティの両者に属し，プロジェクト全体における情報流通に重要な役割を担う開発者である．通常は開発者として開発に携わる傍ら，ユーザからの不具合やユーザビリティに関するフィードバックを開発者コミュニティに伝達し，プロダクトの品質向上に努める．一方，新機能の開発予定やユーザからの要望が今後の開発にどの様に生かされるかなど，開発に関する情報をユーザコミュニティに伝達する役割も担っており，プロジェクトを円滑に実施するための重要人物である．したがって，コーディネータの活動は，プロダクトの品質に重要な影響を与えるのと同時に，プロジェクトの持続性にも大きな影響を与える．

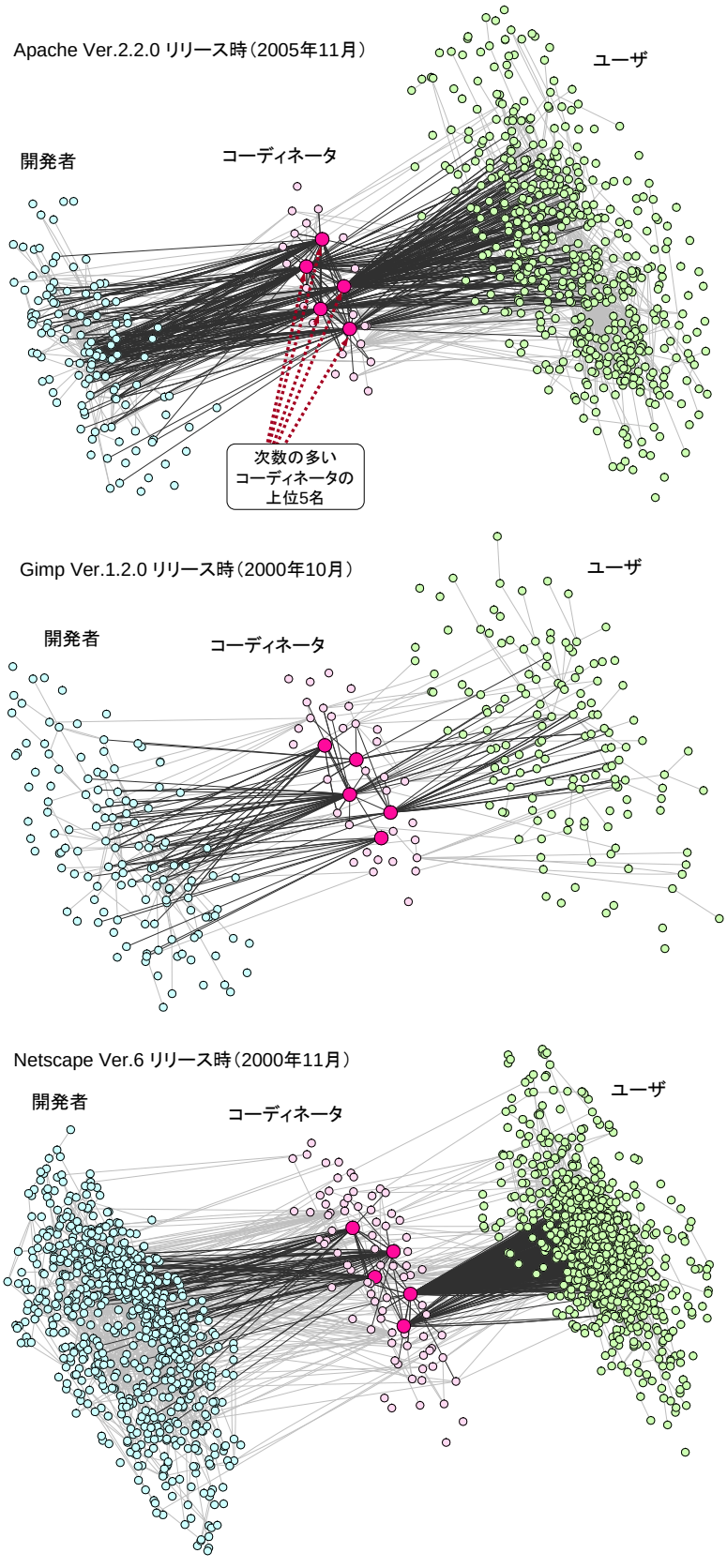


図 3-1-15　ある期間における OSS プロジェクトの組織構造

次に，組織構造の経時的変化を示すものの例として，Apache プロジェクトにおけるコーディネータの組織構造を 1 ヶ月毎に可視化した結果を図 3-1-16 に示す.プロジェクト開始当初（図 3-1-16 の左上隅）は人数も少なく目立った特徴は少ないが，開発が進むにつれて開発者が増えるともに組織構造が複雑になる様子が見て取れる．また，開発者同士の結びつきが時間とともに強くなる時期や弱くなる時期が存在することもわかる.

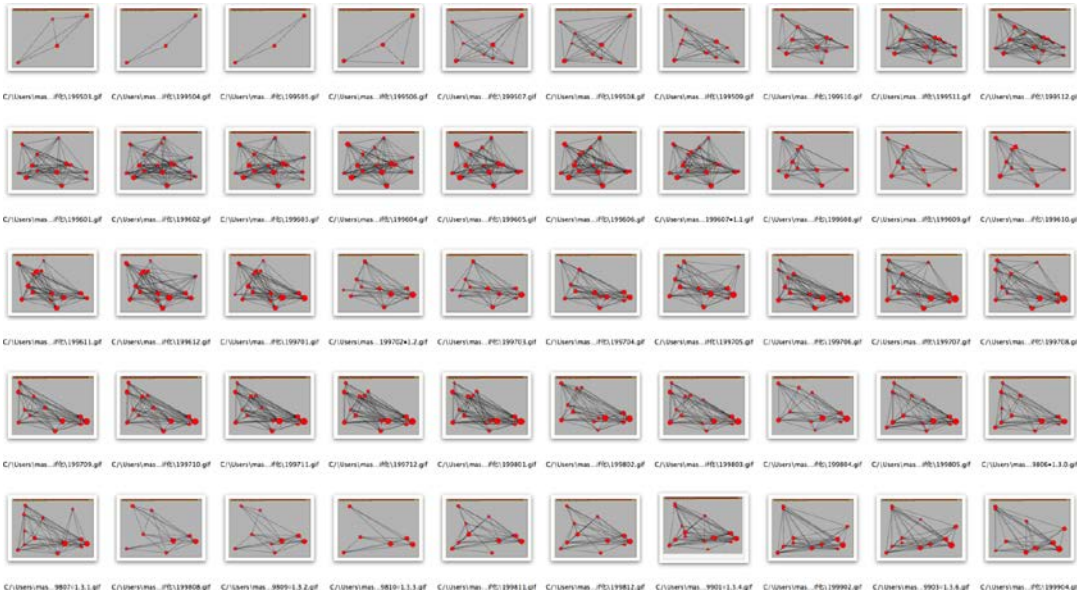


図 3-1-16　Apache プロジェクトにおける組織構造の経時的変化

更に，3 種類の中心性計測指標（次数中心性，媒介中心性，近接中心性）とプロジェクトの統計量を図 3-1-17 に示す．ここで，次数中心性とは，各ノードがどの程度の情報を発しているか，媒介中心性とは，各ノードがどの程度の情報を媒介しているか，近接中心性とは，各ノードがネットワーク全体に対してどの程度効率よく情報を発しているかを表す指標である．中心性の指標は，電子メールなどのデータを基に企業内での従業員の構造的立場を分析するために利用された事例がある.3 つ中心性を用いることにより，ネットワーク全体の構造の特徴や，個々のノードの性質について定量的な計測が可能となる．組織構造の時間的な変化を，中心性指標を用いて定量的に計測することで，個々のネットワークの構造や個々のノードの性質が時間の経過に従いどのように変化するかについても容易に理解が可能となる.

図 3-1-17 より，Apache と GIMP のコミュニティは参加者（A-1，G-1）が減少傾向にあるが各ネットワークの中心性（A-3,4,5，G-3,4,5）は，分析期間の後半にかけて著しく向上しており，組織構造が小さく密にまとまっていることが読み取れる．これは時間の経過に伴って，モチベーションの高い参加者同士が密に連絡を取り合うことで活発にコラボレーションを進められていると考えられる．一方，Netscape の参加者数（N-1）では分析期間初期から単調に参加人数が減少傾向にある．特に Ver.8 リリース直後にかけて媒介中心性（N-4)，近接中心性 (N-5) が大きく下がっており，組織構造の急激な過疎化が読み取れる.

次に，開発者コミュニティとユーザコミュニティの違いに着目する．参加者同士の緊密さを現す近接中心性はApacheとGIMPではほぼ全期間に渡り開発者がユーザより高い値を取っている一方で，Netscapeではユーザのほうが高い値をとる傾向にある．さらにNetscapeではユーザが開発者の倍以上参加していることからNetscapeコミュニティはユーザの盛り上がりに対して開発者のモチベーションが低く，ユーザからの意見や評価が成果物に反映しきれない状態であると考えられる．

このように，従来，木構造として表現される開発チームの組織構造を，コミュニケーションデータに基づいて構成した実際の組織構造との差を明確にすることで，組織構造の変化や重要人物にかかる負荷などが容易に，かつ，定量的に把握できることを確認した．

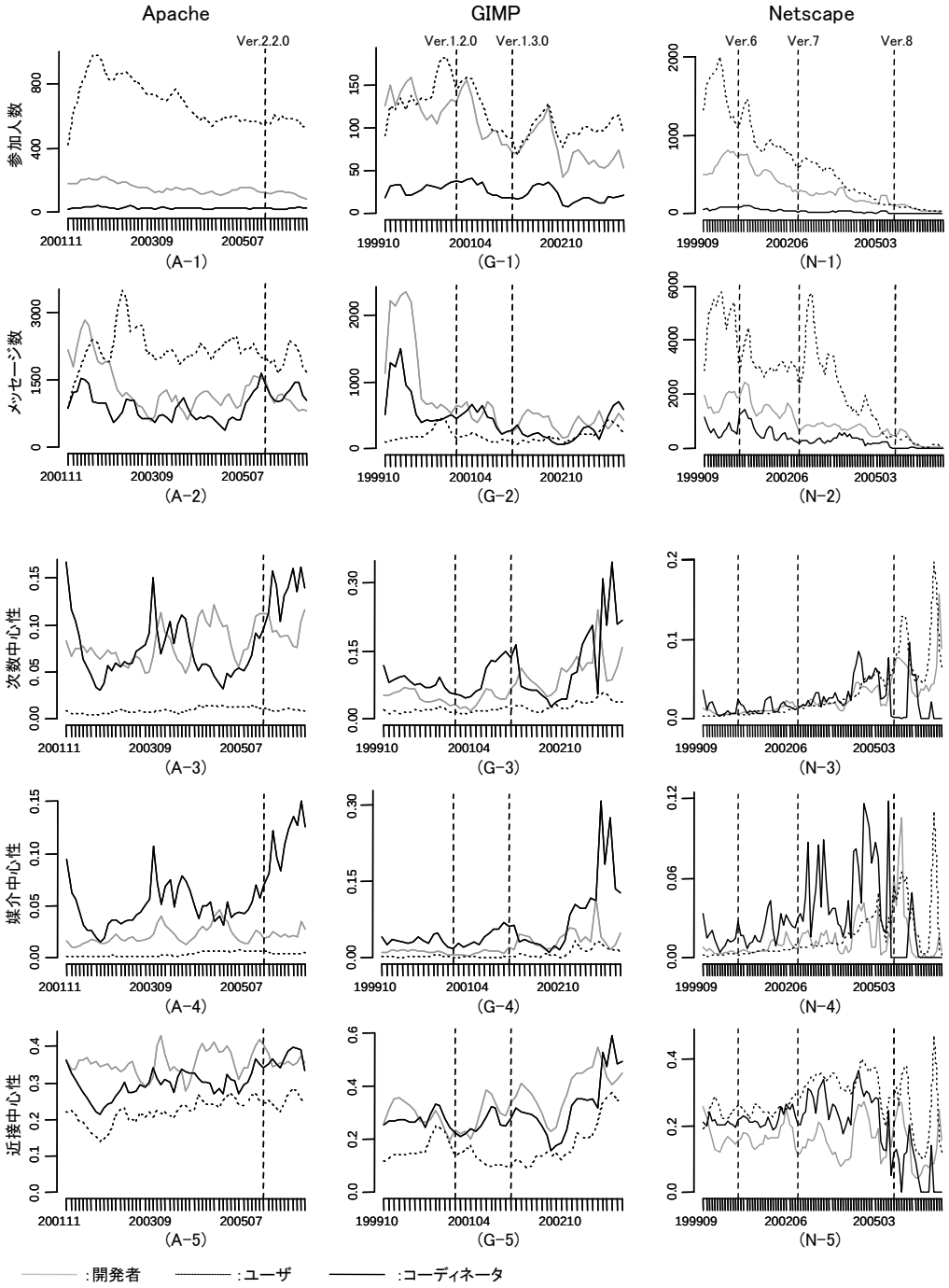


図 3-1-17　各 OSS コミュニティの統計量と中心性の推移

[4] プロダクトペイン

ケーススタディは 5 種類のオープンソースソフトウェア（OSS）を対象に実施した．対象とした OSS とそのバージョン数を表 3-1-9 に示す．対象における各メトリクスの時系列変化（バージョン推移）を計測し，作成した基準値により評価した．外れ値が観測された場合は，当該バージョンのソースコードを調査して原因を考察した．

ケーススタディの結果より，メトリクスの時系列変化から外れ値の観測されたバージョンを特定することができた．ケーススタディでは表 3-1-6 および表 3-1-7 で示したすべてのメトリクスについて，前述した OSS における時系列変化を計測した結果，外れ値が観測されたバージョンを特定することができた．外れ値が観測されたメトリクスの時系列変化を図 3-1-18，図 3-1-19 に示す．図 3-1-18，図 3-1-19 の横軸は OSS のバージョンを，縦軸はメトリクス値を表す．赤色と青色の直線はそれぞれ上限および下限の基準値を，破線はそれぞれ第三四分位および第一四分位を表している．図 3-1-19 の外れ値が観測されたバージョンに対して TreeMap を用いた分析を行った結果，該当バージョン以外のバージョンには存在しない複雑なメソッドを含むファイル（StreamPostTokenizer.java）が含まれていることを確認した（図 3-1-20 参照）．

本ケーススタディにおいて，オープンソースソフトウェアの各バージョンにおけるメトリクス値と基準値データを照らし合わせることで，基準値を超えるバージョンを特定することができた．また，TreemMap を合わせて利用することで，基準値を超過した原因となったファイルを特定することができた．

本ケーススタディでは，分析者は特に対象としたオープンソースソフトウェアに関する知識を持っていなかったがメトリクス値の時系列データや各メトリクスの基準値データ，TreeMap のみ用いて容易に品質に問題が発生したバージョン及びファイルを分析することができた．表計算ソフトやソフトウェア開発企業において広く使用されている Understand の機能のみを用いているため，高度な専門知識を持たない第三者であっても同様の品質評価を行う事ができると考えられる．また．今回用いたメトリクスの範囲であれば，企業のソフトウェア開発が対象であっても，同様の方法で分析できると考えられる．

表 3-1-9　ケーススタディ対象の OSS プロジェクトとそのバージョン数

名称	apacheAnt [UCI]	CAROL [SCITOOLS1]	Dnsjava [TECHMTX1]	jEdit [SCITOOLS2]	JHotDraw [TECHMTX2]
バージョン数	24	12	62	73	16

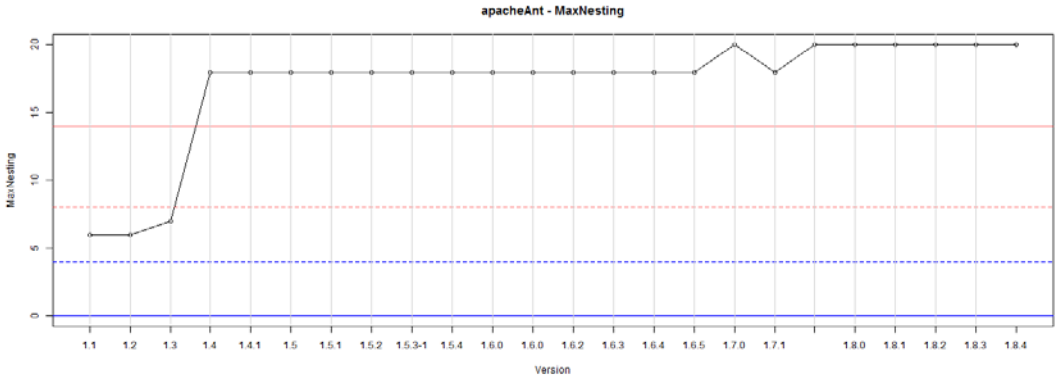


図 3-1-18　観測された外れ値（Apache Ant の MaxNesting メトリック）

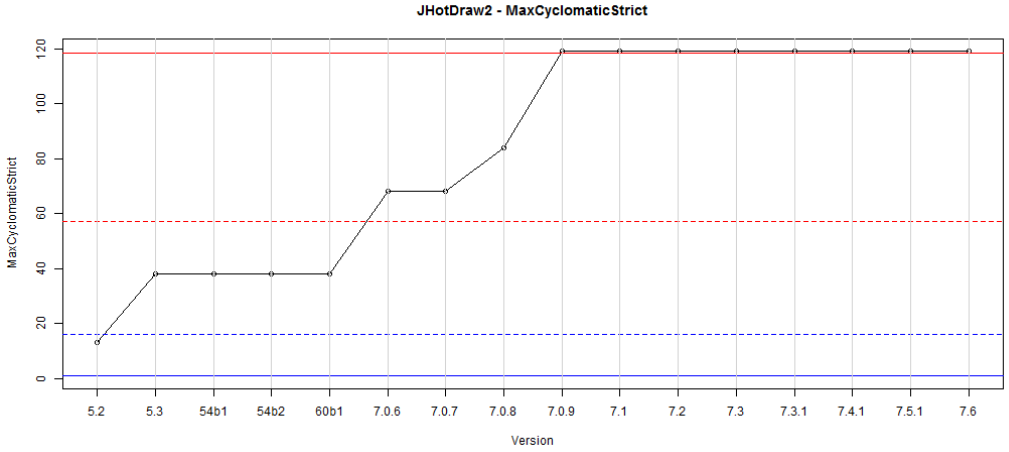


図 3-1-19　観測された外れ値（JHotDraw の MaxCyclomaticStrict メトリック）

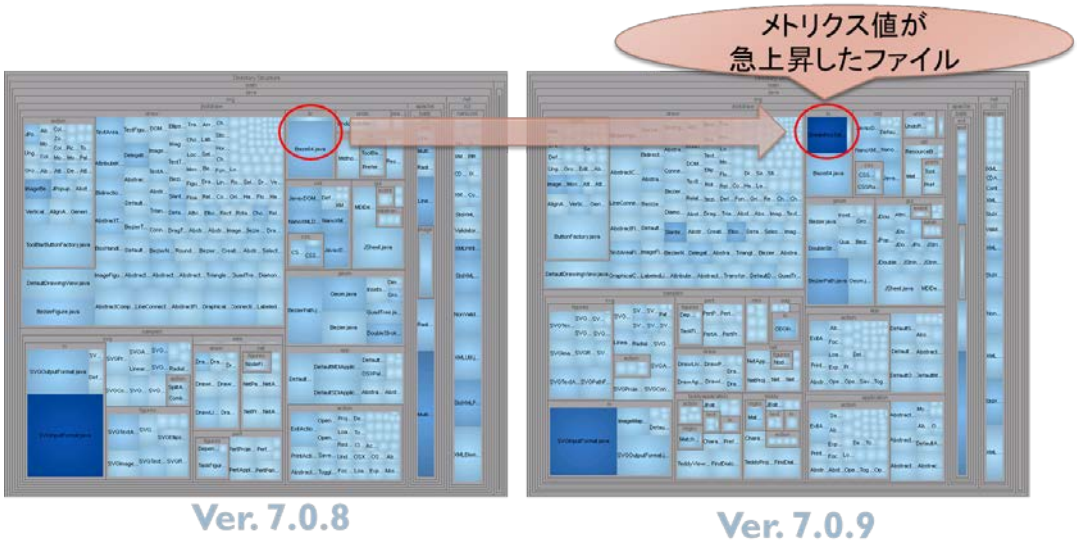


図 3-1-20　TreeMap を用いたメトリクス値の分析

表 3-1-10　課題票数，ソースコード数，リンス数の推移

	2004 年	2005 年	2006 年
課題票数	32,606	40,374	46,876
ソースコード数	58,945,705	58,319,220	78,822,485
リンク数	8,072	8,303	8,721

[5] 課題ペイン

「低品質モジュール可視化システム」（図 3-1-8 参照）を用いてオープンソース開発データを対象に，低品質モジュールを含むスナップショットの特定を行った．実験の詳細は，「2」 解析方式［1］検証型解析技術」における「低品質モジュール生成要因解析」にて説明するが，試作システムを用いて，あるスナップショットから多くの未対応課題が存在していた一因を発見することができた．プロダクトペインでは，ソフトウェア品質の外的妥当性を評価する実験を行っていたが，課題ペインでは，プロジェクト内の異常値を発見するための内的妥当性を評価する技術を提案した．

2) 解析方式

[1] 検証型解析技術

Eclipse プロジェクトの開発データ（2004 年～2006 年）を対象に，低品質モジュールの生成要因解析の実験を行った．スナップショットは，１年に 1 度行われるリリース時点のものを計 3 個作成した．表 3-1-10 は，各年に報告された課題票数，プロダクトを構成するソースコード数，課題修正のためにソースコードと紐付けることができた課題票数（リンク数）である．

OSS プロジェクトで報告される課題票のうち約 40%は重複した内容であるとの報告がある[Hooimeijer]．課題票の 40%を重複するものとして除いても，リンク数は課題票の約 40%(2004 年であれば 8072/(32,606*(1-0.4))に過ぎないことがわかる．OSS プロジェクトを対象とした他の実験においても，この割合は同程度である．課題票と課題修正されたソースコードのリンク数は，課題票の記述，および，リポジトリにソースコードを登録する際のコメント記述が明確でないことを意味しており，低品質モジュールの生成要因の解析に有用である．

[2] 探索型解析技術

Eclipse プロジェクトを対象に，低品質モジュール予測モデルの適用実験を行った．実験では，Eclipse ver. 3.1 のスナップショットを学習データ（モデル構築用データ），Eclipse ver. 3.2 のスナップショット（に含まれるモジュール）を評価対象とした．いずれも，開発期間は 1 年である．モデルへの入力となるモジュール特性値の計測には統計解析ツール Understand を用いた，Understand で計測した特性値を表 3-1-11 に示す．また，予測モデルの構築にはランダムフォレスト法を用いた．ランダムフォレスト法とは，決定木（回帰木，分類木）を用いて集団学習を行う手法である．モデル構築用のデータセットに対して復元抽出を複数回行い，得られた各サンプル群に対して決定木を構築し，各決定

木の多数決または平均により最終的な予測結果を得る．予測精度の評価には，Alberg Diagram の AUC（Area Under the Curve：曲線下面積），および，適合率（Precision），再現率（Recall），F1 値（F1-measure）を用いた．

表 3-1-11　低品質モジュールの予測に用いた特性値

特性値	定義
CountLineComment	コメント行数
CountStmtDecl	宣言ステートメント行数
CountStmtExe	実行可能ステートメント数
CountLineBlank	空白行数
CountLineCode	コード行数
CountLineCodeDecl	宣言コード行数
CountLineCodeExe	実行可能コード行数
AvgCyclomatic	関数・メソッドのCyclomatic複雑度の平均
AvgCyclomaticModified	関数・メソッドのModified Cyclomatic複雑度の平均
AvgCyclomaticStrict	関数・メソッドのStrict Cyclomatic複雑度の平均
AvgEssential	関数・メソッドのEssential複雑度の平均
CountDeclClass	クラス数
CountDeclFunction	関数の数

適用実験の結果を図 3-1-21 に示す．図 3-1-21 より，開発終了の 3 ヶ月前，6 ヶ月前，9 ヶ月前と予測時期が早まるにつれ，予測精度が低下していることがわかる．ただし，精度の低下はわずかである．例えば，AUC で比較してみると，開発終了時とその 9 ヶ月前では，0.015 の差しかない．予測精度が若干低くなるとはいえ，開発期間 1 年のプロジェクトにおいて，開発終了時に欠陥が含まれる可能性の高いモジュールを，その 9 ヶ月前に予測できることは，ソフトウェア開発管理上，内的妥当性の評価において非常に有用である．

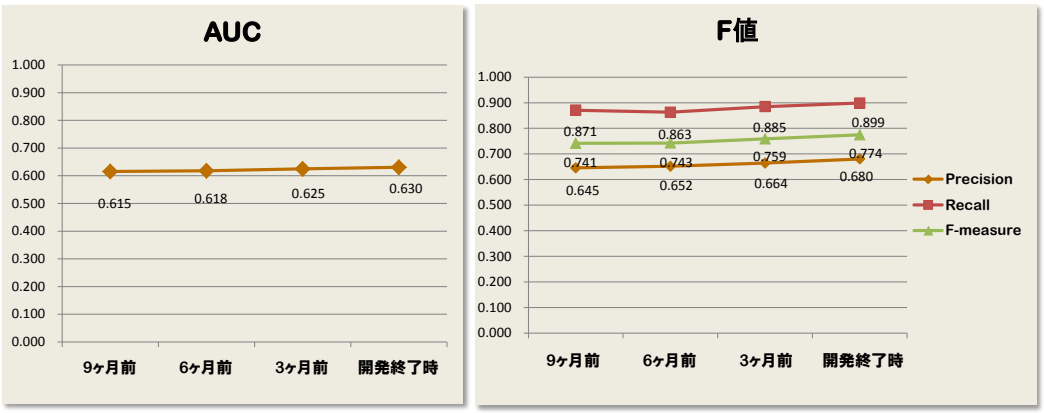


図 3-1-21　低品質モジュールの予測精度

3)可視化方式

［1］検証型可視化技術

本項「2）解析方式［1］検証型解析技術」で示した結果を，試作した低品質モジュール可視化システムを用いて視覚的にも確認した．Eclipse プロジェクトの開発データ（2004 年～2006 年）における，3 個のスナップショット間での未対応課題数，ソースコード行数，ソースコードの複雑さ，それぞれの推移を図 3-1-22 に示す．

図 3-1-22 に示した 3 つの図を見比べると，未対応課題数の推移（図 3-1-22 (a)）とソースコードの複雑さの推移（図 3-1-22 (c)）が非常に似ていることがわかる．一方，ソースコード行数（図 3-1-22 (b)）は，未対応課題数とは全く異なる推移となっている．この結果から，未対応課題数が生成される要因をより詳細に解析するのであれば，まずは，ソースコードの複雑さとの関係を分析する必要があると考えられる．また，こうした分析では，メトリクス値をソースコードの規模で正規化することも多いが，未対応課題数とソースコード行数の間に明確な関係性が見られないことから，「ソースコード行数あたりの複雑さ」といった数値を分析で用いるのであれば注意が必要と言える．

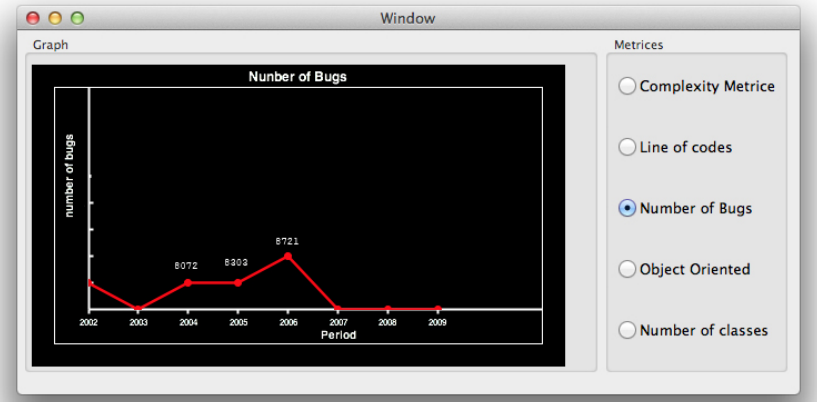


図 3-1-22(a)　試作システムの出力：未対応課題数の推移

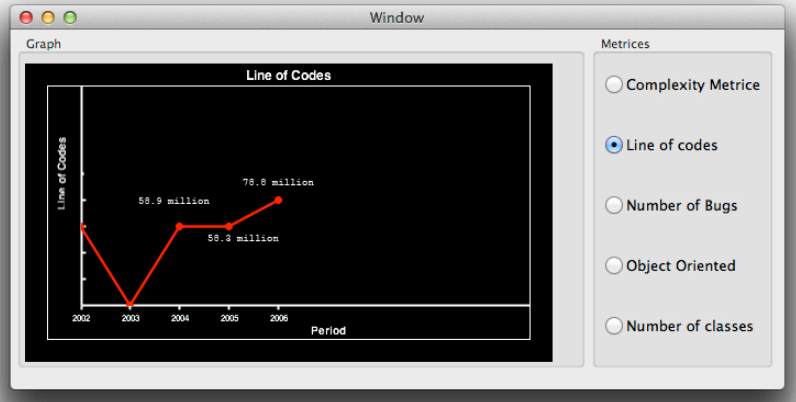


図 3-1-22(b)　試作システムの出力：ソースコード行数の推移

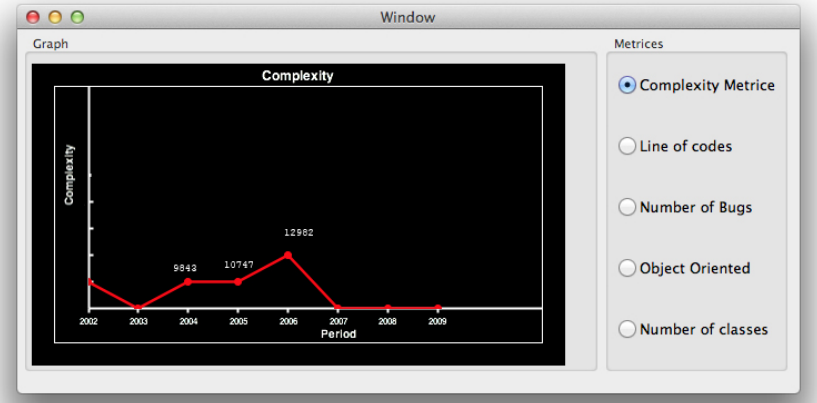


図 3-1-22(c)　試作システムの出力：ソースコードの複雑さの推移

[2] 探索型可視化技術

検証実験では，OSS 開発を熟知しているグループとそうでないグループの 2 群に分け，Eclipse Platform プロジェクトの課題データ約 7,000 件を HCE ツールで分析させた．実験の目的は，(1)Rank-by feature framework に基づく探索型可視化分析が仮説生成を支援できるかどうか,(2)分析対象のドメイン知識の有無が生成される仮説数にどれくらいの影響を与えるのか,(3)被験者が生成した仮説が国際会議等ですでに発表された知見に含まれるかどうか，の 3 点を検証することである．本実験で用いたデータセットに含まれる 11 の変数を表 3-1-12 に示す．

表 3-1-12　実験に用いた課題データに含まれる変数（項目）

項目（変数）	詳細
AssignTime	修正者が決定するまでの時間
Fixtime	修正されるまでの時間
Priority	優先度
Severity	重要度
CCCount	メーリングリストの登録者数
Comment	コメントの総文字数
Attachcount	添付ファイルの数
Descriptionword	記述情報の文字数
reporter	報告者の名前
fixer	修正者の名前
component	不具合が発生したコンポーネント名

被験者は,OSS およびデータマイニングに関連する研究を行っている大学院生 3 名をグループ A とし，それらに精通していない大学生 3 名をグループ B として構成した．HCE ツール自体の使い方に関するインストラクションを約 1 時間行った後,1 時間分析してもらった．１時間で生成された仮説数を表 3-1-13 に示す．

表 3-1-13　仮説生成実験の結果

被験者群	仮説生成数（平均）	重要な仮説の数
グループ A	18 (6.0)	4
グループ B	8 (2.7)	2

実験の結果，ドメイン知識を有した被験者が短時間で数多くの仮説を生成できることが分かった．ただし，ドメイン知識のない被験者であっても，一人平均 2,7 件の仮説を生成でき，さらに,『「課題が報告されてから担当者に割り当てられるまでの時間」と「課題が割り当てられてから解決されるまでの時間」は反比例する.』といった，すでにソフトウェア工学の研究分野で検証済みの重要な法則を仮説として生成できることが分かった．これらの結果は，ドメイン知識の不足する分析者であっても，課題対応プロセスの実施・適用技術の妥当性評価につながるものと考えることができる．

3.1.3 実用化へ向けた課題と問題点

(1) 課題と問題点

① スナップショットに関する課題と問題点

「3.1.2 研究プロセスと成果」では，スナップショットの詳細設計として，プロジェクト構造モデル（ER 図）を示したが，実用化に向けては，実装を念頭においたもう少し詳細な検討が必要となる．例えば，ソフトウェアプロジェクトの 5 つの構成要素それぞれについては，一般的な階層構造で表現されるとして，それ以上の検討は行っていない．また，各エンティティの属性についても，追加することは容易であるが，どのような属性をいくつ追加するのか，実用化の観点からの検討が必要である．

実用化へ向けた別の技術的課題としては，「既に収集・蓄積されているソフトウェアプロジェクトデータのスナップショット化」がある．スナップショットは，「ソフトウェアプロジェクトデータの共有や再利用の促進」に有効であると考えているが，その効果をより高め，技術としてより普及させるためには，ソフトウェアプロジェクトデータをスナップショット（の集合体）として新規に収集・蓄積するだけでなく，既に収集・蓄積されているデータをスナップショット（の集合体）に変換し共有や再利用の対象とすることも必要になってくる．スナップショットへの変換は難しくはないが煩雑ではある．スナップショットの効果として「ソフトウェアプロジェクトの解析や可視化に要するコスト低減」を挙げているが，その効果が相殺されてしまわないよう，既に収集・蓄積されているデータからスナップショットへの自動，あるいは，半自動変換技術の開発が必要である．

② スナップショットの生成・解析・可視化方式に関する課題と問題点

1）生成方式

[1] 要件ペイン

Web 上で公開されている（官公庁や病院などの）提案依頼書を用いた評価を行ったが，民間企業の提案依頼書の評価は（入手が困難なため）行えていない．

非機能要件キーワード評価シートに基づく採点は，人手で行う必要があるため，初心者による評価は難しい．今後，評価の（半）自動化が必要となる．今後の課題として，（１）さらに多くの自然言語で記述された RFP サンプルデータの入手（２）非機能要件キーワードと文章構造としてのコーパスの構築（３）「非機能要件特性キーワード評価表」の汎用化（４）意味論的な文章のあいまい性を非機能要件記述の問題点として評価する手法および（５）RFP における非機能要件とソフトウェアアーキテクチャへのトレーサビリティ方法論　について検討を行っていく必要がある.また,要件ペインとその他のペインとのデータの連携についてはソフトウェアプロジェクトトモグラフィ全体のデータ構造に準じて生成する必要がある．一方，機能要件については，ドメインの依存性が高いため汎用的な要件評価モデルとドメインごとの要件評価モデルについて構築を行う必要がある．

[2] 作業ペイン

タスクの自動計測では，コンピュータを使わないタスクの計測が行えないことが問題である．従って，開発作業の第三者評価のためには，従来の作業日報と併用することが必要となる．また，計測ツール TaskPit の制約上，計測結果が常時，被計測者（開発者）にも見えてしまうため,開発者自身が計測結果をコントロールすることもある程度可能である．例えば，定期的にプログラミング環境を立ち上げて作業をしているように見せかけることもできてしまう可能性がある．今後，計測ツールの改良や，新たな設計・開発が必要となる．

[3] 組織ペイン

要求分析や設計など，上流工程で実施される作業における組織構造と，実装やテストなど，下流工程で実施される作業における組織構造とを比較するためには，利用するデータの粒度を揃える必要があり，現時点ではまだ不十分である．また，組織構造を抽出するために利用するメール等のコミュニケーションデータからは，区切りとなる工程は識別できないため，ソフトウェアトモグラフィ全体のデータ構造から工程を識別する仕組みが今後必要となる．

組織構造を時系列で可視化・計測する際には，ノード間の関係が継続する期間を，プロジェクト毎に適切に定義する必要がある．例えば，検証実験では 3 つの OSS プロジェクトを対象としたが，すべて長期間継続した（Apache プロジェクトは約 10 年分のデータセット）プロジェクトであったため，3 か月ごとにデータを分割し中心性指標を求めた．中心性指標は，計測対象のノード数に依存して大きく値が変わるものであるため，プロジェクトに所属する開発者の数によって計測期間を定義できるようガイドラインを整備することが，今後の実用・普及にとって必要となることが分かった．

[4] プロダクトペイン

より実用性の高い解析を行うためには，プロダクトペインに取り込むメトリクスの種類を増加させる必要があると考えられる．現状では，静的解析ソフトウェア Understand が計測可能なメトリクスを中心にプロダクトペインに取り込んでいるが，企業で行うソフトウェア開発において頻繁に使用されるテストカバレッジ等のメトリクスを取り込む必要があると考えられる．

[5] 課題ペイン

全ての課題票を課題解決のために変更されたソースコードと紐付けることは困難である．その主な理由は，ソースコードの変更履歴に課題票に関する情報が記載されていないことにある．オープンソース開発データを調査した結果，正確に紐付けできるのは 40%程度である．現在，国際会議 ICSE，FSE，MSR などで，課題票とソースコードを紐付けるための新たな技術を提案する研究が数多く発表されており，今後，精度の向上が期待される．

2) 解析方式

カリフォルニア大アーバイン校が提供する大規模ソースコード集合を基に品質メトリクスの基準値データを作成したが，これらソースコードは Java 言語で記述されているため，今後 C/C++言語等の他の言語についても基準値データを作成する必要がある．基準値データの作成のためには，適切な大規模ソースコード集合を収集する必要がある．収集先の候

補としては，各種 Linux ディストリビューションに含まれている C/C++で開発されたオープンソースソフトウェアや，FreeBSD の ports に含まれているオープンソースソフトウェアが挙げられるが，適切な基準値作成のためには偏りの少ないオープンソースソフトウェアの選別方法を考案する必要がある

3) 可視化方式

[1] 検証型可視化技術

試作したシステムは，プロジェクトデータからスナップショットを抽出し，品質を評価するプロセスを容易にするために開発された．オープンソースプロジェクトのデータを対象に，本プロジェクトの学生研究員が試作システムを用いて低品質モジュールの評価を行った．低品質モジュール生成要因を分析するために，ソフトウェア評価で一般的に使用されるメトリクスで分析を行ったため，実際の開発者でも同様の結果が得られると考えられる．しなしながら，この評価は，実験対象としたオープンソースの開発者が評価してないため，厳密には内的妥当性を評価できたわけではない．今後は，実験対象としたソフトウェアの開発者が分析するというシナリオを用いて，検証型可視化技術の効果を検証する必要がある．加えて，システムを使用しない場合と比べて，評価するための効率が向上しているか否かについても評価する必要がある．

[2] 探索型可視化技術

実験に用いたデータセットは，オープンソースプロジェクト Eclipse Platform から取得したプロダクトおよび課題データの一部（バージョン 3.1 の約 7,000 件程度の不具合票）である．さらに大規模なデータを用いた場合でも検証を行い，計算量の観点からのスケーラビリティを評価する必要がある．また，データが大規模になった場合でも，第三者が容易に仮説立案／検証できるのかどうかについても調べる必要がある．

また，実験では，オープンソースプロジェクトのデータを用い，かつ，学生を被験者としていたため，実務者が分析した場合にも同様の結果が得られるのかは定かではない．実務者の観点からデータ分析を行えば，更に有用な知見となる仮説が生成できる可能性もある.今後は,ソフトウェア開発企業のデータを実務者が分析するというシナリオを用いて，探索型可視化分析技術の効果を検証する必要がある．

③ 「ソフトウェア品質の第三者評価」の妥当性評価実験に関する課題と問題点

今後は，ソフトウェア開発企業が開発したプロダクトや実プロジェクトそのものを対象とする必要がある．なお，個別の企業で評価実験を行い，その結果をその企業内だけで共有するのでは，得られた知見に基づく更なる技術開発が難しくなる．評価実験や得られた結果の匿名化方法を考案し，広く適用結果を普及できる仕組みを実現することが課題の一つである．

(2) 将来の応用方法

① スナスナップショットの応用方法

「3.1.3 実用化へ向けた課題と問題点」で述べた通り,実用化に向けては,スナップショットのより詳細なデータ構造と実装方式の検討が必要となる．処理効率などの要検討事項を

ひとまず棚上げしてよければ，比較的単純な実装方式の一つとして考えられるのは，「プロジェクト構造モデル上の各エンティティをスプレッドシートで表現し，中核となる「実施」エンティティをチケット駆動開発における「チケット」に対応づける」方式である．チケットの構成要素はカスタマイズ可能であり，スナップショットと親和性の高いチケットを定義することができれば，ソフトウェアプロジェクトトモグラフィ技術を実プロジェクトに導入するための，実現可能性の高いアプローチの一つとなる．また，その結果として，チケット駆動開発におけるプロジェクト管理や品質管理の高度化が期待される．

② スナップショットの生成・解析・可視化方式の応用方法

1) 生成方式

[1] 要件ペイン

非機能要件のスナップショットは，RFP や要件定義書の評価を行う目的に使用するのみならず，テスト計画書における非機能要件のテスト網羅性の評価にも応用することが考えられる．また，時系列に遷移する非機能要件と他のペインのスナップショットにより，要求変更が及ぼす影響を予測し,プロジェクトを管理するために利用することが可能となる．

[2] 作業ペイン

ソフトウェア品質の第三者評価のために開発タスクの自動計測を行い，タスク実績シートを生成するが，開発組織にとってもタスク実績シートは有効である．多数の部署から成る開発組織において，仕事が暇な部署があったとしてもわざわざそのことを報告しないケースがある．このような場合，作業日報は信頼できないため，自動計測に基づくタスク実績シートが役立つと考えられる．

[3] 組織ペイン

作業進捗の予実管理は一般的であるが，組織構造の予実管理が行えるようになる．想定の組織構造からの乖離を確認できれば，開発途中でのメンバ変更・追加をより適切に行えるようになる．

[4] プロダクトペイン

本委託研究において，一次解析結果としてプロダクトペインに取り込んだメトリクス以外にも，ソフトウェアテストにおけるカバレッジなど，企業が行うソフトウェア開発において頻繁に使用されているプロダクトメトリクスが存在する．これらメトリクスをプロダクトペインに取り込むことで，より効果的に第三者評価を支援することができると考えられる．

[5] 課題ペイン

課題とソースコードを対象に紐付けを行っていたが，ソースコードと設計書の機能要求と紐付けるなど，その他のペインの情報と紐づけることで，ソフトウェア開発全体の品質評価を可能にする．

2) 解析方式

解析に用いられるメトリクス値やその基準値のうち，プロダクト（ソースコード）に関するものは,Java 言語で記述されたソースコードを対象としている．今後は,Linux のディストリビューションに含まれている C/C++言語で記述されたソースコードも対象とするこ

とで，本委託研究の成果の適用や応用の範囲が広がると考えられる．また，Java と C++の間で基準値データの比較を行い，どちらの言語を選択した方がより保守性を向上させることができるかという議論を行うことができると考えられる．

3）可視化方式

検証型可視化ではソフトウェア品質そのものを，探索型可視化ではソフトウェア品質の第三者評価に求められるプロセス実施や採用技術をそれぞれ対象とした．ただし，ソフトウェア開発データがスナップショットの形で整理されていれば，可視化システムへの入力は容易であり，解析方式と合わせての検討は必要であるが，可視化の対象そのものが限定されるものではない．また，特定の解析方式に強く依存しない可視化手法・ツールも存在する.例えば,可視化の対象をその類似性で可視化したいのであれば自己組織化マップを，対象の特性を量的に捉えたいのであれば Treemap を，それぞれ用いることが考えられる．今後は，スナップショットが提供する 5 つのペインにまたがる可視化を推進することで，ソフトウェアプロジェクトをより多面的に議論できるようになると考えられる．

③ 「ソフトウェア品質の第三者評価」の妥当性評価実験の応用方法

妥当性評価実験は，オープンソースソフトウェアを対象として行った．ただし，ソフトウェア開発企業で行われている実プロジェクトで同様の評価実験を行うことは可能である．今後は，ソフトウェア開発企業と連携し，ソフトウェア品質の第三者評価をシナリオとして具体化したうえで，実プロジェクトでの評価実験を行いたいと考えている．例えば，本委託研究では，プロダクトの一次解析結果として，静的解析ソフトウェア Understand が計測可能なメトリクスを中心に扱ったが，ソフトウェア開発企業と連携し，実プロジェクトで広く用いられているメトリクスや必要性が高いとされるメトリクスの情報を得て，ソフトウェア品質の外的妥当性をより適切に評価することが考えられる．

3.2 研究目標 2「クラウド型開発管理環境の有効性の実証と課題の抽出」

3.2.1 当初の想定

(1) 想定する仮説等

① クラウド型開発管理環境に関する仮説等

開発管理のクラウドサービスと定量的プロジェクト管理ツールを組み合わせることで，クラウド型開発管理環境を構築することができる．

② 組織間協業機構に関する仮説等

クラウド型開発管理環境は，ソフトウェアプロジェクトデータをクラウド化し，組織間協業を促す．

(2) 当初の到達目標

① クラウド型開発管理環境に関する到達目標
クラウド型開発管理環境の構成方式および運用方式の評価

② 組織間協業機構に関する到達目標
クラウド方式の利点を活用した組織間協業の実現可能性評価

(3) 当初の期待される効果

① クラウド型開発管理環境に期待される効果
これまでの長年の研究・実環境での評価を反映したプロジェクト計測・可視化機構をクラウド型で利用すれば，プロジェクト運営の円滑化，生産性・品質向上に大きく貢献するはずだ．このことを通して，こうしたプロジェクト可視化によるプロジェクト運営手法の一層の普及が期待できる．

② 組織間協業機構に期待される効果
高度な機能を装備したプロジェクト可視化機構をクラウド型で利用する形態が普及すれば，そのネットワーク機能を活用して，プロジェクトの説明性，追跡性が高まり，さまざまな利害関係者の存在する組織間の協業が促進されるはずである．クラウド形態の実証実験により，こうした組織間協業のイメージが高度化されることが期待できる．

3.2.2 研究プロセスと成果

(1) 研究プロセス

開発管理のクラウドサービスと定量的プロジェクト管理ツールを組み合わせることでクラウド型開発管理環境を実現し，ソフトウェアプロジェクトデータのクラウド化に向けた基盤技術を開発する．具体的には，次の通りである．

ファーエンドテクノロジー社（島根県松江市）よりすでにクラウド形態で商用提供されている課題管理システム Redmine と構成管理システム Subversion のサービスに IPA/SEC 提供の定量的プロジェクト管理システム（EPM-X）を組み合わせた環境を実現した．次いで島根県で Ruby を中心にゆるやかな結びつきではあるが進取の気性に富み，ソフトウェア開発の分野で活発に活動する産業コミュニティを構成しているファーエンドテクノロジー社の顧客企業群に実証活動への参加を呼び掛けた．

実際に呼びかけに応じた１プロジェクトにこの環境を提供し，これを本委託研究の研究員が進行中にモニタリングした．

対象プロジェクトは元請け企業，協力会社から構成される受託開発，プログラミング言語は Ruby，既存のオープンソースプログラムの機能に追加して使用する機能の開発で，ま

た内部に別のオープンソースプログラムの機能を取り込む構成．開発期間は 3 カ月弱．プロジェクトのモニタリングにあたって大学と元請け企業と守秘覚書を交わした．

当該実証活動の企画から経緯，結果を素材に，ファーエンドテクノロジー社の顧客企業を対象にファーエンドテクノロジー社を含めてソフトウェア企業 4 社の様々な立場の人と意見交換した．具体的な手順は次の通り．

1）クラウドサービス・パートナの確保
2）計測・実証活動ターゲットの設定
3）計測・可視化環境構築
4）守秘契約の検討
5）計測・実証活動の勧誘
6）参加プロジェクトの確保
7）守秘覚え書きの締結
8）チケット設定
9）開発プロジェクト推進とモニタリング
10）フォロー活動
11）地域企業群での協業の可能性等の調査

これらの手順の中で，特記すべきいくつかプロセスについて，その詳細を次に述べる．

1）クラウドサービス・パートナの確保

本委託研究では，ソフトウェア開発管理環境のクラウド型サービス提供の先駆企業であるファーエンドテクノロジー社およびその創立者の前田剛氏（代表取締役）とパートナ関係を構築した．具体的には，実現した環境の上で，適切なアクセス管理のもとにプロジェクトの計測・可視化と情報の共有を図った．

また，斯界の第一人者である前田剛氏の経験，見識と人的あるいはビジネス上のネットワークを研究遂行に反映した．前田氏は 2007 年に Redmine 利用者向けの我が国最初のポータルサイト Redmine.JP を開設し，以降この運用を通して斯界でリーダシップを発揮してきた．前田氏は 2010 年に共著でファーエンドテクノロジー社から「オープンソースによるプロジェクト管理入門」を出版した[Farend]．また，2008 年に「入門 Redmine」を出版し，改版を重ね本委託研究中の 2012 年 8 月には第 3 版を出版した[Maeda]．前田氏は，本委託研究期間中に出版され斯界で多く読まれている，小川明彦，阪井誠著「チケット駆動開発」（2012 年）のレビューアを勤められている[Sakai]．前田氏は，IPA/SEC から「定量的プロジェクト管理ツール EPM-X」が提供された際，いち早く（即日）その Redmine 連携版の公開評価サイトを立ち上げた．ファーエンドテクノロジー社は Redmine について唯一の日本人コミッターである，「まるやま」氏を技術顧問に擁し，Redmine に関し日本の情報中枢となっている．また前田氏は島根大学で非常勤講師として Ruby プログラミングの講義もしている．

5）計測・実証活動への勧誘

先に述べた緩やかなコミュニティの中で，企業訪問などを通して，活動参加企業を募った．勧誘にあたって，図 3-2-1 に示すような考え方で，可能な限り活動参加障壁の低い企

業へのアプローチをこころみた．勘と経験と度胸(KKD)によるマネジメントではなく，ソフトウェアエンジニアリングやメトリクスへの理解があり，構成管理システム(Configuration Management System:CMS)として Subversion，課題追跡システム(Issue Tracking System:ITS)として Redmine を適用済みで，さらにチケット駆動開発を試み，クラウド型の開発管理環境を使用している企業群を参加呼びかけのターゲットとした．

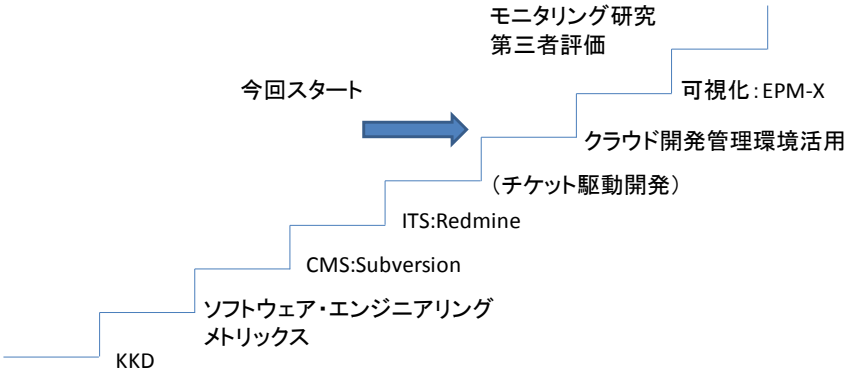


図 3-2-1　実証活動への参加障壁と参加勧誘のスタートポイント

6）参加プロジェクトの確保

参加プロジェクトの確保はなかなか容易でなかったが，その中で 1 件，計測の機会を得ることができた．図 3-2-2 に示すような日本で典型的な受発注関係の，短期，小規模開発である．

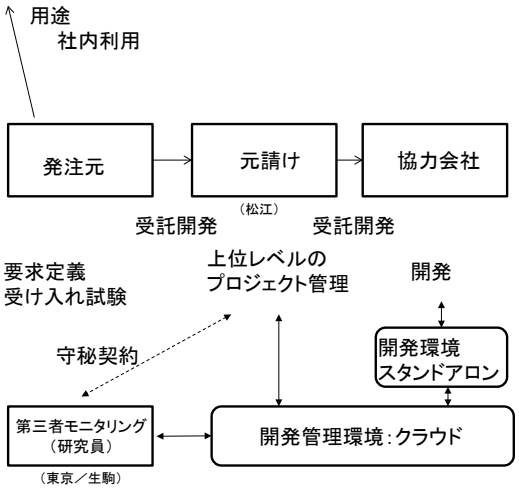


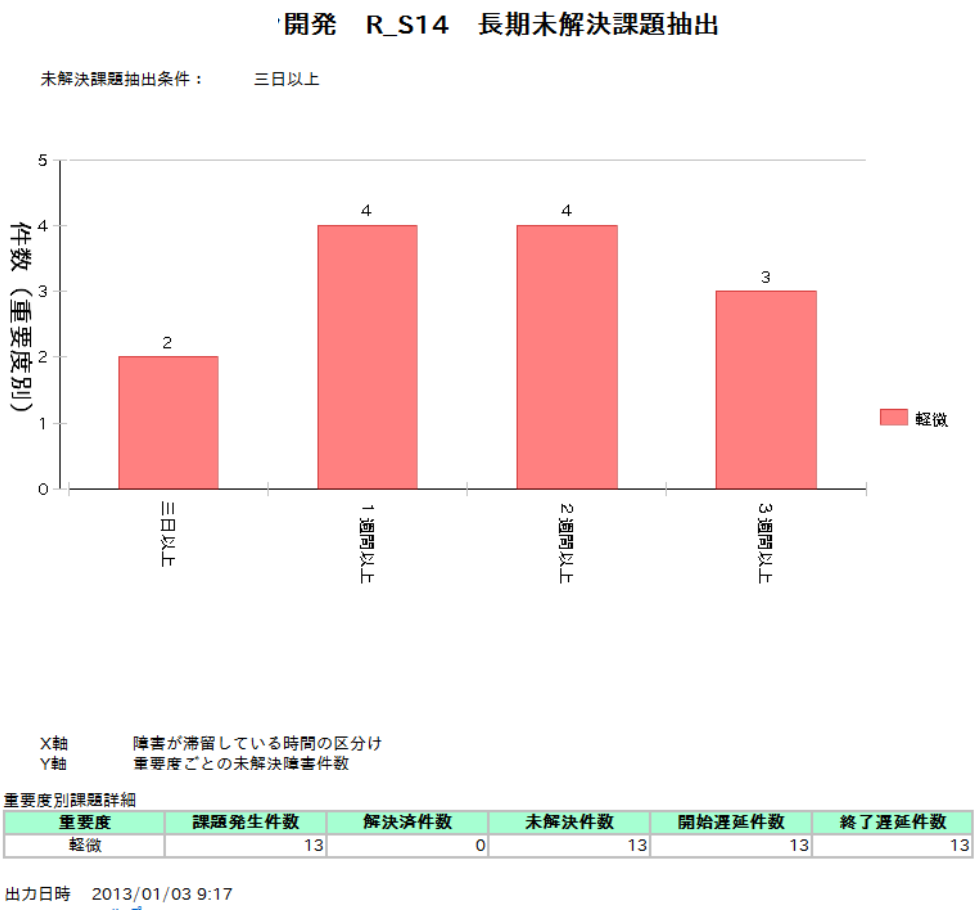
図 3-2-2　参加プロジェクトの受発注構造

8）チケット設定

Redmine のチケットの利用にあたっては，基本的な事項として，チケットの種類に相当するトラッカー，チケットの記述項目，チケット関係を規定するチケット構造の定義が必要である．今回は参加プロジェクトの元請け企業が標準的に考えているチケット構造と管

理負荷の軽減を念頭に，大幅に軽装なチケット定義を基本とした．具体的には，EPM-X 提供の標準サンプルのインストール・キットのチケット定義から，オプションの項目を大幅に削減した．トラッカーを WBS，障害，課題の 3 種をそのまま採用した．

EPM-X を介して結果として実際に得ることができたグラフは，図 3-2-3，図 3-2-4 に示すようなもので，長期未決課題と遅延重要タスクなど目指したもののごく一部しか得られなかった．プロジェクトの可視化結果の進行中のプロジェクト運営への反映は想定通りできなかった．この想定と実績の乖離の中に，ソフトウェア開発現場の実相と今後のテーマとなる大きな課題が判明した．一方，クラウド型環境による開発管理の大きな枠組みの有用性は明らかにできた．



重要度	課題発生件数	解決済件数	未解決件数	開始遅延件数	終了遅延件数
軽微	13	0	13	13	13

図 3-2-3　出力可視化グラフ例（長期未決課題：EPM-X 画面）

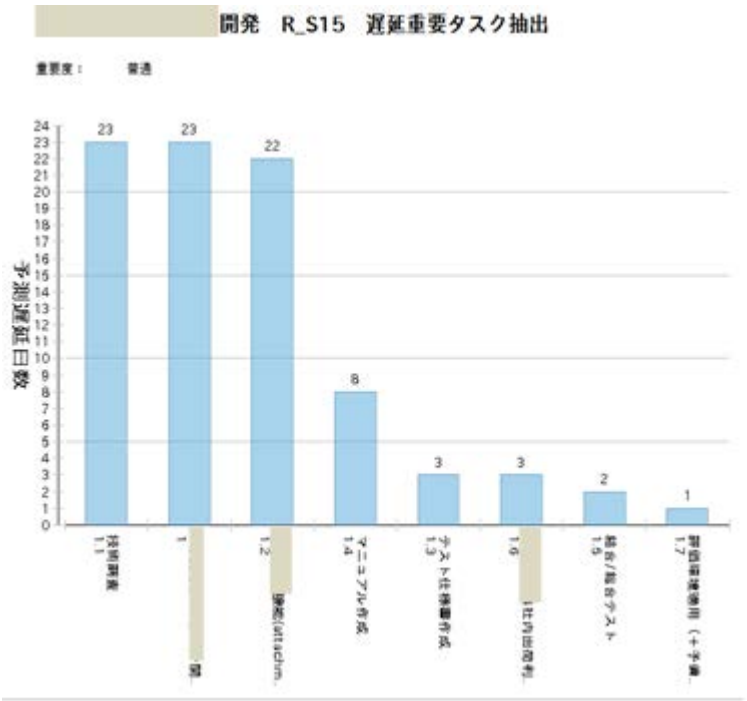


図 3-2-4　出力可視化グラフ例（遅延重要タスク：EPM-X 画面）

11）地域企業群での協業の可能性等の調査

実際のモニタリング結果と当初意図した計画などをもとに，実証活動参加を勧誘したいくつかの企業を訪問調査，意見交換した．意見交換先は，現場の実務者，中堅管理者，指導的な経営者などを含んでいる．表 3-2-1 に訪問した調査協力企業の概略プロファイルを示す．これらの企業が参加・支援しているコミュニティとその活動の例を示す．

表 3-2-1　調査協力企業の概略プロファイル

企業	社員数	資本金	年間売上げ
A	200人以下	1億円(授権4億円)	50億円以下
B	150人以下	2,500万円(授権5,000万円)	15億円以下
C	50人以下	5,000万円以下	
D	10人以下	500万円以下	

・コミュニティ組織

Ruby アソシエーション

島根県情報産業協会

しまねソフト産業ビジネス研究会

しまねオープンソースソフトウェア協議会

・活動例

Ruby World Conference

Ruby ビジネスセミナ
オープンソース・サロン，オープンソース・カンファレンス
オープンソース活用ビジネスコンテスト
各種講習会，講演会，交流会
オープンソース・ラボ開設・運用（松江駅前）
松江 Ruby 会議
しまね IT 産業育成事業（各種連続講座，研修）

(2) 具体的な研究成果の内容

① クラウド型開発管理環境に関する研究成果

新しいクラウド型の統合的なプロジェクト管理・可視化機構を実現することで，これまでの数年間の研究や実証活動を反映した統合的な開発管理環境の姿を描き出すことができた．それは図 3-2-5 に示すように，WBS や EVM といったプロジェクト管理手法，ストーリーカード，タスクカードといったアジャイル手法，そして伝統的な問題処理票，障害票とこれらを統合したチケット駆動開発手法，さらに構成管理システム，版管理システムを駆使したプロジェクト管理方式，およびこれらの連携機能，そしてここ数年研究してきたインプロセス計測と可視化の技術，さらにビジネスインテリジェンス（BI）と呼ばれる企業の業務システム分野の可視化技術を統合したものである．

本委託研究の実適用プロジェクトにおいて,描かれた統合環境は想定通りには機能せず,想定通りには受け入れらなかった．かわりに，実適用プロジェクトの推進によって，島根県のような地域における，小規模あるいは中規模のソフトウェア開発を対象とする企業群における，ソフトウェア開発管理の実相を明らかにすることができ，次の段階への道程について重要な示唆を得た．

すなわち，現場のプロジェクト管理手法は WBS 駆動型の管理と，開発するソフトウェアの機能中心に考えるフィーチャ駆動型に 2 分されている．前者は EXCEL 帳票により管理されていて，後者は Redmine のようなチケット駆動で管理されている．

どちらもプロジェクト計測や定量的管理からは遠い．特に後者は，プロジェクトの直接的な進捗を重視し，ソースコード行数，障害原因の分類と記録・集計といったメトリックスには興味を持たない，というものである．特に，EPM-X のデフォルト機能としてサンプル提供されている WBS チケットを主体とした手法は Redmine を使用するチケット駆動のグループには異質なものとして全く受け入れられなかった．

統合的なプロジェクト管理機構はこれらの実情にきめ細かく適合した機能が必要で，またその実情を反映した普及策，ソフトウェアエンジニアリング啓発策が必要なことが明らかになった．

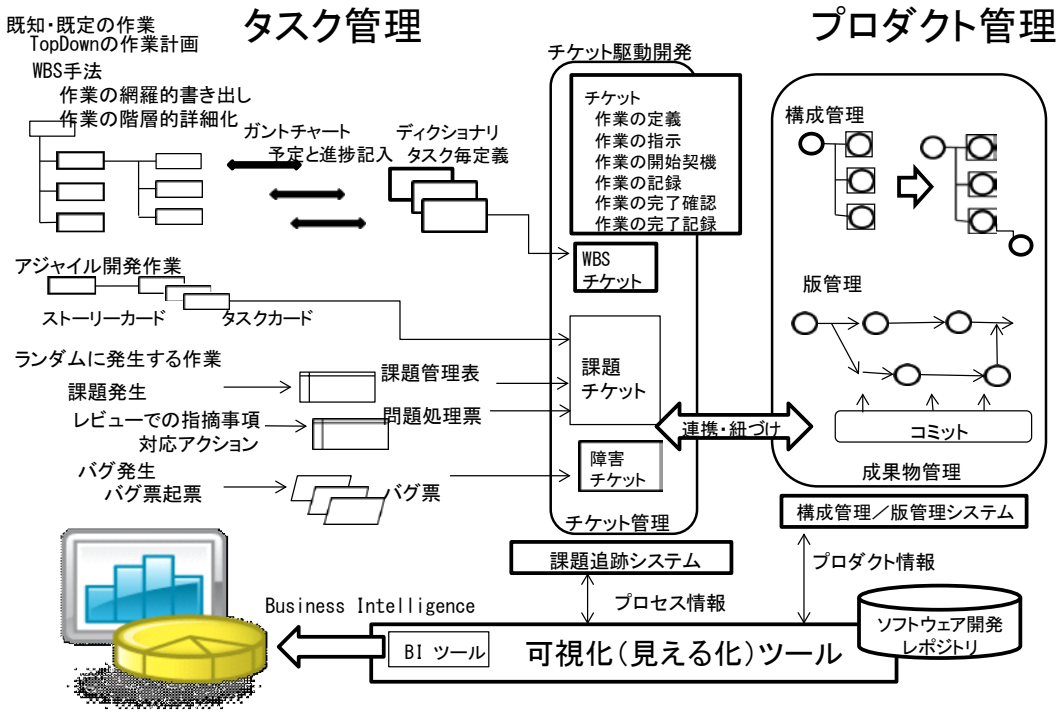


図 3-2-5　統合的なソフトウェア開発管理環境の構成像

1）新しい統合型ソフトウェア開発管理環境の組み立てと実プロジェクトへの適用

ソフトウェア開発に関する近年の進化した技術を集積して，図 3-2-5 に示すような環境を商用のクラウド環境の中に構築し，実際の開発プロジェクトに供して評価することができた意義は，ソフトウェア開発環境の一つの姿を具体化したものとして大きい．

その特徴は産業界で一般的なビジネス・システム，企業のいわゆる業務情報システムと対比して考えると明らかである．産業界の業務システムでは，その事業の中のあらゆる計測可能なイベント，データがコンピュータ・システムに捕捉され，その中に格納・整理され，通信ネットワークによって運ばれて業務が進められる．いわゆる，「人，物，金」に関する状態と動きは隅々まで把握されている．これに対してこれまでソフトウェア開発に関しては把握できない事象やデータが存在し，BlackBox が多かった．また，ソフトウェアは芸術作品にも似て，その開発は定量的な把握に不向きという考え方も根強くあった．

しかしながら実現したソフトウェア開発管理環境の構成を図 3-2-6 のような一般的なビジネス・システムと比較すると，その枠組みには何ら相違がない．実現した構成ではソフト開発プロセスを捕捉するイベントは個々のチケット，あるいはそれよりもさらに細かいチケットのステータスであり,プロダクトに関してはソースコードなど生産物の一行一行,あるいはコミットごとの契機を捉えている．これらの粒度，分解能はビジネス・システムにおける各種の伝票処理，あるいは物理的な生産物の流れ，サービスの提供，そして金銭の支払いや受領の契機と比べてなんら遜色がない．

ソフトウェアは芸術作品にも似て，開発プロセスの計測や定量的管理に馴染まないという指摘があり,確かにそのプロセスには人間依存性の高いある種の BlackBox があるのは事

実である．しかしながら，それはビジネス領域でもあまり変わらない．商取引において，商談の成立の成否，発注するか，受注できるかどうか，といった事象は極めて人間的な活動であって，その内部のプロセスを定量管理することは難しい．しかしながらその人間的な行為の結果のトリガを捕捉することで，産業界のビジネス・マネジメントはある程度円滑に進められている．

今回の統合型ソフトウェア開発管理環境の実現と適用で，ソフトウェア開発の領域においても，少なくとも一般的なビジネス・システムの領域と同程度には，プロセスを把握してマネジメント可能な枠組みの実現が実証されたと言える．

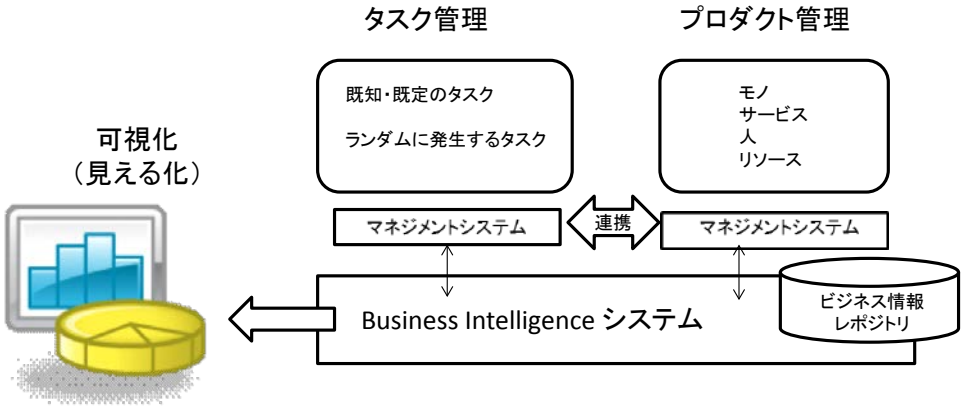


図 3-2-6　一般的なビジネス・システムの構成

2）統合環境の機能評価に関するケーススタディ

実現した開発管理環境の実開発プロジェクトへの適用で，得られた可視化グラフはごく一部で，想定した可視化機能は発揮されなかった．定量的プロジェクト管理ツール EPM-X の機能は，使用されるチケット構造，チケット記載項目と，計測・可視化機能が対になって構成されている．今回はチケットに設定される情報が少なく，計測・可視化機構と整合しない部分が有り，十分な可視化情報を得られなかった．ここから新しい基本的な課題が浮き彫りとなった．

定量的プロジェクト管理ツールで今回使用した Redmine/Subversion 利用者向け標準サンプルのインストール・キットでは，一定のチケット構造を想定している．すなわち工程毎に定義される WBS チケット，そしてこれに紐づけられる課題チケットと障害チケットである．たとえば障害チケットには，記入する原因分類のプルダウン選択肢がある．ここには仕様確認漏れに始まる原因分類が 21 種提示され，分類コードまで付属している．そしてこれらの記入情報から多彩な情報を管理することを想定している．管理項目数はトラッカーから原因分類まで 31 種にのぼる．

EPM-X の標準サンプル・キットでは想定するチケット構造と想定する情報項目が記入される場合，それらの情報を組み合わせて，多彩な可視化情報をグラフで表示する機能を装備している．具体的には WBS 管理，品質管理，障害管理，課題管理，要員負荷管理の 5 つの管理系で 14 種のグラフを表示できる．WBS，障害，課題の 3 種のチケットに記入された

情報を組み合わせたグラフや，チケット記載情報と版管理システムからのソースコード行数の情報，そして工数の情報を組み合わせたグラフがある．

これに対して今回の適用プロジェクトの管理者および開発者の興味は次のようであった．チケット構造は，発注者の要求もしくは実現する機能を中心に考えたフィーチャー型である．図 3-2-7 にその構造を示す．今回の適用では，ここをサンプル・キットの設定に合わせて WBS 型で運用した．

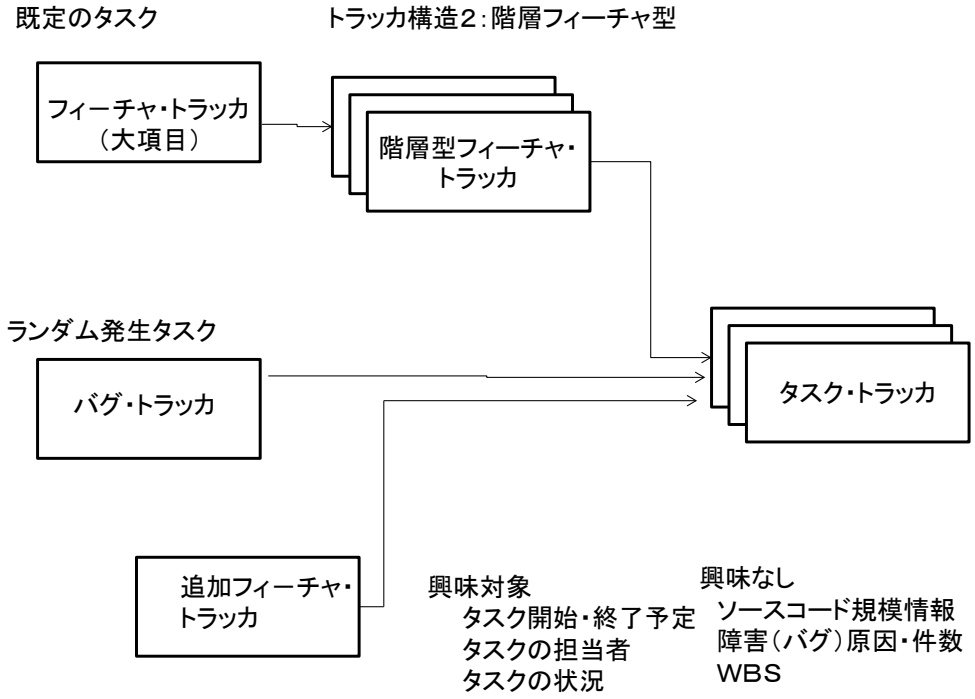


図 3-2-7　フィーチャー型チケット構造

適用プロジェクトのマネジメント上の興味は短期的なプロジェクトの単独の遂行で，次のような項目である．

- タスクの開始・終了予定
- 進行中のタスクのステータス（例：担当者によるコード修正終了．管理者によるタスククローズ）
- 障害チケット発行時の障害状況，障害タスク遂行中のタスク状況
- 課題タスク発行時の課題の状況，課題タスク遂行中のタスク状況
- 一方，次のような項目には興味がなかった．
- WBS．作業はあくまでも要求や機能に対して行うもので，スケジュール消化ではない．
- ソースコード規模，規模推移．あくまでも機能を実現するのであり，結果としてのコード規模には興味がない．
- 障害原因の分類，その記録，その集計．障害チケットにはフリーハンドで障害状況を記述する．

・ 課題の分類, その記録, その集計. 課題チケットにはフリーハンドで課題を記述する.
・ 消費工数.

表 3-2-2 に適用プロジェクトでの管理項目，別途ヒアリング調査で別の企業から聴取した関心のある管理項目を示す.

表 3-2-2　チケットで管理を希望する情報項目

	管理希望 （〇:管理したい／×:不要）	投入しているか （〇:投入している／×:投入していない）	ヒヤリング調査企業の意見 （〇:管理したい／×:不要）
題名	〇	〇	〇
予定工数	〇	×	〇
開始日	〇	〇	〇
終了日	〇	〇	〇
優先度	〇	〇	〇
ステータス	〇	〇	〇
担当者	〇	〇	〇
重要度	×	×	×
試験数計画	×	－	×
試験数実績	×	－	×
SLOC計画値・実績値	×	×	×
想定バグ密度	×	×	×
開始日(実績)	〇	×	×
終了日(実績)	〇	×	×
試験計画項目密度指標値上限値	×	×	×
試験計画項目密度指標値下限値	×	×	×
グループ	×	×	×
原因分類	×	×	×
WBS番号	×	〇※	×
関連チケット	×	×	〇
WBSチケット	×	〇※	×
		※EPM-Xを動作させるため やむを得ず入力	

このように，クラウド環境上に実現された統合的なソフトウェア開発管理環境は，開発管理の枠組みとしては関係者の物理的な所在を越えて充分に機能したが，実際のソフトウェア開発現場に対してはサンプル・キットを用いたそのままの形では想定した機能を発揮できないということ，そしてその第一次の要因の所在が明らかになった.

3）統合開発管理環境の活用法に関するケーススタディ

1 ケーススタディではあるが，実現したクラウド型開発管理環境を実プロジェクトに適用することで，その機能を想定通り発揮させるためには，個別のプロジェクト・マネジメント方式を反映したチケット構造に呼応した計測・可視化機能の設定が必要なことが明らかとなった. プロジェクト・マネジメント方式の個別性は高く，これまでのソフトウェア・エンジニアリングの成果を集積し，環境導入の容易化を図った標準サンプルのインストール・キットは今回の開発現場には受け入れられなかった.

標準サンプルのチケット構造には，次のような機能が含まれている.

・ 作業の定義
・ 作業の指示
・ 作業の予定（開始：終了時期）
・ 作業の状態（ステータス）記録
・ 作業の完了確認
・ 作業の完了記録

・ 作業の実績記録（開始・終了時期）

このうち今回のプロジェクト・マネジメントでは，作業の実績記録に興味を持たなかった．一方，EPM-X のサンプル・インストール・キットで出力する多くの可視化情報は作業の実績記録の投入を想定していた．

ここにチケットを用いたプロジェクト・マネジメントに関する 2 つの考え方の相違が顕著となった．一つは，チケットを「作業指示の受け渡しと完了の確認」に使用するもの．もう一つは，「これに加えて，作業の属性や結果を集計可能な形に分類して記録し，その集計状況を逐次プロジェクト中で活用しようとするもの」である．また「その集計結果を蓄積し，プロジェクトの評価や，プロセス改善，そして次のプロジェクトの運営に役立てよう」とするものである．いわば前者が「作業受け渡し型」で後者が「作業属性分類・記録型」とも言える．

今回の実適用プロジェクトでは，この 2 つのマネジメント方式が混在し，新しい開発環境の仕組みは意図通り機能したものの，プロジェクトの計測と可視化については当初の想定どおり機能しなかった．そこから次の結論が得られる．

- チケット駆動開発，版管理・構成管理システムによるプロダクト管理，プロジェクトの自動計測と可視化機構を組み合わせた統合型開発管理システムでは，チケット構造，記入項目と計測・可視化機構を整合させた設定が必要である．
- チケットを活用したプロジェクト・マネジメントには「作業受け渡し型」と，「作業属性分類・記録型」の 2 つの方式がある．またマネジメントの流儀として，「WBS 方式」と「フィーチャー方式」がある．統合開発管理システムの利用ではチケット構造の設定と計測・可視化機構の設定にあたって，これらの考え方を反映しておく必要がある．

② 組織間協業機構に関する研究成果

1）地域コミュニティ企業群へのヒアリングの成果

組織間協業に関するクラウド環境の可能性に関しては，上記の環境の適用状況を素材として提示しながら地域企業群へのヒアリング中心の活動となった．ヒアリングは現場技術者のほかに，中堅マネージャ，管理者，経営者を含んで実施できた．結果として期待したほどネットワークを活用したサービスイメージを深めることはできなかった．

明らかになったのは,島根県を中心としたソフトウェア企業の Ruby 活用グループは非常に勉強熱心で,かつ新しいことへのチャレンジ精神に富んでいる.これを応援する人々（企業）もいる．こうしたコミュニティの当面の目標は，情報交換や切磋琢磨から地域としての技術力を高め，ブランド的に磨いて，受注につなげてゆく，ということである．

利害関係者への説明性と言うことでは，やはり得意先，受発注関係者への説明と言うことが考えられる．利害関係者が地域的に広がっている例もある．地域でも中国地方全域に広がり，また東京などからの仕事もある．ソフトウェア係争のようなことはない．組み込みソフトウェアの一部にコンシューマ製品への組み込みがあるが，これに関連したソフトウェア係争は経験がない.こうした状況で,本委託研究で提唱するようなクラウド環境は，当面は地域企業群の技術力向上に貢献し，受注増につなげてゆくのが実際的であるということが明らかになった．企業ヒアリングでつぎのような意見を得た．

[1] A 社

・経営者

今回のような試みについて：

PMO を置いたり，全社プロジェクトの面倒を見て，生産性や品質などを考えたりする部署のある企業では，SEC の狙うような技術が有効であろう．大きなプロジェクト，ある程度の期間を要し，途中でレビューしたり，その結果をフィードバックしたりするようなプロジェクトでは有効であろう．そうでない，短い，小さなプロジェクト，自分のプロジェクトの完遂だけを考えているところでは難しいのではないか．開発するシステムや環境は千差万別であり，そもそも障害原因を分類し集計することに意味があるのか疑問である．一方，運用やサポートの場合では原因分類は参考になると思う．かつて，ソフトウェア開発ツールや環境，プロジェクト・マネジメントの方式を社内標準化しようと考え努力したことがあった．いまでは，その考えはもっていない．ソフトウェア開発プロジェクトには様々な形があるので，いろいろな開発法，管理法，環境があってよいと思っている．プロジェクトの特性にあわせていろいろ工夫することがあってよいと思っている．KKD（勘と経験，度胸）によるマネジメントというが，各リーダーはステップ数や生産性，工数などは，経験的にデータを持っていて，いろいろ判断している．但し，個人差により判断の基準がバラバラであるので，一定レベルまではチケットによる作業管理は必要であると考える．

コミュニティについて：

Ruby のコミュニティの人々は非常に研究熱心で，いつも新しいことにチャレンジしている．そういうマインドなので，このコミュニティを支援している．Redmine はおそらく，海外での使われ方と日本での使われ方は違うだろう．日本の開発管理は諸外国より非常に進んでいる気がしている．Redmine もおそらくそうした日本の特性にあわせた進んだ使われ方をしているに違いない．地域のコミュニティの技術力を上げ，ブランドになって受注につながればよい．

・中堅マネージャ

社内のプロジェクト・マネジメントには WBS のグループと Redmine のグループがある．WBS のグループはスプレッドシートによってプロジェクトを管理している．Redmine のグループは，WBS は使わない．フィーチャー型のチケット構造を使う．バグ原因の分類や記録は行わない．ソースコード行数には興味がない．タスクの開始，終了時期の実績には興味がない．Redmine を使うグループは，設計工程はスプレッドシートなどを使い，実装工程で Redmine を使っている．

現状では各社どころか同一社内でも各プロジェクトによって管理ツールは異なっている．それぞれがそれぞれの特性に応じて，或いはマネージャにとって使い易いツールによって管理されている．この統一を実現するためには，統一ツールによるプロジェクト事例を用いるなどして，企業文化に根付かせる事が必要ではないだろうか．

説明性については，取引先関係への説明が考えられる．ソフトウェアに関する係争はないし，係争を心配するようなことはない．

EPM-X は大きなプロジェクト向けと考えられるが，大きなプロジェクトで全メンバーに使い方を徹底するのは難しいだろう．

[2] B 社

・上級管理者

プロジェクトのマネジメント方式はグループ毎に異なる．全社横通しの，障害管理，生産性管理はない．自社のみで開発する案件には Redmine を適用しつつある．

EPM-X は系列上位の企業が元請けで獲ってくるような仕事を分担するような時に有用と思われる．

説明性に関しては，取引先への説明が考えられる．係争に関係するようなことはない．

・実務者

Redmine を使用中である，チケット構造はフィーチャー型　WBS は使わない．トラッカーを追加している（現象，機能，バグ，サポート，検証，設計）．バグ原因の分類はグループ内でおこなっている場合があるが，集計できるような環境になっていない．全社統一のバグ分類はない．ソースコード行数に関心はない．

旧来の WBS 型管理から Redmine へ移った感想は，とにかくプロジェクトが楽しくマネジメントされて進むということだ．

[3] C 社

・プレイングマネージャ

受託開発（業務システム）と自社製品組み込みソフトがある．発注主が遠隔地（例：岡山）のこともある．主に中国地方である．ソフト開発プロジェクトの期間は，数ヶ月程度で数年というのはない．開発は，アジャイル型ではない．チケット構造はフィーチャー型チケット構造はグループ毎に定め，社内標準はなく，頻繁な変更はない．主にチケット件数を計測している．バーンダウンチャートがほしい．バグ分類，バグ原因記録はない．バグ票（チケット）のバグ記述はフリーフォーマットである．バグの分類や集計は行わない．

開発者はチケットに記述された作業が終わるとチケットのステータスを「解決」状態にする．「解決」となっているチケットを管理者が確認した上で「完了」にしている．作業の完了日はチケットに表示されるコミット日時で把握している．ソースコード行数の積算はない．コード行数をベースとしたメトリックスは使っていない．

説明性について：

地域的に離れた発注主との間の説明が必要なことがある．係争例はない．一般消費者向け製品もあるが，特別な説明例はない．

[4] 適用システム，元請け企業

SEC からサンプルで提供されている各種の可視化グラフの利点は全く理解できない．これらのグラフを得るためのチケットの入力は非常に煩わしい．サンプルで提供されているチケットは，Redmine を使ってはいるが，本来の Redmine の使い方とは全く別モノに見える．WBS をベースした EPM-X の標準サンプル・キットの機能を Redmine をベースとしたツールであると説明して導入すると，Redmine との違いの大きさに戸惑うのではないかと思われる．サンプルでは WBS から始まっているが，WBS は使わないし，このような作業法はこの方式の適用普及の障壁になる．本来は，「フィーチャー」，「機能」，あるいは「要求」

を示すチケットから始める．一般にRedmineのユーザがイメージする使い方は，フィーチャー型で，軽く，デフォルトのチケットに「開始時期」「終了時期」の「実績」記入欄はなく，チケットのバーンダウンチャート程度で管理を行う，といったものである．ソースコード行数には全く興味が無い．興味の対象は，どれだけの機能を作ってゆくかということで，コード行数とは関係ない．障害チケットに障害区分を記録することには興味がない（あとから障害原因を分析したりしない）．メトリックスとしては，消化チケットのバーンダウンチャートがほしい．EPMのサンプルでは，直接このバーンダウンチャートが出ない．Redmineの機能でも出ない．

EPMのツールや考え方の普及のためには，まず，サンプルにあるような可視化グラフの効用を啓発することが重要だろう．そののちに，そのようなグラフを得たいと考えるようになり，そのためのチケット項目入力が理解されるようにする必要がある．出力の利点が理解される前に，チケット項目の入力を求めても普及は無理である．

コミュニティでのクラウドの活用は，なかなか難しい．クラウド環境は，現在，各企業個別に使われているだけである．また，松江市，島根県，などのコミュニティは，現状は情報交換の域を出ていない．ビジネスに（受注に）直結する形のコミュニティ活動は難しい．

地域コミュニティでの活用としては,受注につながる活動ができればいい.地域のコミュニティは企業同士の交流の場でありそのコミュニティ自体が受注活動をするわけではないので，コミュニティでRedmineを共有するというのはイメージがわかない．

[5]ファーエンドテクノロジー社：

・代表

Redmineは世界の中では日本で一番多く使われている.EPM-Xがデフォルトでサンプル提供しているWBS型のチケット構造は,一般的なRedmineの使い方とは異なるように見える．Redmineの仕組みを使った全く別のものとして説明してゆく必要がある．ファーエンドテクノロジー社のクラウドサービスでEPM-Xを商用サービスすることは現段階ではリスクが大きすぎると考える．サポート負荷が大きいと想定される． WBS型のいわゆるソフトウェアエンジニアリング手法を普及させるには，道具立ての前に，精力的な啓発活動が必要である．まず，提唱する手法の利点を感じてもらう必要がある．

プロジェクトの計測や可視化の仕掛けがあって,こんな可視化ができます,どうですか？というアプローチではなく，プロジェクトの中では，こんな難しい局面があります，そのときの，このような計測・可視化の機能を使うと，こういう利便があります，という説明のほうが良い（こういう説明が是非必要だ）．いろいろ可視化機能を見せられても，その利便はなかなか理解されないのではないか．

・コンサルタント

大学で管理学を専攻し，その後企業の工場総務として小集団活動の事務局や，工場管理のコンサルタントの経験があり,製造現場での20世紀初頭からの科学的アプローチによる管理手法の歴史的変遷や現在の企業での導入について，若干の認識がある．その視点からソフトウェアのプロジェクト管理の「見える化」については大変興味深い．結論として，IT開発の世界でも「管理手法の標準化」と「見える化」は浸透させなければならないもの．

そして，競争力や付加価値を生み出す基盤になると考えられる．しかしながら，現状では多くのソフトウェア開発が小規模・小ロットの仕事で成り立っていて，ちょうど中小零細の町工場と似た様相であると考えられる．中小零細の町工場では，「見える化」や「作業の標準化」は経営者にとっては競争力と付加価値を付け，自社の売上向上と生き残りを実現させる非常に魅力的な手法である．しかし，現場の作業者から見れば「面倒」「3S や 5S の必要性がわからない」「今の作業方法は先輩や自分が経験で作ってきたものなので間違いない」と拒絶反応を示され，なかなか受け入れられない．そこで，「見える化」や「標準化」によって「自分たちの仕事が楽になる」「会社が儲かれば自分たちの給与が上がる」ということを説明するのに時間を費やしているのが現状である．

今回提示されたようなソフトウェア開発プロセスのグラフは大変興味深い．これを多くの方に感じてもらうには，先に問題事例を挙げて，その問題の発生要因にはこのような傾向があるという帰納法的アプローチでグラフを示すことができれば良いのではないかと感じた．

2）オープンソース連携製品の維持管理の課題

本委託研究のクラウド環境の活用に関する検討の中で一つの課題が明らかになった．今回の適用システムの開発対象ソフトウェアは,社内利用ソフトウェアの受託開発であるが,それはオープンソース製品と連携して動作し，かつ内部に別のオープンソースとして提供されている製品を取り込む方式を採用している．このように，今日の企業ソフトウェアには多様な形態でオープンソースの製品が多く含まれるようになり，オープンソースなしに企業システムの構築は考えられなくなった．企業システムに含まれるオープンソース製品の形態は様々である．コピーしてきてそのまま独自の版管理となってゆくもの，常にオープンソースの最新製品を取り込むようになっているもの，他の環境にあるオープンソース製品をサービスとして使いにゆくもの，など様々である．そしてその維持管理が大きな課題となる．

この課題に対してクラウド型のソフトウェア開発管理環境の活用が考えられる．現在,版管理システムの Git を核としたソフトウェア開発者用の共用ネットワークサービス GitHub が提供されている．提供元（GitHub 社）の本拠地は米国であるがサービスは全世界で広く利用されている．このサービスはクラウド型の Git 搭載サーバを中心に，オープンソース製品を様々な開発グループが成長させてゆく開発と提供のためのコミュニティ基盤を実現している.開発者は関心をもったオープンソース製品を GitHub センタから環境毎自分の環境にコピーし，適宜改良，追加開発を行ったのち，一定のルールでもとの GitHub センターに戻し，追加・改良部分をマージして，皆に提供することができる．

本委託研究では，こうしたオープンソース製品と企業あるいは企業グループ内に閉じて開発・維持する固有の機能との組み合わせ製品の維持管理環境，そして説明性，追跡性の要求に応えられる環境確保の課題が明らかになった．

3.2.3 実用化へ向けた課題と問題点

(1) 課題と問題点

① クラウド型開発管理環境に関する課題と問題点

実現したクラウド型の統合開発管理環境は，地域的に広がる複数の企業，松江，東京と距離のある関係者間をむすんで充分に機能した．しかしながら，これまでのソフトウェアエンジニアリングに関する経験を集積し，これを簡易に利用することを可能にした標準サンプルのインストール・キットは現場の条件ではそのままでは機能せず，実プロジェクトには全く受け入れなかった．

ソフトウェア・マネジメントの考え方はプロジェクトの文脈に沿って多様で，その考え方ごとにこれを反映したチケット構造と記入項目，そしてこれを反映した計測・表示機能の設定が必要である．サンプル・キットはあくまでもサンプルであって，直接多様なプロジェクト・マネジメントの考え方を充足するものではなかった．

一方，各開発プロジェクトの現場における既存のプロジェクト・マネジメントの考え方は，必ずしも，これまでの長年のソフトウェアエンジニアリングの経験，研究成果を反映しているわけではない．そこで，現場においては，実現した開発管理環境やサンプル・キットに込められたこれまでの成果への歩み寄りも重要な課題として浮上した．

② 組織間協業機構に関する課題と問題点

今回研究対象とした地域の産業コミュニティを構成する企業群では，クラウド環境の利便を活用するような特段の説明性や追跡性へのニーズは明らかにならなかった．一方，先進の開発管理環境を活用した技術力の向上を通して，地域の産業力のレベルアップを図り，その評価の向上，さらには地域産業のブランド化を図るニーズがあった．

実現したクラウド型の統合開発管理環境は近年の開発管理技術の格段の進歩を反映しており，またこれまでの経験と研究の成果を内包しているので，その活用が現場技術力の向上に貢献することは充分に期待できる．地域の産業界でこれを現場の条件に即した形で咀嚼し普及・活用して行くことが課題である．

(2) 将来の応用方法

① クラウド型開発管理環境の応用方法

実現したクラウド開発管理環境はさまざまなソフトウェア開発環境に適応し，幅広く活用されるようになる．その活用にあたって，まずプロジェクト・マネジメントの考え方を反映したチケット構造，チケット記入項目を設定する．このとき，サンプル・キットの出力グラフなどを参考に，追加の項目について検討し，実際にチケットを投入する開発現場とコンセンサスをとりながら，チケット記入項目の調節・増減を行う．こうして設定したチケットに合わせてデータ計測・表示機能を設定する．次の段階として，こうしたチケットを評価し，チケットデータ，プロジェクト文脈情報，プロジェクト評価を対にしてライ

ブラリ化し，オープンな世界，あるいはそれぞれのコミュニティ，企業グループで再利用を可能とする．

② 組織間協業機構の応用方法

組織間協業機構の応用については，普及に向けてより幅広い実証活動の積み上げが必要であるが，現状でも，クラウド環境に蓄積された可視化データへの適切なアクセス権の設定で，様々な取引先企業，協力会社群との開発に関するコミュニケーション向上，コンセンサスの早期形成に役立つと考えられる．また，こうした環境の応用事例を産業コミュニティ内で共有することで，その技術力向上に役立てることができる．

４．考察

4.1 研究により判明した効果や問題点等

① スナップショットの効果・問題点

ソフトウェアプロジェクトを表現するに十分な情報を含むスナップショットが実現可能であることを示すことができた．また，本委託研究で開発した「プロジェクト構造モデル（ＥＲ図）」が，ソフトウェア品質の第三者評価の技術基盤の一つとなり得ることは，「3.1.2 研究プロセスと成果 (2)具体的な研究成果の内容」で示した成果の数々より明らかであると自己評価している．特に，プロジェクト構造モデルを開発するにあたり，ソフトウェアプロジェクトの全体構造を「要件」，「作業」，「組織」，「プロダクト」，および，「課題」という５つの構成要素とそれらの間の関係で表すとした点が，本委託研究の独創性・新規性の一つであり，成果を上げることのできた大きな要因の一つと考えている．類似研究は少ないが，例えば，Sarma らは，ソフトウェア開発におけるプロダクト，組織，バグ，コミュニケーションなどに間に存在する Socio-technical な依存関係を見るためのブラウザを提案している[Sarma]．ブラウザに表示される情報は，本委託研究におけるスナップショットと共通する部分も多いが，要件に関する情報は含まれておらず，提案においてもその扱いは言及されていない．

以上の通り，ソフトウェアプロジェクトを５つの構成要素，あるいは，５つの視点で捉えることは妥当であると考えているが，ここでは，別の可能性について少し考察を加えてみる．

これまで示してきたものとは少し異なるプロジェクト構造モデルを図 4-1 に示す．図 4-1 で示すモデルでは，ソフトウェアプロジェクトの５つの構成要素のうち，「要件」と「課題」が一つのエンティティ「要件／課題」にまとめられている．また，それに合わせて，エンティティ「要件操作」とエンティティ「課題操作」もエンティティ「要件／課題操作」に置き換えられている．図 3-1-1 で示した「プロジェクト構造モデル」を見るとわかるが，エンティティ「要件」・「要件操作」という組と，エンティティ「課題」・「課題操作」という組は，要件と課題という字面の違いを除けば，構造や属性は同じであり，同モデルのコアとなるエンティティ「実施」との間の関係も同じである．したがって，この二つの組をひとつにまとめてしまうことは，ER 図としては特に不都合はない．ただし，ソフトウェアプロジェクトの全体構造を捉える上で，要件と課題を同一視することになる．

要件と課題を同一視することには無理があるようにも思えるが，図 4-1 に示したモデルでは，次のように考えていることになる．

> 「要件も課題も，ソフトウェアを完成させるために実現・解決すべき事柄である．両者の違いは，要件は開発当初に（必然的に）与えられているが，課題は開発途中で（偶発的に）発生する，という点である．」

プロジェクト構造モデルを構築する上で，要件と課題を同一視する必要はないかもしれない．同一視することで不都合が生じる可能性もある．ただし，解析や可視化の検討において参考にすべき議論である．例えば，要件と課題の数や内容を対比させることで．これまで以上に有用な解析・可視化結果がもたらされる可能性がある．

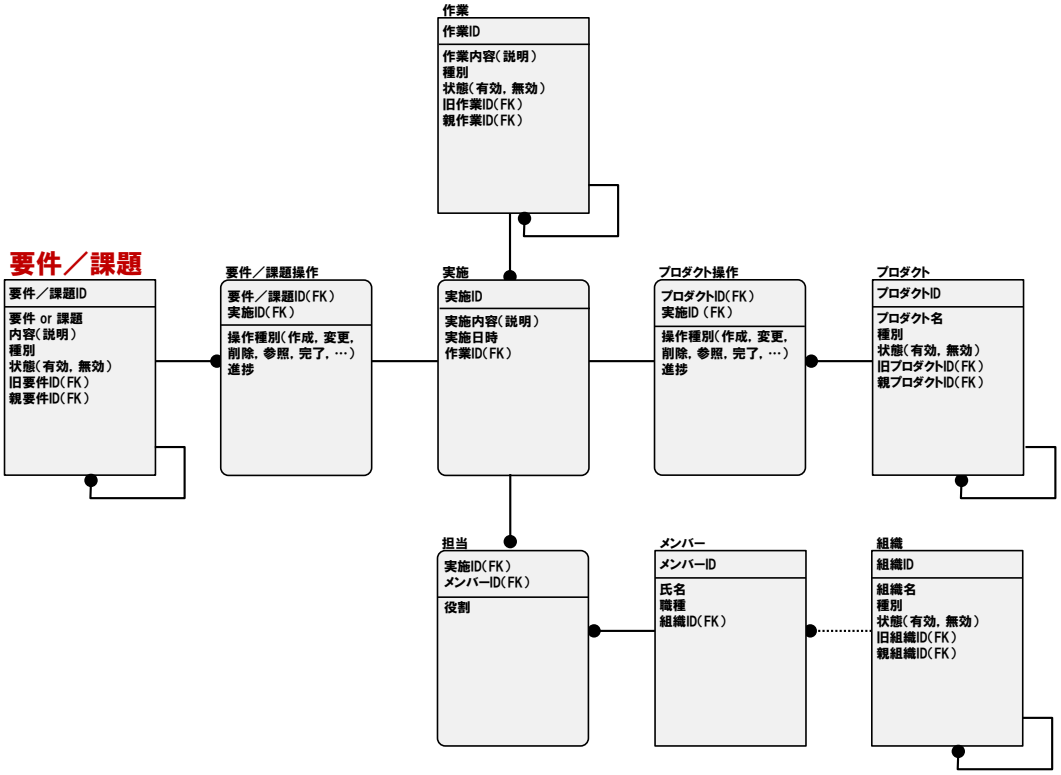


図 4-1　プロジェクト構造モデル（別バージョン）

② スナップショットの生成・解析．可視化方式の効果・問題点

1）生成方式

[1] 要件ペイン

「非機能要件評価表」に基づいたスナップショット生成に要した時間は，Web 上に公開されている RFP のページ数が最も多いもので 600 ページであり，RFP 1 件あたり 30 分から 1 時間程度であった．このことから，経験を持った技術者であれば，600 ページ程度の RFP については，現実的な時間で生成できることが分かった．しかし，情報化対象となるドメインにより RFP や要件定義書のページ数が数千ページになることもあり，実際の評価作業を人手によって行うことは困難であることが推定される．一方では，初心者による評価は難しいこともわかった．本委託研究により非機能要件を非機能特性として段階的に評価した結果として把握することが可能となった．

[2] 作業ペイン

ケーススタディにおいて，自動計測ツールによる開発タスクの計測結果に加えて，ソフトウェア開発者自身が開発タスク実施時間を推定した結果，実際の時間よりも推定時間の方が過大となる傾向にあることが分かった．このことは，たとえ開発者に悪意がなくても，作業日報などの人手による作業時間の記録は誤差を含むことを意味している．このことか

ら，ソフトウェア開発作業の第三者を行うにあたっては，開発タスクの自動計測により得られる客観的なデータが重要となることが示された．

一方，課題としては，開発者ごとに設定ファイルを記述する必要があり，大規模プロジェクトでの運用は現時点では難しいことが分かった．タスクとアプリケーションのマッピングについて，様々な開発環境に対応可能な汎用的な設定ファイルの作成について今後検討する必要がある．また，3.1.3 節でも述べたように，現在の計測ツールの制約上，計測結果が常時，被計測者（開発者）にも見えてしまうことは問題である．

[3] 組織ペイン

メーリングリスト等のコミュニケーションデータから，プロジェクト開始当初には規定されていなかったような，実際の組織構造を可視化・計測できることが分かった．ただし，開発現場で行われる対面でのコミュニケーションとその効果（あるいは影響）がコミュニケーションデータにも反映されているとは限らない．国際会議 ICSE では，コミュニケーションデータから構築したネットワークと，開発現場の実務者が感じている認知的な組織構造はほぼ一致しているという結果が報告されている[Meneely]．ただし，摺り合わせを重視する我が国のソフトウェア開発にも当てはまるかどうかは不明である．この点については，今後，我が国のソフトウェア開発企業のデータを用いた実験を行い，組織ペインから分かる組織構造やその特徴と，実際の現場での実感とがどれ程一致しているかを実務者等からのインタービューを通じて明らかにする必要がある．

一方，近年我が国においても盛んにおこなわれているオフショア開発では，コミュニケーションの手段として電子メール等が活用されていることから，開発の委託先がどのような組織構造となっているか，その中のキーパーソンは誰か，などを委託先が把握できるようになる可能性が高く，課題ペインが効果的なプロジェクト管理支援ツールとなることが期待される．

[4] プロダクトペイン

複数のオープンソースソフトウェアを対象に適用した結果，品質に問題のあるバージョンを特定することができたが，企業への適用は実施できていない．

[5] 課題ペイン

低品質モジュールの生成要因解析や予測の効果は，課題票と課題解決のために変更されたソースコードの変更記録が正しく記録されているか否かに依存する．

2) 解析方式

カリフォルニア大アーバイン校が提供する大規模ソースコード集合を基に品質メトリクスの基準値データを作成した．しかし，現状では Java 言語で記述されたソースコードのみを対象としているため，Linux ディストリビューションや FreeBSD の ports 等から適切にソフトウェアの選択する方法を考案し，C/C++の基準値データの作成をする必要がある．また，ソフトウェア開発企業と連携し，テストカバレッジ等の今回は扱わなかったが頻繁に用いられているメトリクスの定義を調査し，同様の基準値データの収集を行う必要がある．

低品質モジュール予測については，Eclipse プロジェクトを対象とした適用実験を行い，開発終了の 9 ヶ月前でも，開発終了時と同程度の精度で予測可能であることが確かめられた．ただし，Eclipse プロジェクトは，（個々のバージョンの開発期間は 1 年程度である

が）プロジェクト全体としては長期にわたって実施されている．バージョンのリリースが繰り返された結果，モジュールの特性が安定し，早期に予測しても精度が大きく下がらなかった可能性がある．これまでにリリースされたバージョンが少ないプロジェクトなどで同じ結果が得られるかどうか検証する必要がある．

3）可視化方式

[1] 検証型可視化技術

検証型可視化技術の一つとして，スナップショットに基づいて低品質モジュールを可視化するシステムを試作した．オープンソースソフトウェアの開発データを用いた適用実験に結果，試作システムにより，ソフトウェア品質の内的妥当性を評価するプロセスの支援が可能であることが確認された.今後は,内的妥当性のより詳細な評価を可能とするため,ソフトウェア開発の実務者を被験者とした検証実験などを行う必要がある．

[2] 探索型可視化技術

オープンソースプロジェクトの課題データを用いた仮説生成実験を行った．今後は，スナップショットが提供する 5 つのペインのデータすべてを用いて，プロジェクトおよびプロダクトの異常検出や品質評価が行えるかどうかを，ソフトウェア開発企業のデータを用いて行っていく必要がある．特に，第三者評価の観点から探索型可視化技術の有用性を検証するために，大学の研究者等が生成した仮説と，実際に開発に携わった管理者・開発者などの実務者が生成した仮説がどの程度一致するかを確認することが求められる．

③ 「ソフトウェア品質の第三者評価」の妥当性評価実験の効果・問題点

妥当性評価実験により,スナップショットによりプロジェクトの解析・可視化・理解が,オープンソースソフトウェアの開発において容易になることは示された．評価実験をソフトウェア開発企業で行われている実プロジェクトで行うことで，その効果や適用範囲をより明確に示すことが可能となる．ただし，その実現のためには，連携へのモチベーションが企業側に高まるよう，新たな取組みも必要となる．例えば，企業には，プロジェクトの時系列データを組織品質の評価に用いたい，というニーズがあるが，組織品質の標準値・基準値を設定するために，評価実験やそこで得られた知見を企業や大学の間で共有するしくみを検討する必要がある．

④ クラウド型開発管理環境の効果・問題点

1）次の段階への示唆

今回のケーススタディで明らかになったプロジェクト・マネジメントの 2 つの方式，2 つの流儀から，今後の方向への重要な示唆が得られる．2 つのマネジメント方式と流儀はいずれもそのままでは十分ではなく，今回実現したクラウド型の統合開発管理環境の活用を契機に，歩み寄りを考えてゆく必要がある．

[1] チケット駆動プロジェクト・マネジメントの 2 方式に関する歩み寄りと止揚

「作業受け渡し型」方式は，チケット発行負荷が軽く，プロジェクト計測負荷をほとんど感じることのない利点がある．開発の実務者にとっていわゆる気持ちの良いプロジェクト運営が進められる利点がある．しかしながら，明らかに狭視野で，チケットの持つ記録・

集計可能性という機能を充分に引き出していない.単独のプロジェクトが進行すれば良い,という視点で，工程の前段の状況を分析して後段に反映したり，事後に評価してプロセス改善に反映したり，多くのプロジェクトのデータを蓄積して分析・評価し，将来にむけた施策に反映するという視点に欠ける.

一方，「作業属性分類・記録型」方式は，多くのデータを得られ，出力の可視化情報も豊かで，これまでのソフトウェアエンジニアリング研究の成果を反映したプロジェクト・マネジメントやプロセス改善の施策に役立てられるが,チケット発行負荷は大きい.また,管理先行でプロジェクト進行を実務者にとって負担が多く，予定消化至上のつまらない,あるいは苦痛を伴うものにする可能性がある．工程区分に結びつけた管理も再考の余地がある.

示唆される今後の施策として,プロジェクト運営に先立って,チケット構造と管理項目,得ようとする可視化データについて充分に検討し，すべての関係者間で合意を得る．その際に，すでにサンプルとして提供されている可視化データ，あるいは過去のプロジェクトで得られた可視化データを参考に，その有用性に関して関係者間で合意できる項目に関して，その出力に必要なチケット項目を設定する，という手法が考えられる．こうした作業をプロジェクト毎に繰り返し，チケット駆動によるソフトウェア開発プロジェクトの自動計測と可視化を洗練させてゆくのが今回の実証活動で示唆された一つの方向である.

[2] チケット駆動開発における 2 つの流儀に関する歩み寄りと止揚

本委託研究では，もう一つ別の視点から，チケット駆動開発における 2 つのマネジメント流儀，すなわち「WBS 方式」と「フィーチャー方式」の相違が顕著になった．すなわち,これまでのソフトウェアエンジニアリングの経験の成果を形にしたサンプル・インストール・キットが「WBS 方式」を想定し，実際に実証プロジェクトとして適用したプロジェクトが「フィーチャー方式」だった.

「WBS 方式」は，プロジェクト全体の作業をマクロに書き出し，それを階層的に細分化して，管理の容易な個々の「タスク」を定義する．タスクはかならずしも開発するプログラムの要求や機能に対応しておらず，複数の機能に共通する作業をくくりだしたものや,別の視点からの作業の定義もある．「タスク」には開始予定と終了予定があり，一般には予定工数がある．開始予定，終了予定が予定工期であり，予定工数の積算が見積もりやプロジェクト予算に直結し，その予実が管理される.

開発作業はこの定義されたタスクの内のりで，予定された工数の範囲内で実行される．ここが，管理者にはよくても，作業者には苦痛のもととなる．あるいは，管理者と作業者の一体感を阻害する要因となる．工期も工数もあらかじめ全体が見通せていることが前提にある.もちろんやってみなくては分からないことも多いから,タスクの中にある種のマージンを埋め込んだり，あるいは「リザーブ」と称して，予備のタスクを明示したりすることもある．しかしながら，「WBS 方式」はあらかじめ定めた所定の機能を所定の予算で所定の時期に実現するためにはある程度避けられない管理方法である．計画遂行に柔軟性を持たせるような工夫が試みられるが，開発作業の中に企業間の契約行為が含まれる場合,工夫の余地は少ない.

「フィーチャー方式」は，開発対象の「要求」「要件」あるいは「機能」の書き出しを出発点とし，これを階層的に詳細化して，管理の容易な「タスク」の定義に落とし込む．開発作業そのものでない作業も，いずれかの要求や機能にむすびつけて，関連タスクとして定義する．「フィーチャー方式」はあくまでも，定義された要求や機能を実現することに主眼を置き，各タスクには開始予定，終了予定，場合によって予定工数が定義されるが，その経過は積極的には管理しない．管理するのは定義したタスクの状態（ステータス）である．未着手，着手中，終了，などである．それぞれのステータスのタスクがいくつあるか，が管理される．「フィーチャー方式」は機能の実現に焦点をあてるため，一つ一つ機能を実現して行くことで作業者にとってモチベーションを高めやすい利点がある．一方管理者にとっては，工期や工数を管理しにくい難点がある．「フィーチャー方式」は所定の要員を確保し，要求された機能を分解して次々と開発し，開発終了次第提供して行く，といった開発に向いている．

「WBS 方式」と「フィーチャー方式」はプロジェクト管理の異なる流儀には違いないが，一つのソフトウェアのライフサイクルに着目すれば，両方式が混在せざるをえないのが現実である．最初の大きなリリースを，所定の期日に所定の予算で実現するためには「WBS 方式」となろう．その後，積み残し機能，新しい追加要求に対応する機能を所定の要員で次々と追加開発し，開発でき次第リリースしてゆく方式は一般には「フィーチャー方式」となる．2 つの流儀は，プロジェクト開始時の開発作業のとらえ方とプロジェクト進行中のマネジメントの着眼点に相違があるが，すこし高い視点から見れば歩み寄ることが可能である．プロジェクト開始時の考え方はいずれも階層化チケットとしてとらえれば，両方式を包み込んで考えることができる．あとは管理項目の相違である．これはプロジェクトの規模や経験の積み重ね状況にあわせて，軽装なものから徐々に精度の高いものへ，あるいは重装なもの，特に過剰な管理から徐々に必要を絞って，開発作業者にとって軽快なものへ歩み寄らせてゆくのが建設的である．チケット駆動開発において管理項目に関する管理の流儀の相違はアナログ的であって，二者択一ではない．

このように，今回実現したクラウド型の統合開発管理環境はこうしたプロジェクト・マネジメントの 2 つの流儀を一段階上の視点で歩み寄らせる媒体になりうる，というのが今回の実証活動から示唆された一つの方向である．しかしながら，同時に，実際に歩み寄る活動は，多くの実績の積み重ねが必要で，容易でないプロセスであることも想定された．

本委託研究で明らかになったのは，これまでの技術発展と研究成果を集積したクラウド型の統合開発管理環境は便利に，容易に利用可能となったが，その活用の前に，個別プロジェクトには幅広い，いわばソフトウェア・マネジメントのスペクトルともいうべきマネジメントの多様性が存在している，ということである．そこには，WBS によるマネジメントは当たり前でなく，1970 年代から広く普及してきた障害票への障害原因の分類と障害票への記録は受け入れられない世界があった．タスクの着手時期，終了時期の記録，ソースコード規模の把握に興味を持たない世界があった．

一方，今回実現した統合的な開発管理環境自体はこうした多様性を受け入れるメカニズムを装備している．実現した環境には，こうした多様な世界を歩み寄らせ，一般化する機能が内包されている．はじめに既存の管理項目をチケット化し，そこから自動収集機能による可視化データを得る．次にそれを評価し，より高度な可視化データの取得が発想され

る場合，これに合わせてチケット構成，記入項目を高度化させる．このプロセスを繰り返すことで，様々なソフトウェア開発の条件，文脈に沿った一定レベルのマネジメント水準に収束させて行くことが想定される．

また逆のプロセスも考えられる．プロジェクト・マネジメントが重装備の管理データ収集によって「管理のための管理」の状態に陥っていて，開発現場に喜びがなく苦痛のみを強いているような場合，まず本環境を活用して管理情報の自動収集と可視化を進める．そしてその可視化情報の必要性を精査し，より軽装な管理を発想し，これをチケット構造と記述項目に反映してその軽装化を図り，より軽快なプロジェクト運営を目指すことが可能と想定される．

⑤ 組織間協業機構の効果・問題点

クラウド型開発管理環境が持つ組織間協業への可能性については当初想定したような実証例を得られなかった．地域の産業界で活発なコミュニティ活動を展開している地域企業群へのヒアリングにより，地域の産業力を高め，一種のブランド化まで考えていることが判明した．今回実現した環境の地域産業界への普及はこうした活動に貢献し得る．

同時に，説明性，追跡性の確保を狙った組織間協業機構としてのクラウド型開発管理環境の有用性の実証，課題の抽出には，日頃からこうした課題に直面し，たとえばソフトウェア開発における係争の頻度の高い都市型，大規模開発企業の場での実証活動が必要と考えられた．

4.2 今後の課題

① スナップショットの課題

スナップショットにより，ソフトウェアプロジェクトの解析や可視化，および，それらに基づく「プロジェクト理解」が容易になることは，本委託研究における様々な試行・試作を通じて示すことができたと考えている．ただし，スナップショットの概念や構造が，プロジェクトを解析・可視化・理解を行う上で制約となったり，それらを非効率なものとした面があったかもしれない．今後は，本委託研究で実現した手法やシステムの側から，また，それらの適用実験の結果や得られた知見に基づいて，スナップショットを再検討し，ソフトウェア品質の第三者評価やその他の分野において，より有効で実用性の高いものとしていくことが考えられる．

② スナップショットの生成・解析・可視化方式の課題

1）生成方式

[1] 要件ペイン

初心者による評価（スナップショット生成）を支援するため，今後，評価の（半）自動化が必要となる．また，改造プロジェクトに対応可能な要件評価モデルについては，既存システムとの連携についての評価が必要である．さらに，実用化に向けては，民間企業との連携が求められ，民間企業により作成される RFP や要件定義書をデータとして用いた協

同作業によるソフトウェアプロジェクトの実証実験により要件ペインを生成し，リポジトリの有効性を向上していく必要がある．

[2] 作業ペイン

ケーススタディによって，開発タスクの計測結果そのものは役立つことが示唆された．今後，計測のための設定ファイルの準備を容易にしたり，開発者自身に計測結果を見せないような仕組みについて検討する必要がある．この点において，本委託研究で用いた計測ツール TaskPit は開発者自らが計測し，改善していくためのものであるため，第三者評価により適した物へと再設計および改良していく必要がある．

[3] 組織ペイン

組織ペイン単体で組織構造を可視化･計測できることが今回の実験で明らかになったが，組織構造と作業品質との関係，組織構造とプロダクト品質との関係など，他のペインと連動させることで，分析機能としての有用性・利便性をより高めることができると考えられるため，今後は，他のペイントの連動機能について検討する予定である．

[4] プロダクトペイン

企業が行うソフトウェア開発に適用するために，企業において頻繁に計測されているテストカバレッジ等のメトリクスの取り込み，および，各種メトリクスの自動収集・整理のためのツールセットの整備・公開が課題として挙げられる．

[5] 課題ペイン

本委託研究では，低品質モジュールを「欠陥（課題）が存在するソースコードファイル」とした．すなわち，モジュールの規模や欠陥（課題）の種類や数と関係なく，欠陥（課題）が一つでも存在すれば，低品質モジュールとみなした．しかし，ソフトウェア品質の第三者評価という視点からみると，規模あたりの欠陥（課題）数や欠陥（課題）の重要度などに基づいて低品質モジュールを議論することも必要と考えられる．

同じく低品質モジュールの生成要因解析では，内的妥当性の解析という位置づけで，同一プロジェクトで生成されたスナップショットのみを利用した．解析を十分に行うためには，全ての課題票を課題解決のために変更されたソースコードと紐付けることが必要である．しかし，ソースコードの変更履歴に課題票に関する情報が記載されていないことが多く，オープンソース開発のデータによれば，正確に紐付けできるのは 40%程度である．そこで，正確な紐付けが困難な場合には，外的妥当性（External Validity）評価のために作成したメトリクス基準値を併用することが考えられる．プロジェクトの個別性をどう評価するかなど技術的課題は多いが，基準値の活用が進めば，課題（欠陥）に関する情報を使わず，モジュール（ソースコード）の特性値のみで低品質モジュールの生成要因解析や予測が可能となることが期待される．

2）解析方式

外的妥当性と内的妥当性の観点から，解析方式における今後の課題について述べる．外的妥当性の評価をより詳細に行うためには，単に，外部，というだけでなく，どのような特性を持ったプロジェクトの集合であるのか，また，そこで作成された基準値や標準値の適用範囲をどう設定すべきか，といった議論が必要になる．外部プロジェクトの層別や対象ソフトウェアのドメインによる区別なども必要になるかもしれない．一方，内的妥当性

の評価をより厳密に行うためには，対象プロジェクトの当事者とのコミュニケーションパスを確保するしくみが必要となる．例えば，内的妥当性を確かめるため当事者へ個別にインタビューすることは，可能かもしれないが，高コストとなる可能性がある．いずれの課題についても，具体的なアプローチが明確となっていないが，ソフトウェア品質の第三者評価を普及・高度化する上で重要な課題と考える．

3) 可視化方式

可視化方式における今後の課題は，解析方式における課題と共通する部分が多い．外的妥当性や内的妥当性についての議論は，可視化についても当てはまる．加えて，外的妥当性と内的妥当性を可視化においてどのように使い分けるか，も今後の課題である．本委託研究では，ソフトウェア品質等の評価において，外的妥当性と内的妥当性を分けて考えたが，可視化においてそれらをどう使い分けるかについては，更なる議論が必要である．対象を外的妥当性と内的妥当性の両面から可視化する，というアプローチも検討に値すると考える．

③ 「ソフトウェア品質の第三者評価」の妥当性評価実験の課題

妥当性評価実験については，ソフトウェア開発企業と連携し，実プロジェクトにて行うことが今後の課題である．そのためには，

- ソフトウェア品質の第三者評価をシナリオとして具体化する．
- スナップショット生成・解析・可視化システムをリファレンス実装として公開する．
- 企業・大学間で評価実験や知見を共有するために情報の匿名化を実現する．

といったことが必要と考える．また，大学でのソフトウェア開発プロジェクトで評価実験を行うなど，「オープンソース開発」と「ソフトウェア開発企業における実プロジェクト」という構図に捉われない取り組みも必要と考える．

④ クラウド型開発管理環境の課題

今回試みたのは一地域企業の小規模・短期開発の文字通り一ケーススタディである．今後の課題は，実現した環境の実証範囲の拡大である．単に事例を増やすと言うことだけでなく，実証活動のパラメータと次の項目がある．

- 地域企業，小規模，短期開発（今回一例実施）
- 都市型，大規模，中・長期開発，社会的に大きな課題のあることが伝えられている官庁系システム開発
- フィーチャー型チケット構造駆動の開発と WBS 型チケット構造駆動開発の比較評価
- 実証的ソフトウェアエンジニアリング領域で重要テーマと位置づけられているグローバル開発への適用活動
  - 定量的プロジェクト管理ツール EPM-X の英語インタフェースの実現
  - 国際的に広く利用されている GitHub サービスと組み合わせた環境での実証活動
  - グローバル環境での産学連携実証活動の実現，国際研究者組織 ISERN (International Software Research Network) の活用

⑤ 組織間協業機構の課題

クラウド型開発管理環境の地域産業界への普及の促進を図り，地域産業全体の高度化に貢献し，地域産業ブランド化への貢献を実証してゆく．クラウド型開発管理環境を，都市型，係争リスク直面領域の開発に適用し，その有用性の実証と課題の抽出を図る．例えば，文部科学省 StagE プロジェクトの成果を反映した，ステークホルダーのニーズの整理，対応策の案出などが考えられる．

参考文献

[Antoniol] G. Antoniol, V. F. Rollo, G. Venturi, “Deteting groups of co-hangingles in CVS repositories,” *Proc. of* 8th *International Workshop on Principles of Software Evolution* (IWPSE'05), pp.23-32, 2005.

[Ardis] M. Ardis, P. B. Henderson, “Making a case for good engineering practices: The Toyota unintended acceleration investigation,” *ACM SIGSOFT Software Engineering Notes*, Vol.36, No.3, May 2011.

[Beyer] D.Beyer, “Co-change visualization applied to PostgreSQL and ArgoUML,” *Proc. of* 2006 *International Workshop on Mining Software Repositories*, pp.165-166. 2006.

[Bowring] J. Bowring, A. Orso, Mary Jean Harrold, “Monitoring deployed software using software tomography,” *Proc. of* 2002 *ACM Workshop on Program Analysis for Software Tools and Engineering*, pp.2-9, 2002.

[Cluzet] S. Cluzet, C. Torregrosa, C. Jacquet, C. Lafitte, J. Fournier, L. Mercier, S. Salamagne, X. Briand, M. T. Esquerre-Tugaye, and B. Dumas, “Gene expression profiling and protection of Medicago truncatula against a fungal infection in response to an elicitor from green algae Ulva spp,” *Plant, Cell and Environment*, Vol.27, pp.917-928, 2004.

[Farend] ファーエンドテクノロジー社,オープンソースによるプロジェクト管理入門, 秀和システム, 2010.

[Geipel] M. M. Geipel, F. Shweitzer, “The Link between dependeny and co-change: Empirial evidene,” *IEEE Transations on Software Engineering*, Vol.38, No.6, pp.1432-1444, Aug. 2011.

[GemX] GUI フロントエンド GemX のチュートリアル, http://www.ccfinder.net/doc/10.2/ja/tutorial-gemx.html

[Higo] 肥後芳樹, 吉田則裕,“コードクローンを対象としたリファクタリング”, コンピュータソフトウェア, Vol.28, No.4, pp.43-56 (平 23-11).

[Hooimeijer] P. Hooimeijer, W. Weimer, “Modeling bug report quality,” *Proc. of* 22nd *International Conference on Automated Software Engineering*, pp.34-43, 2007.

[IPA] 情報処理推進機構, “ソフトウェアの品質説明力強化のための制度フレームワークに関する提案 (中間報告)” (平 23-9).

[JUAS] 社団法人日本情報システム・ユーザー協会 (編) : 非機能要求仕様定義ガイドライン, 2008.

[Kamiya] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code,” *IEEE Transactions on Software Engineering*, Vol.28, No.7, pp.654-670, 2002.

[KHC] http://khc.sourceforge.net/

[Lopes] C. Lopes, S. Bajracharya, J. Ossher, and P. Baldi: UCI Source Code Data Sets, 2010. http://www.ics.uci.edu/~lopes/datasets

[Krinke] J, Krinke, "Statement-Level Cohesion Metrics and their Visualization", *Proc. of 7th IEEE International Working Conference on Source Code Analysis and Manipulation,* pp.37-46, 2007.

[Livshits] B. Livshits, T. Zimmermann, "DynaMine: Finding common error patterns by mining software revision histories," *Proc. of 10th European software engineering conference*, pp.296-305, 2005.

[Maeda] 前田剛, 入門 Redmine, 第3版, 秀和システム, 2012.

[METI] 経済産業省 商務情報政策局 情報処理振興課, 独立行政法人 情報処理推進機構: 情報システム調達のための技術参照モデル(TRM)平成22年度版, 2011.

[Meyers] T. M. Meyers and D. Binkley, "An Empirical Study of Slice-Based Cohesion and Coupling Metrics," *ACM Transactions on Software Engineering and Methodology*, Vol.17, No.2, 2007.

[MRI] 株式会社三菱総合研究所: 経済産業省平成22年度 プロダクト品質メトリクスWG実施内容 ―ソフトウェアメトリクス高度化プロジェクト, 2010.

[MatsuATGSE] K. Matsumoto, "STAGE Project (Software Traceability and Accountability for Global software Engineering) - Purchaser-Centered Approach in Empirical Software Engineering -," *Proc. of Workshop on Accountability and Traceability in Global Software Engineering* (ATGSE 2007), December 2007.

[MatsuICSEC] K. Matsumoto, "Empirical Software Engineering," 2011 International Computer Science and Engineering Conference, September 2011

[MatsuCSPIN] K. Matsumoto, "Empirical Approach to Software Engineering," *Proc. of 7th China System and Software Process Improvement Annual Meeting* (CSPIN-CSSPI 2008), September 2008.

[Matsuo] 松尾 豊, 篠田考祐, 中島秀之: 中心性に着目した合理エージェントのネットワーク形成, 人工知能学会論文誌, Vol.21, No.1, pp.122-132 (2006).

[MatsuSEC] 松本健一, "事故前提社会に向けたユーザ・ベンダ間での開発データ共有ーStagEプロジェクトとソフトウェアタグー", SEC journal, Vol.5, No.3, pp.198-203 (平21-6).

[MatsuSPES] 松本健一, 松村知子, "ユーザ・ベンダ間での情報共有技術「ソフトウェアタグ」の実用化に向けて ～利用シナリオと事例解説～", ソフトウェアプロセスエンジニアリングシンポジウム2010 (平22-7).

[Meneely] A. Meneely and L. Williams, "Socio-technical developer networks: should we trust our measurements?," *Proc. of 33rd International Conference on Software Engineering*, pp.281-290, 2011.

[Nikkei] 日経ソリューションビジネス編: システム構築のトラブルを回避するためのITシステム契約締結の手順とポイント, 日経BP社, 2008.

[Ossher] J. Ossher, H. Sajnani, and C. Lopes. "File cloning in open source Java projects: The good, the bad, and the ugly", *Proc. of International Conference on Software Mainteinance*, pp.283-292, 2011.

[Pan] K. Pan, S. Kim, E. J. Whitehead Jr., "Toward an understanding of bug fix patterns," *Empirical Software Engineering*, Vol. 14, Issue 3, pp.286-315, Jun, 2009.

[Sakai] 阪井誠（監修）, 小川明彦, 阪井誠（著）, チケット駆動開発, 翔泳社, 2012.

[Sarma] A. Sarma, L. Maccherone, P. Wagstrom, J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," *Proc. of* 31*th International Conference on Software Engineering*, pp.22-33, 2009.

[SCITOOLS1] http://www.scitools.com/

[SCITOOLS2] http://www.scitools.com/features/metrics.php

[SEC] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター, ソフトウェアエンジニアリングの実践, SEC BOOKS, 翔泳社, 2007.

[Seo2005] J. Seo, B. Shneiderman, "A rank-by-feature framework for interactive exploration of multidimensional data," *Information Visualization*, Vol.4, No.2, pp.96-113, 2005.

[Seo2007] J. Seo and H. Gordish-Dressman, "Exploratory Data Analysis with Categorical Variables: An Improved Rank-by-Feature Framework and a Case Study," *International Journal of Human-Computer Interaction*, Vol.23, No.3, pp.287-314, 2007.

[Śliwerski] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes?. *Proc. of* 2005 *International Workshop on Mining Software Repositories*, pp.1-5, 2005.

[Taskpit] http://taskpit.jpn.org/

[TECHMTX1] http://www.techmatrix.co.jp/

[TECHMTX2] http://www.techmatrix.co.jp/i/qu/understand/app/documents/metrics.html

[Tsai] J.-M. Tsai, H.-C. Wang, J.-H. Leu, H.-H. Hsiao, A. H. J. Wang, G.-H. Kou, C.-F. Lo, "Genomic and proteomic analysis of Thirty-nine structural proteins of shrimp white spot syndrome virus," *Journal of Virology*, Vol.78, pp.11360-11370, 2004.

[UCI] http://www.ics.uci.edu/~lopes/datasets/SDS_source-repo-18k.html

[Weiser] M. Weiser, "Program slicing", *Proc. of* 5*th Internationl Confefence on Software Engineering*, pp.439-449, 1981.

［Yasuda］　保田勝通，ソフトウェア品質保証の考え方と実際，実践ソフトウェア開発工学シリーズ１３，日科技連，1995.