修士学位論文

題目

トレーサビリティ実現のためのソフトウェアタグ生成基盤システムの開発

指導教員
楠本 真二 教授

報告者
鵜久森 将隆

平成 22 年 2 月 8 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻

平成 21 年度 修士学位論文

トレーサビリティ実現のためのソフトウェアタグ生成基盤システムの開発

鵜久森 将隆

内容梗概

ソフトウェアの開発における定量的データの収集・分析は，ソフトウェア受注者側のプロセス改善やプロジェクトの進捗・品質管理を目的としている．しかし近年，ソフトウェア発注者側の視点に立った開発プロジェクトデータのより幅広い活用が求められ，ソフトウェアへのトレーサビリティの適用が提案されている．ソフトウェアに対するトレーサビリティとは製品品質や開発状況の把握が発注者にも可能なことを意味する．このソフトウェアトレーサビリティを実現する技術としてソフトウェアタグに関する研究・開発が行われている．ソフトウェアタグとはソフトウェア開発中に収集された種々のメトリクスをパッケージ化したものである．ソフトウェアタグを実現するにはソフトウェアタグの規格化，データ収集・可視化・評価手法の開発，実証実験の実施，法的諸問題の検討といった様々な課題がある．

本研究ではソフトウェア開発プロジェクトの実証データからデータを収集し，ソフトウェアタグを生成するシステムを提案する．ソフトウェアタグの生成方法は，どのようなプロジェクトにも適応できること，低コストであること，生成されたソフトウェアタグが利便性の高いものであることが要求されている．

提案するシステムは，収集する対象データをユーザが定義できる機能をもち，この定義に従った GUI の入力フォーマットを提供する．収集対象データをユーザが定義できることにより，どのようなプロジェクトにも利用が可能なシステムを実現した．また一部のデータ項目では自動データ収集機能を提供する．自動データ収集機能により，低コストでのソフトウェアタグ生成を実現した．自動データ収集機能の実装においては，ソフトウェア開発時に利用される開発支援ツールの機能や特徴を調査し，開発支援ツールから生成される実証データからデータを収集する方法を検討した．さらに，収集，生成されたデータを XML 形式のソフトウェアタグとして出力することによりソフトウェアタグの利便性向上の要求に対応した．

実験ではあるソフトウェア開発プロジェクトのデータを用いて，開発したシステムでのデータ入力コストを評価した．その結果，1 回の入力において利用者の作業時間は平均 2 分程度であり，入力における負荷はそれほど高くないことを確認した．

主な用語

ソフトウェアメトリクス
データ収集
ソフトウェアトレーサビリティ
ソフトウェアタグ

目 次

1　まえがき

近年，ソフトウェアの大規模化と社会的影響の拡大によりソフトウェアの重要性が高まると共に，さまざまな問題が発生している [1]．例えば，ソフトウェアに障害が発生することで，金融システムや交通システムなどの重大な社会インフラが停止したり，航空管制システムや自動車安全システムなどが人命に関わる危険を引き起こす．また，それに伴いユーザ・ベンダに莫大な経済的損失を与える．さらに開発期間の短縮のためにコストの低減や生産性の向上が要求されたり，情報システム開発に関わる訴訟が頻繁に発生している [2]．

これらの問題を解決するアプローチの一つとして，ソフトウェアトレーサビリティの概念が提案されている．トレーサビリティとは本来，野菜や食肉等の物品の生産・流通履歴を確認可能であることを意味するが，これをソフトウェアに対し適用することでソフトウェアユーザによるソフトウェアの製品品質・開発状況の把握や，発注者・受注者間の法的係争発生時の処理の短期化を可能とすることが期待されている．

このソフトウェアトレーサビリティを実現するためにソフトウェアタグの研究開発が行われている [3]．ソフトウェアタグは，ソフトウェアの開発中に収集された管理データや品質情報など複数のデータから構成され，各データ種類をタグ項目と呼ぶ．タグ項目に従って，開発成果物や副次的に発生するデータ（実証データ/エンピリカルデータ）からタグデータが開発中，もしくは開発完了時に収集され，ソフトウェアタグが作成される．作成されたソフトウェアタグはソフトウェアに付随し，ユーザはタグの内容を見ることによって，対象ソフトウェアの開発過程や品質保証のプロセス，成果物の品質情報を入手できる．

ソフトウェアタグによるソフトウェアトレーサビリティの実現にはいくつかの課題があり，その一つにソフトウェアタグを生成する技術の開発がある．そこで本研究ではソフトウェアトレーサビリティ実現の一環として，ソフトウェアタグ生成システムの提案を行う．

ソフトウェアタグの生成については，収集データが異なるプロジェクトに対して適用できること，低コストで生成できること，生成されたソフトウェアタグが利便性の高いものであることが求められる．本研究ではこれらの要求を解決するため，収集するタグデータ項目をシステムユーザが定義可能なソフトウェアタグ生成システムを提案する．システムユーザは標準で用意されているデータ項目以外に，必要なデータ項目を追加可能である．システムは追加されたデータ項目定義を読み込み，それに従った入力フォーマットを提供する．

また，一部の項目については自動データ収集機能を提供し，低コストでソフトウェアタグを生成する．収集されたタグデータは，統一的な記法をもつ XML[4] で出力することにより，ソフトウェアタグ分析ツール [5][6] などで利用することが可能である．

以降，2 章ではソフトウェアトレーサビリティ実現にむけた概況を述べ，3 章ではソフトウェアタグの定義について説明する．4 章ではソフトウェアタグ生成システムに対する要求

や利用モデル，システム構成について述べる．5 章では実プロジェクトからシステムを利用してソフトウェアタグを生成し，ソフトウェアタグ生成のコストを計測する．6 章，7 章では，システムを利用したソフトウェアタグ生成の利点や問題点を考察する．8 章では本研究で参考にした文献や関連研究を挙げる．最後に 9 章でまとめと今後の課題を述べる．

2　ソフトウェアトレーサビリティの実現

2.1　ソフトウェア受発注における問題

ソフトウェア開発においてユーザは要件定義などの上流工程からベンダ任せにするケースが従来多かった [7]．しかしそういった開発では，要件のあいまいさから設計ミスや頻繁な仕様変更が発生し，品質低下，重大な不具合によるシステムダウン，コスト増大，納期遅れなどをもたらす．また，障害発生時には，ユーザによる原因や責任所在の追究が困難で，対応が遅れ，甚大な損害が発生するケースも多い．さらに，法的な係争に発展すると，裁判が長期化し，解決まで発注者・開発者双方の被害が増大する．

以上から近年，ソフトウェア開発プロジェクトにおいてユーザの参画が重要視されてきている．しかしこれまでソフトウェア開発におけるデータ収集・分析はベンダ側でのプロセス改善やプロジェクト内部の進捗・品質管理を目的とし，ユーザの視点に立って行われてこなかった．そこで開発プロジェクトデータのより幅広い活用が求められている．

2.2　ソフトウェアトレーサビリティ

2.1 節の問題の解決策としてソフトウェアトレーサビリティの概念が提案されている．トレーサビリティとは，物品の生産・流通履歴を確認可能であることを意味し,「追跡可能性」とも言われている．近年，食の安全上の問題から食品に対するトレーサビリティへの取り組みが活発に行われており，生産・処理・加工・流通・販売等の各段階で，情報の追跡・遡及が可能になるよう推し進められている (図 1)．

トレーサビリティの概念をソフトウェア開発に適用したソフトウェアトレーサビリティでは，ソフトウェア開発過程で作成される実証データからソフトウェアタグを作成し，ソフトウェア製品に添付して流通させる (図 2)．製品に添付されたソフトウェアタグからユーザはソフトウェアの製品品質や開発状況を把握する．

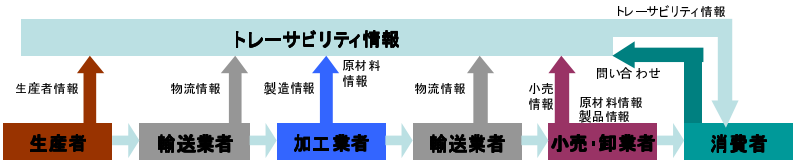

図 1: 食品のトレーサビリティ [3]

なお，過去にソフトウェア開発分野においてトレーサビリティという言葉は，能力成熟度モデル [8] で使用されてきた．能力成熟度モデル（CMMI）とは，ソフトウェア開発組織のソフトウェア開発能力向上のために利用されるプロセス改善モデルである．CMMI ではプ

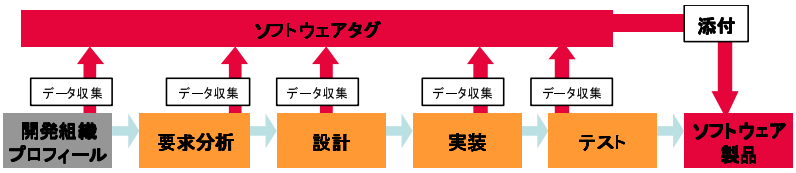

図 2: ソフトウェアトレーサビリティ [3]

ロセス改善方法の一つとして要求仕様と計画・開発・成果物間のトレーサビリティの必要性を定義している．しかし，ここで述べられているトレーサビリティは開発組織のプロセス改善のためであり，ユーザ側にトレーサビリティを与えるものではない．

2.3 StagE プロジェクト

StagE(Software Traceability and Accountability of Global Software Engineering) プロジェクト [3] とは，文部科学省のプロジェクト「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と普及」の略称として，平成 19 年度から奈良先端科学技術大学院大学と大阪大学が産学官連携で進めているプロジェクトである．ソフトウェアトレーサビリティの実現を目的とし，ソフトウェアタグの研究開発を行っている．

2.4 ソフトウェアトレーサビリティ実現に向けての研究開発活動

ソフトウェアタグによるソフトウェアトレーサビリティの実現に向けて，５つの研究開発活動を行っている（図 3）．

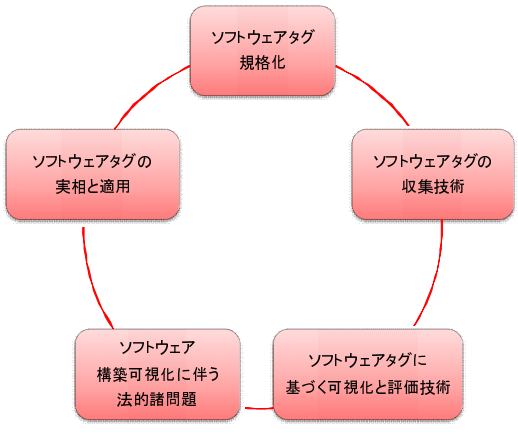

図 3: ソフトウェアトレーサビリティに向けての研究開発活動

本研究はソフトウェアタグ生成技術の開発を目標としている．

3　ソフトウェアタグ

3.1　概要

ソフトウェアタグとは開発に関する実証的データ（エンピリカルデータ）[1] を収集しそのデータを一定の形式でパッケージ化したものである．StagE プロジェクトはソフトウェアタグ規格 第 1.0 版を策定している [9]．この規格においてソフトウェアタグは管理データや品質情報など複数のデータから構成され，各データ種類をタグ項目と呼ぶ．タグ項目は発注者・受注者間で共有する候補として定義してあり，発注者・受注者は用途に応じてタグ項目を選択しソフトウェアタグを構成する．

タグ項目は全 41 項目あり，プロジェクト情報と進捗情報に大別される．

3.2　プロジェクト情報

ソフトウェアタグの内，プロジェクト情報とは開発プロジェクトおよびシステムの基本的な情報を指す．プロジェクト情報はプロジェクトの途中で変更がなくプロジェクト全体を通して不変である．

プロジェクト情報内でタグ項目は 12 個あり，情報の種類ごとに以下の 5 つに分類される．

- 基本情報
- システム情報
- 開発情報
- プロジェクトの階層構造情報
- その他

表 1 にプロジェクト情報の一覧を示す．表 1 の予定・実績の要否は予定と実績による管理が必要かを示している．成果物やプロセス，目標についてベースラインを作成することはソフトウェア開発の改善において重要であるため [10]，いくつかのタグ項目に対し予定と実績による管理を要求している．

3.3　進捗情報

ソフトウェアタグの内，進捗情報とはソフトウェア開発によって得られた作業の進捗状況や，成果物やプロセスの品質を表す情報を指す．進捗情報はプロジェクト情報とは異なりプロジェクト途中で値が変化する．

表 1: ソフトウェアタグ:プロジェクト情報

分類	項番	タグ項目	説明	具体化例	実証データ例	予定・実績の要否	工程別の要否
基本情報	1	プロジェクト名	プロジェクトを一意に決定するための識別名		プロジェクト計画書 など		
	2	開発組織の情報	当該プロジェクトの開発を担当する組織の情報．一般には，受注者となる組織情報となる．	ISO/IEC 15504 Process Assessment，米カーネギメロン大CMMI®などを用いたアセスメント結果のプロセス達成度一覧または組織成熟度レベル	発注仕様書 アセスメント結果に関する情報 体制表 キャリアシート など		
				開発組織名，対象業種の経験 など			
	3	開発プロジェクト情報	開発プロジェクトの特徴や当該タグデータの対象とするプロジェクトの種類を示す情報．タグデータの解釈や分析時に必要なデータ．	開発プロジェクト種別：新規・改造・保守・運用・流用など	要件定義書 プロジェクト計画書 品質計画書 設計ドキュメント ソースコード テスト計画書 など		
				設計書再利用率・ソースコード再利用率・コンポーネント再利用率など		○	
				開発プロジェクト形態（商用パッケージ，受託開発），利用パッケージ など			
	4	顧客情報	当該システムのユーザ，もしくは第1発注者となる組織の情報．	顧客名，新規顧客か否か など	顧客との議事録 プロジェクト計画書 など		
システム情報	5	システム構成	開発システム構成の特徴や当該タグデータの対象とするシステムの種類を示す情報．タグデータの解釈や分析時に必要なデータ．	利用したサブシステムの安定度：利用ハードウェア、ライブラリ、OS等、調達したサブシステムの安定度合い（初回、不安定等）	基本設計書 プロジェクト計画書 工程管理表 勤務実績データ など		
				サブシステムの検証期間/工数：利用を検討したサブシステムの検証期間と工数			
				利用したサブシステムに関する法的要件 など			
	6	システム規模	開発システムの規模．計画値と最終実績値とする．進捗情報に同じ情報が含まれる場合は，省略可．	受注者側の想定する顧客要件の数，実装した要件数など	基本設計書 ソースコード 要件定義書 など	○	
				工程終了時のソースコード行数,機能,FP,処理データ量,処理データ数など		○	
開発情報	7	開発手法	開発システム開発に用いたプロセスや手法についての情報．タグデータの解釈や分析時に必要なデータ．	適用プロセスタイプ（ウォーターフォール・アジャイル・プロトタイプ），適用手法（OOAD，SASD，DDE）など	プロジェクト計画書 プロジェクト生産管理計画書 など		
	8	開発体制	開発側の要員に関する情報．タグデータの解釈や分析時に必要なデータ． ※開示対象範囲は，発注者・受注者側での協議により決定する	組織図・人員配置図 作業体制の複雑さ・親密さ(結合度・分散度)：組織図・人員配置図から計測 プロジェクト体制の階層数　など	体制表 発注書 キャリアシート 勤務実績データ 工程管理表 など		○
				外注率，ピーク時の開発人員，延べ開発人員，チーム別の開発要員　など		○	○
				経験年数，レベル別割合(上級・中級・初級)，新人比率， 類似プロジェクトの経験(開発、ユーザともに類似プロジェクト参加の経験があるか：経験なし、3回未満、5回以上)， 要員スキル（PM,業務，分析・設計，言語・ツール，開発プラットフォーム：ITSSなどに準拠）　など			○
	9	プロジェクト期間	当該プロジェクトの開発期間に関する情報	稼働時期、開始、終了を含む（時間、日、月）など	勤務実績データ 工程管理表 など	○	○
				工期 など		○	○
プロジェクトの階層構造情報	10	親プロジェクト情報	本プロジェクトが別のプロジェクトのサブ(子)プロジェクトである場合，付加	親プロジェクトのプロジェクト名	プロジェクト計画書 機能設計書 構造設計書 など		
	11	サブ(子)プロジェクト情報	本プロジェクトがサブ(子)プロジェクトを持つ場合，その数やサブ(子)プロジェクトに関する情報	サブ（子）プロジェクトの数，プロジェクト名など	プロジェクト計画書 機能設計書 構造設計書 など		
その他	12	特記事項	その他，タグデータの解釈や分析時に必要，もしくは有用なデータ．				

進捗情報内ではタグ項目は 29 項目存在し，そのタグ項目がどの開発プロセスに関するかでさらに以下の 8 つに分類される．

- 要件定義
- 設計
- プログラミング
- テスト
- 品質
- 工数
- 計画・管理
- その他成果物

表 2 に進捗情報の一覧を示す．なお，タグ項目名の後ろに [推移] という語句が付けられているものは，時系列で連続するデータが想定されるタグ項目である．

3.4 タグ利用のシナリオ

ソフトウェアタグ定義を発注者・受注者がどのように利用するのか一例を述べる．ソフトウェアタグの内，19 番プログラミングの規模としてソースコードからコード行数を，20 番プログラミングの変更としてソースコード行数の差分量を選択したとする．これにより，発注者は実装時にコード行数がどのように推移して作成されたか，また，日々のコード行数の推移がどれくらいの変更・削除・追加によって行われたかを把握可能になる．また，30 番の欠陥対応件数として不具合消化数を選択すれば，不具合を修正するためにどの程度ソースコードの修正を行ったか分かる．タグの利用方法としては，製品品質の把握，開発状況の把握，紛争処理がある．実際に開発者・発注者間でソフトウェアタグを作成する際に，これらの目的に応じてどのソフトウェアタグ項目を使用するかを選択する必要がある．各利用方法に対し，各タグ項目がどのように用いられるかを述べる．

製品品質を把握するには，進捗情報の分類”品質”に含まれるタグ項目が用いられる．例えばバグ数やレビュー状況からソフトウェアが程度作りこまれたかを調べる．また，製品品質を把握することで，ソフトウェアを部品として再利用することも可能である．その際にはシステム構成や，テスト項目数，リスト項目数などが使用される．

表 2: ソフトウェアタグ:進捗情報

分類	項番	タグ項目	説明	具体化例	実証データ例	予定・実績の要否	備考
要件定義	13	ユーザヒアリング情報	要件に関してユーザに行ったヒアリングに関する情報	ユーザヒアリング実施件数(回)	ユーザヒアリング議事録 ユーザヒアリング質問票 など	○	
				ユーザヒアリング項目数(件), ユーザヒアリング回答率(ユーザヒアリング回答数÷ユーザヒアリング項目数) など			
	14	規模[推移]	開発側で作成した要件数	画面、機能項目、ユースケース、アクター、顧客要件、機能、FPなど	要件定義書 など	○	何を要件の基本単位とするかは、要合意事項
	15	変更[推移]	変更された要件数	規模の計測単位に依存	要件定義書 要件定義書の変更履歴 など		
設計	16	規模[推移]	設計成果物の規模 ※新規・改造・再利用(流用)毎に計測する	機能設計(ページ数・帳票数・画面数・ファイル数・項目数・UML図の数、クラス数、バッチプログラム数, 重要な機能数など) 構造設計(データ項目数, DFDデータ数, DFDプロセス数, DBテーブル数 など) など	基本設計書 機能設計書 構造設計書 詳細設計書 など	○	
	17	変更[推移]	変更された設計成果物の数, もしくは変更量	規模の計測単位に依存	基本設計書 機能設計書 構造設計書 詳細設計書 各設計書の変更履歴 など		
	18	要件の網羅率	要件定義で作成された要件の実装率	設計に取り入れられた要件数÷要件数	要件定義書 基本設計書 機能設計書 構造設計書 詳細設計書 など		
プログラミング	19	規模[推移]	プログラミング成果物の規模 ※新規・改造・再利用(流用)毎に計測する	LOCなど	ソースコード など	○	ファイル別・モジュール別
	20	変更[推移]	変更されたプログラムの数, もしくは変更量	変更回数, 削除行数, 追加行数など	ソースコード リポジトリ更新履歴 など		ファイル別・モジュール別
	21	複雑度	プログラムの品質(保守性)	Halstead, McCabe, CK, クローン含有率, コメント平均利用率など	ソースコード など		ファイル別・モジュール別
テスト	22	規模[推移]	テストの規模 ※新規・改造・再利用(流用)毎に計測する	テスト項目数など	テストケース テスト計画書 など	○	
	23	変更[推移]	変更されたテスト項目数や変更量	追加テスト項目数, 変更テスト項目数など	テストケース 変更履歴 など		
	24	密度	テストの品質	テスト項目数÷システム規模(ソースコード行数など)	テストケース テスト結果 ソースコード	○	「No.6 システム規模」と「No.22 規模」から導出可
				C0(命令網羅), C1(分岐網羅), 要件に対するカバレッジ率(結合/受け入れテスト) など	要件定義書 など	○	
	25	消化	テストの進捗, プログラムの品質	テストケース進捗, 消化テスト項目推移, テストプログラム消化率, クリティカルパス進捗など	テストケース テスト計画書 テスト結果 工程管理表 など	○	
品質	26	レビュー状況	成果物(仕様書, 設計書, プログラムコード, テスト仕様書など)のレビューに関する情報	レビュー回数, 対象規模, 実績回数, ユーザ参画の有無, レビュー時間など	レビュー議事録 など	○	
	27	レビュー作業密度	レビュープロセスの品質, もしくはレビュー対象の品質	レビュー工数率, レビュー密度など	レビュー議事録 レビュー対象 (ソースコード, 各設計書) など		
	28	レビュー指摘率[推移]	レビュープロセスの品質, もしくはレビュー対象の品質	レビュー指摘数, 指摘密度, 不具合指摘数推移など	レビュー議事録 レビュー対象 (ソースコード, 各設計書) など		
	29	欠陥件数[推移]	テスト設計の品質とコード品質		テスト結果 障害管理票 品質管理表 など	○	欠陥が発生した工程別, 原因ごとにも集計
	30	欠陥対応件数	欠陥の対応進捗, 対応内容	不具合消化数, 障害滞留時間, 検出欠陥の分析実施回数, 類似欠陥調査実施回数など	テスト結果 障害管理票 品質管理表 など		
	31	欠陥密度	テスト設計の品質とコード品質	検出欠陥数をシステム規模で割る:不良摘出件数÷ステップ数など	テスト結果 障害管理票 品質管理表 ソースコード など	○	「No.6 システム規模」と「No.29 欠陥件数」から導出可

	32	欠陥指摘率	テスト設計の品質	不良摘出件数÷テストケース消化数	テスト計画書 テストケース テスト結果 障害管理表 品質管理表 など	○	「No.25 消化」と「No.29 欠陥件数」から導出可
	33	静的チェックの結果	プログラムの品質（保守性）	チェッカによる総エラー数，チェッカによる総警告数など	ソースコード など		
工数	34	作業工数	作業に要する工数，仕様変更作業工数		工程管理表 勤務実績データ など	○	
	35	生産性	工数に対する成果物の比率	規模（画面数・帳票数）÷工数 頁÷工数(設計工程) ステップ数÷工数（製造工程） 不良摘出件数÷工数(テスト工程) など	構造設計書 勤務実績データ ソースコード テスト結果 など	○	「No.34 作業工数」と各成果物の規模などから導出可
計画・管理	36	プロセス管理情報	開発プロセスの管理に関する情報	プロセス規定度（提出されたプロセス記述の詳細さに基づく：WBSのタスク数など）， 標準プロセスに対する網羅度，プロセス遵守度（プロセス記述と実際の作業間の適合の度合いに基づく）， プロセス管理度（各ステップごとに設定された会議体の数や提供された管理指標の数に基づく） など	プロジェクト計画書 プロジェクト生産管理計画書 勤務実績データ 工程管理表 議事録 など		
	37	会議実施状況	ユーザ・ベンダ間，ベンダ間での情報共有状況を把握	各会議の時間、参加人数、資料量、議事数、議事録量、情報共有者数など	議事録 など		会議：要件検討会，進捗会議，納入／検収など
	38	累積リスク項目数	リスク認識が十分であったかを把握	進捗会議で挙げられたリスク項目数の累積、軽微、重大、その他の項目でわける	議事録 リスク管理表 など	○	
	39	リスク項目の滞留時間	リスク対策が適切になされていたかを把握	リスク管理項目の最長、平均滞留時間	議事録 リスク管理表 など		
その他成果物	40	規模[推移]	対象成果物の規模 ※新規・改造・再利用(流用)毎に計測する		ドキュメント 計画書 など	○	対象成果物：マニュアル，計画書，スケジュール表など
	41	変更[推移]	変更された対象成果物の数，もしくは変更量．	変更回数，変更ページ数など	ドキュメント変更履歴 など		

開発状況を把握するには，開発がどのような環境で行われたかを表すために開発体制や開発手法が使用される．また，どういったプロセスを追って開発されたかを調べるためにプロセス管理情報などが使用される．

紛争処理の際には，ソフトウェアが十分テストされたものかを見るためにバグ数，レビュー状況，テスト項目数などが使用される．また，発注者・開発者間でソフトウェアに対しどのよな合意があったかを確かめるために発注者ヒアリング情報，要件変更，会議実施状況が使用される．

4　ソフトウェアタグ生成システム

4.1　ソフトウェアタグ生成に対する要求

ソフトウェアタグの生成に対する要求について述べる．

プロジェクトに依存しないソフトウェアタグの生成方法

3.1 節で述べたとおり，ソフトウェアタグとなるデータ項目は各発注者，受注者間で協議した上で選択される．協議の上でのデータ項目は様々な形が考えられる．例として，プロジェクト A の協議ではタグ項目 26 番 [レビュー状況] のデータとしてレビュー回数を選択したとする．しかし，プロジェクト B の協議ではレビュー合計時間を選択する可能性があり，またプロジェクト C ではその両方，もしくはまったく別のデータを収集対象とするかもしれない．このようにプロジェクトにより収集データの項目は異なる．システムはこれに対応できるように，プロジェクトに依存しないソフトウェアタグの生成方法を提供する必要がある．

低コストでのソフトウェアタグの生成

ソフトウェア開発プロジェクトにおいてコストを抑えることは重要な課題の１つである．コストオーバーとなるプロジェクトは後を絶たず，特にソフトウェア開発においては，２次受け，３次受けのプロジェクトはコストに余裕のないことも多い．このようなプロジェクトは，ソフトウェアタグを生成するためのリソースを十分に割り当てることができないと考えられる．このため，低コストでソフトウェアタグを生成できることが要求される．

利便性の高いソフトウェアタグの出力

これまでも何度か述べたが，ソフトウェアタグの目的は「ソフトウェア開発データの発注者との共有を行い，ソフトウェアの品質や由来を発注者に示す」ことである．ソフトウェアタグがこの目的を達成するためには，生成されたソフトウェアタグのデータの確認が容易であること，分析ツールなどで利用できる形式であることが要求される．

4.2　設計方針

4.1 節で述べた要求に対応する本システムの設計方針について述べる．

拡張性の高い収集対象タグデータの定義

前節で述べたとおり，開発ベンダやプロジェクトにより収集されるデータは異なるため，それらに依存しないソフトウェアタグの生成方法を考える必要がある．そこで収集対象とな

るデータをシステムユーザが定義可能なシステムを実現する．

収集するデータをタグデータ，タグデータに関する定義をタグデータ定義とよぶ．タグデータ定義を用いたタグデータ収集方法のイメージを図 4 に示す．

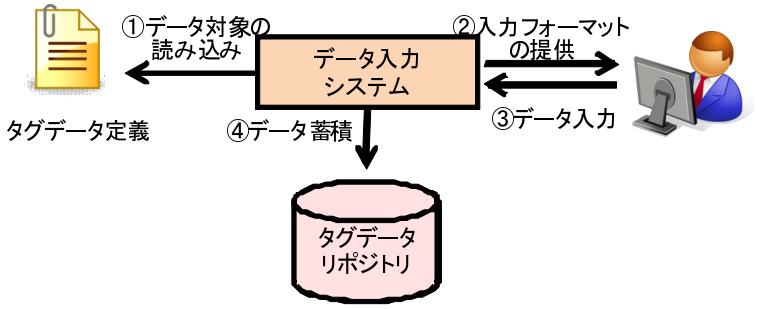

図 4: タグデータ定義と入力システムのイメージ

データ入力システムは，タグデータ定義を読み込み，それに従った入力フォーマットをユーザに提供する．ユーザは提供された入力フォーマットに従い，プロジェクトのデータを入力していく．入力が完了すると，そのデータはタグデータリポジトリに格納される．

この手法は柔軟で拡張性が高いタグデータの収集を可能にする．標準のタグデータ定義をシステム側で提供しておくと，データ入力システムは定義を読みこみ，システムユーザに標準データ項目の入力フォーマットを提供する．

タグデータ定義はシステムユーザにより追加が可能であるので，標準タグデータ定義で足りなければ，独自のタグ定義を追加する．これによりシステムはプロジェクトに依存しないソフトウェアタグ生成方法をシステムユーザに提供可能である．

このようにユーザにより追加が可能なタグデータ定義を利用することにより，プロジェクトに依存しないシステムを構成する．

プロジェクトデータ自動収集機能の提供

低コストでのソフトウェアタグ生成を目指すため，データ自動収集機能の実装を検討する．

一般的に，ソフトウェア開発プロジェクトは Subversion[11] や CVS[12] などのバージョン管理システムを利用している．バージョン管理システムはソースコードの管理に使われることが多く，そのリポジトリにはソースコードの更新履歴が蓄積されている．本システムはリポジトリから得られるデータの自動収集を考える．さらに開発プロジェクトでは Gnats[13] や影舞 [14] などのバグトラッキングシステムが利用されている．バグトラッキングシステムにはソフトウェアのバグに関する情報が蓄積されており，このログから品質に関するデー

タの自動収集を検討する．またソフトウェア開発では JUDE[1][15]，IBM Rational Rose[16]，Microsoft officeVisio[17] などの UML モデリングツールが利用されている．本システムではフリーで利用できる JUDE のファイルから要件定義，設計のデータを自動収集する方法について検討する．

データ自動収集機能の詳細については，4.5 節で述べる．

XML 形式でのソフトウェアタグ出力

XML とは文書やデータの意味や構造を記述するためのマークアップ言語の１つである．XML により統一的な記法を用いながら独自の意味や構造をもった言語を作成することができるため，ソフトウェア間の通信・情報交換に用いるデータ形式や，様々な種類のデータを保存するためのファイルフォーマットの定義によく用いられている．本システムは入力，収集されたデータを XML 形式のソフトウェアタグとして出力することを検討する．これによりソフトウェアタグの利便性が高くなり，グラフ表示ツールによるプロジェクトデータの確認や，ソフトウェアタグを入力とした分析ツールによるプロジェクトの分析が容易となる．

4.3　システム利用モデル

この節では本システムを利用したソフトウェアタグ生成のモデルについて述べる．

システムを利用したプロジェクトの開始から終了までのソフトウェアタグ生成の手順は次のようになる．

手順１ ソフトウェアタグプロジェクトの新規作成

手順 2 収集対象データ定義（タグデータ定義）の作成

手順 3 プロジェクトデータを収集

手順 4 手順 3 をプロジェクト終了まで一定の期間で繰り返す

手順 5 ソフトウェアタグの出力

手順 3，手順 4 の概要図を図 5 に示す．図 5 はプロジェクトの流れが時系列で表現されている．プロジェクトの開始後，システムを利用してデータを収集する．その後，一定期間ごとに同様の操作を繰り返して，タグデータを蓄積していく．図 5 では 3/27 に１度目のデータ収集を行い，3 日後の 3/30，その後３日置きにデータの収集を行っている様子が示されている．期間はさまざま考えられるが，データ収集のコスト，受注者へのタグデータ提示の手

[1]現在は astah*という名称に変更

間を考えると，発注者，受注者間で協議の上，決定すると良いと思われる．データ収集のコストに関しては第 5 章で実験を行う．

収集されたタグデータはタグデータリポジトリに蓄積される．その後，プロジェクトが終了，あるいは必要に応じて途中で，XML フォーマットのソフトウェアタグを出力する．

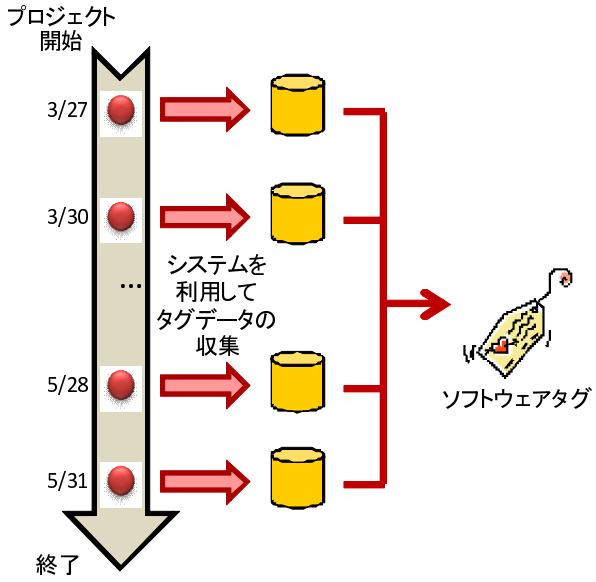

図 5: システムを利用したプロジェクトデータ収集モデル

4.4 システム構成

システムの全体構成を図 6 に示す．システムは主にタグデータ定義編集部，タグデータ入力部，タグデータ出力部から構成される．

4.4.1 タグデータの構成要素

タグデータの構成要素を図 7 に示す．

タグデータは具体値を示す value の他，その値についてのメタデータを持つ．tagIdentifier はデータ項目をユニークにする識別子，id はハッシュ値，name はデータ項目名，user はメトリクスの計測者，unit は値の単位，object は値の根拠となった議事録やソースコード，date は値を入力，もしくは計測した日時，remarks はその他の備考記録をもつ．

具体例を図 8 に示す．図 8 のタグデータから，実測値が 10，この値はレビュー回数の値であること，2007 年 4 月 18 日 15 時の時点でレビューを 10 回行ったこと，計測の根拠となっ

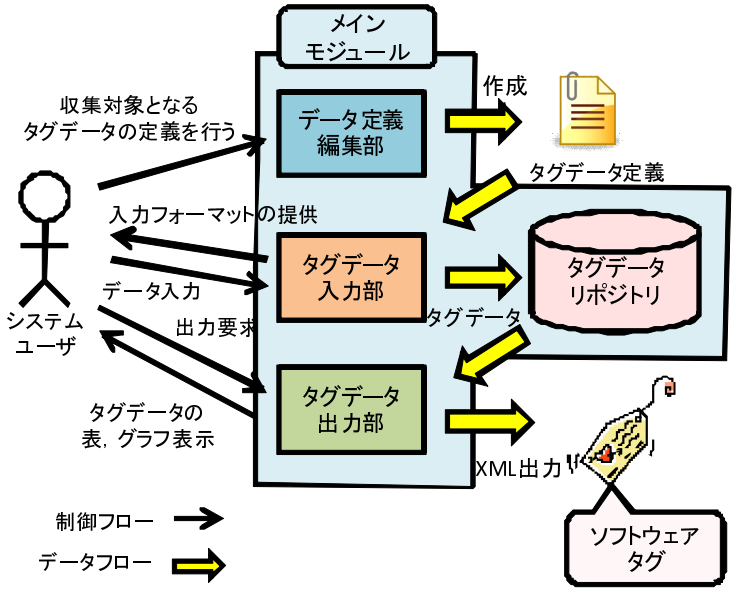

図 6: システムの全体構成

たものはレビュー議事録であることなどがわかる．このようにメタデータを記録しておくことで，さらに詳細なデータが必要な場合は議事録を参照するなどの行動が可能である．

4.4.2 タグデータ定義の構成要素

前項に続き，タグデータ定義の構成要素について説明する．タグデータ定義は，タグデータが表す内容についての定義を保持するものである．

構成要素を図 9 に示す．タグデータ定義は，タグデータをユニークにする識別子 tagIdentifier，タグデータの項目名 name，詳細を記す description，単位を示す unit，そして収集頻度を示す frequency からなる．

具体例として図 8 に示されたタグデータのタグデータ定義を図 10 に示す．このタグデータ定義から，レビュー回数のタグデータを計測する頻度や，レビュー回数についての詳細を得ることが可能である．

4.4.3 タグデータ定義編集部

タグデータ定義編集部は，4.4.2 項で説明したタグデータ定義を追加，除去する機能をもつ．図 11 にタグデータ定義編集部の構成を示す．システムのユーザはタグデータ定義編集機能を利用して収集対象となるタグデータ定義を作成する．ユーザにより作成されたタグ

```
TagData {
  value : 具体値
  tagIdentifier : 識別子
  id : ハッシュ値
  name : データ項目名
  user : 計測者
  unit : 単位
  object : 計測対象
  date : 日時
  remarks : その他備考
}
```

図 7: タグデータの構成要素

```
TagData {
  value : 10
  tagIdentifier :
        progressInformation.quality.
              reviewStatus.reviewNum
  id : 1365874845...
  name :レビュー回数
  user : 鵜久森
  unit : 件
  object : レビュー議事録
  date : 2007/04/18 15:00:00
  remarks : レビュー実施日 3/27, 3/30, ...
}
```

図 8: タグデータ具体例

```
TagDataDefinition {
  tagIdentifier : 識別子
  name : データ項目名
  description : タグデータ項目の詳細
  unit : 単位
  frequency : 収集頻度
}
```

図 9: タグデータ定義の構成要素

```
TagDataDefinition {
  tagIdentifier :
        progressInformation.quality.
              reviewStatus.reviewNum

  name : レビュー回数
  description : レビューを行った回数.
              レビューは毎週水曜と金曜に実施.
  unit : 件
  frequency : タグデータ計測日に準拠
}
```

図 10: タグデータ定義の具体例

データ定義は XML ファイルとして，ソフトウェアタグプロジェクトのタグデータ定義ディレクトリに保存される．タグデータ定義の XML ファイルを図 12 に示す．

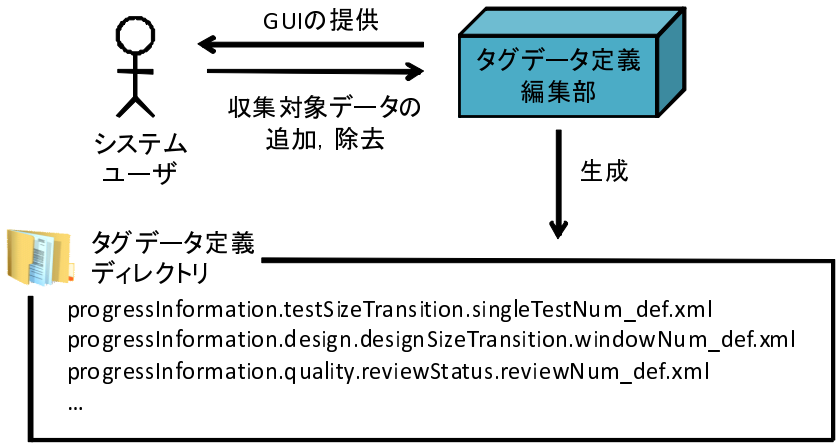

図 11: タグデータ定義編集部の構成

progressInformation.quality.reviewStatus.reviewNum_def.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<tagDataDefinition >
  <tagIdentifier>progressInformation.quality.reviewStatus.reviewNum</tagIdentifier>
  <name>レビュー回数</name>
  <description>レビューを行った回数．レビューは毎週水曜と金曜に実施．</description>
  <unit>件</unit>
  <frequency>タグデータ計測日に準拠</frequency>
</tagDataDefinition>
```

図 12: タグデータ定義の XML ファイル

また，プロジェクト A で作成されたタグデータ定義ファイルは，プロジェクト B のタグデータ定義ディレクトリにコピーすることで，プロジェクト B でも利用可能である（図 13）．ベンダやプロジェクトにより，よく使うタグデータ定義をまとめておくとタグデータ定義を作成する時間を削減できる．

4.4.4 タグデータ入力部

タグデータ入力部の構成を図 14 に示す．

タグデータ入力部は入力ウィザード，入力フォーマット生成部，自動データ収集部から構成される．入力フォーマット生成部はユーザによって定義されたタグデータ定義を読み込み，

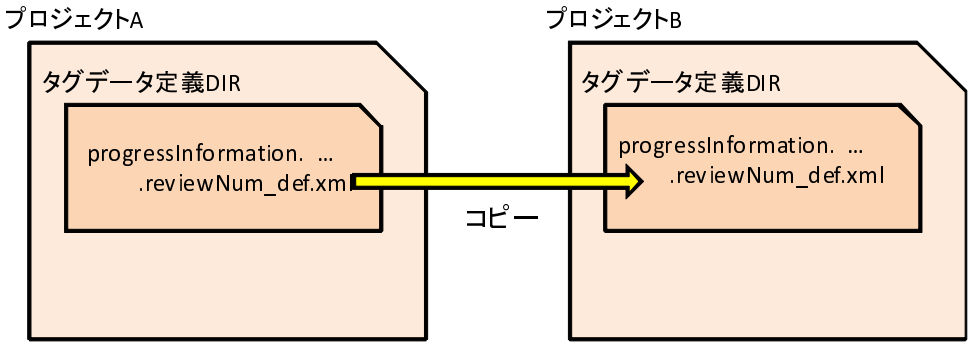

図 13: タグデータ定義の流用

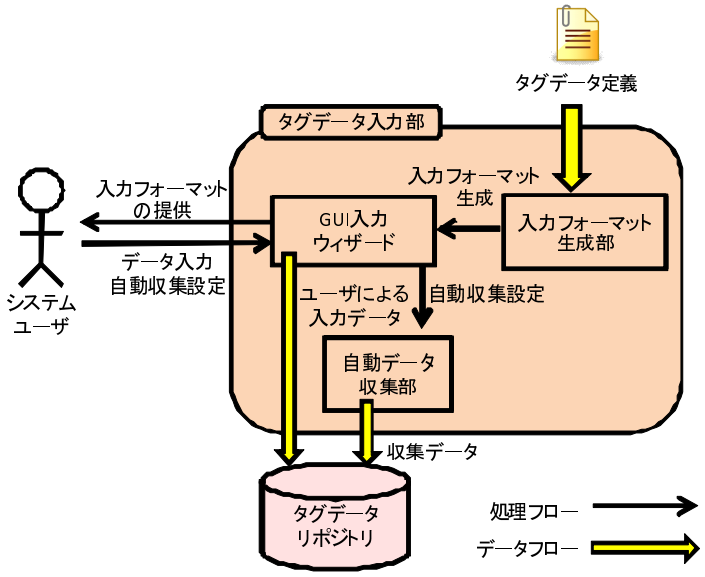

図 14: タグデータ入力部の構成

タグデータの入力フォーマットを生成する．入力ウィザードは入力フォーマットをシステムユーザに提示し，ユーザはフォーマットに従ってデータを入力する．また，同時に自動収集する項目についても設定を行う．入力が完了すると，ユーザによる入力データはタグデータリポジトリに格納され，自動収集項目についてはその設定が自動データ収集部に渡される．自動データ収集部は受け取った設定に従ってデータの計測，収集を行い，同じくタグデータリポジトリに格納する．

また，データ入力のサポートとして，前回の入力データを保持しておく機能をもつ．入力フォーマットには前回入力データがすでに入力されているため，変更がない項目については入力する必要がない．これは計測者情報である user や計測単位である unit，計測根拠の object などあまり変更がない項目に対して，入力コストの面で効果的である．

4.4.5 タグデータ出力部

タグデータ出力部の構成を図 15 に示す．

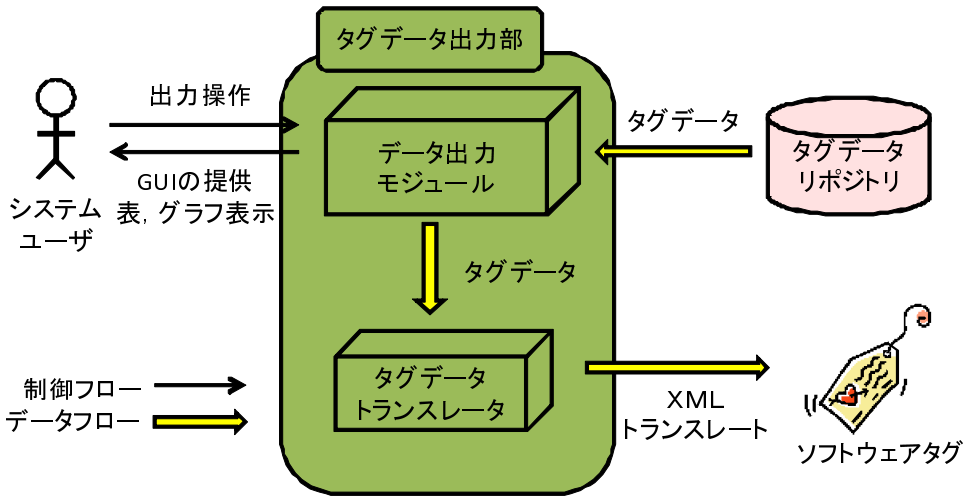

図 15: タグデータ出力部の構成

タグデータ出力部はデータ出力モジュールとタグデータトランスレータから構成される．システムユーザは GUI を通して出力操作を行い，データ出力モジュールは操作に応じてタグデータの表，グラフ表示やトランスレータ命令を行う．タグデータトランスレータはタグデータを XML にトランスレートし，ソフトウェアタグを出力する．ソフトウェアタグの出力のタイミングは任意であり，必要に応じてプロジェクトの途中で出力も可能である．出力のタイミングとして，プロジェクト完了時の他，発注者への開発経過報告時などが考えれる．

表 3: システムが提供する自動収集データ

分類	項番	タグ項目	収集データ	入力
要件定義	14	規模 [推移]	ユースケース図数 ユースケース数 アクター数	JUDE で作成した モデリングファイル
	15	変更 [推移]	ユースケース図追加数 ユースケース図削除数 ユースケース追加数 ユースケース削除数 アクター追加数 アクター削除数	
設計	16	規模 [推移]	クラス図数 シーケンス図数 ステートマシン図数 アクティビティ図数	
	17	変更 [推移]	クラス図追加数 クラス図削除数 シーケンス図追加数 シーケンス図削除数 ステートマシン図追加数 ステートマシン図削除数 アクティビティ図追加数 アクティビティ図削除数	
プログラミング	19	規模 [推移]	ソースコード行数	Subversion, CVS の リポジトリ
	20	変更 [推移]	追加行数 削除行数 更新回数	
	21	複雑度	ＣＫメトリクス	
品質	29	欠陥件数 [推移]	バグ報告数	Gnats, 影舞の ログファイル
	30	欠陥対応件数 [推移]	バグクローズ数	
	31	欠陥密度	バグ報告数 / ソースコード行数	
	32	欠陥指摘率	バグ報告数 / 消化テスト項目数	

4.5 自動データ収集手法

システムが提供する自動収集のデータ項目を表 3 に示す．データの収集手法については以降の項で説明する．

4.5.1 分類 [要件定義，設計] のメトリクス計測

分類 [要件定義]，[設計] の自動データ収集手法について説明する．システムは UML モデリングツール JUDE[15] で作成されたＵＭＬモデルからのデータ収集に対応している．データ収集手法の概要図を図 16 に示す.

JUDE では，JUDE のモデルデータを活用するアプリケーションソフトウェアを開発するための Java インターフェース群である JUDE API を公開している．この JUDE API は

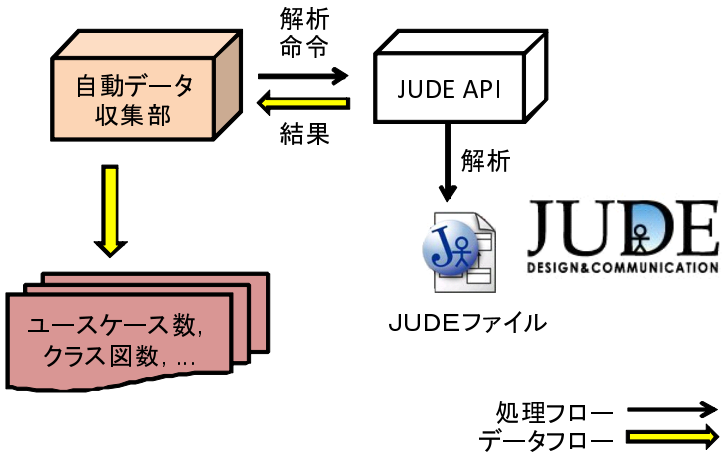

図 16: 要件定義，設計のデータ収集手法概要

JUDE のインストールディレクトリに存在し，JUDE Professional のインストールディレクトリには jude-api.jar と jude-pro.jar が，JUDE Community のインストールディレクトリには jude-api.jar と jude-community.jar が含まれている[2]．自動データ収集部はこの JUDE API を利用し，JUDE で作成された UML モデルからデータ計測を行う．必要となるのは jude-api.jar と，jude-pro.jar か jude-community.jar のどちらかである．

4.5.2 分類 [プログラミング] のメトリクス計測

分類 [プログラミング] のデータ収集手法について述べる．ソフトウェア開発プロジェクトではソースコードの管理にバージョン管理システムが用いられている．そこで本システムは比較的高いシェアと認知度をもつフリーのバージョン管理ツールである Subversion，CVS のリポジトリからデータを収集する．

概要図を図 17 に示す．バージョン管理ツールのリポジトリから，指定した日付の最終コミットリビジョンのソースコードをエクスポートし，メトリクスを計測する．計測はメトリクス計測プラグインプラットフォーム MASU[18] を利用する．MASU の出力結果から，タグ項目規模に該当するソースコード行数，複雑度に該当する CK メトリクス [19] を収集する．

また，タグ項目変更に関しては svn diff や cvs diff のコマンドを利用する．概要図を図 18 に示す．

[2]2009 年 9 月 15 日，JUDE は astah*に改名された．それに伴い，JUDE Professional は astah* professional，JUDE Community は astah* community に改名された．API は jude-api.jar は astah-api.jar に，jude-pro.jar は ashta-pro.jar に，jude-community.jar は astah-community にファイル名が変更されている．自動データ収集手法は astah-api.jar，astah-pro.jar，astah-community.jar を用いても有効であることを確認している．

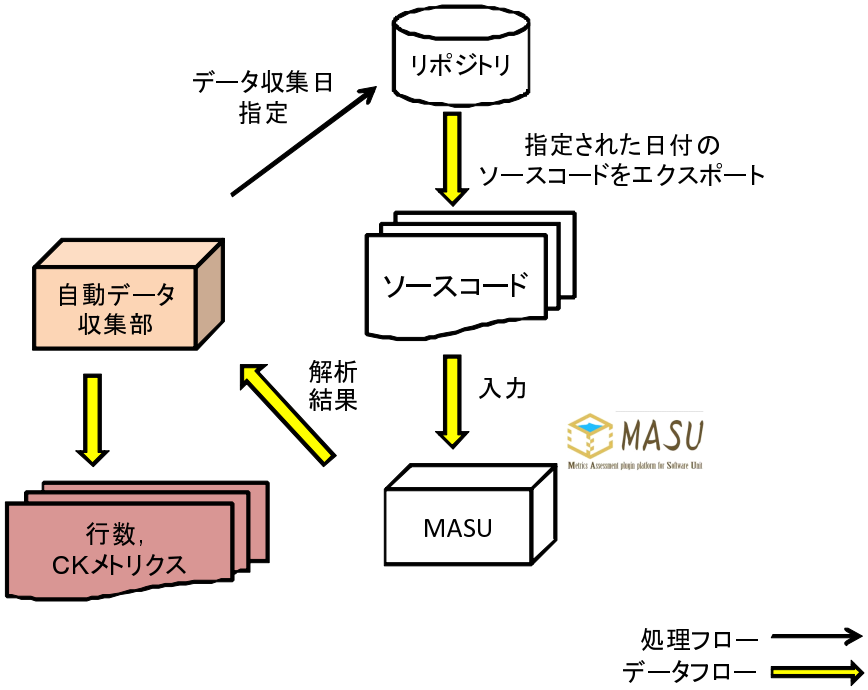

図 17: 規模，複雑度のデータ収集手法概要

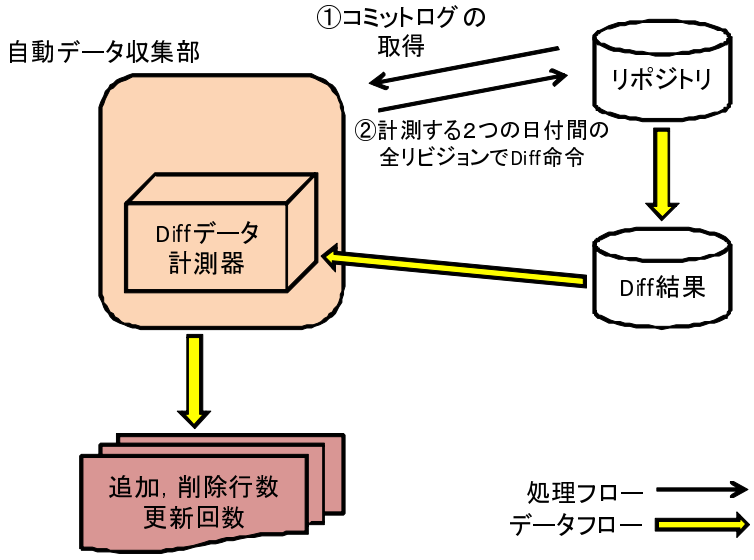

図 18: 変更のデータ収集手法概要

まずリポジトリからコミットログを取得する．コミットログから，計測する 2 つの日付間でコミットされたリビジョン番号を調べ，その全リビジョン間で Diff 命令を行う．例えば，2 月 1 日から 2 月 4 日の間でコミットされたリビジョン番号が 700，701，702，703 ならば，700-701 間，701-702 間，702-703 間で Diff を出力する．この Diff の出力結果を入力とした Diff データ計測器から，ファイルごとの追加行数，削除行数，コミット数を取得する．Diff データ計測器は本研究で実装した．

4.5.3 分類 [品質] のメトリクス計測

分類 [品質] のデータ収集手法について説明する．たいていのソフトウェア開発プロジェクトではプロジェクトのバグ登録，及び修正状況の追跡をするシステムとしてバグトラッキングシステムが用いられている．バグトラッキングシステムのデータベースには，報告されたバグの種類や報告日，修正されたか否かなどの情報が蓄積されている．バグトラッキングシステムはこの情報をログとして出力することができ，本システムはこのログ情報からタグ項目 29 番の欠陥件数，30 番の欠陥対応件数の情報を収集する．31 番の欠陥密度についてはソースコード行数の入力が必要であり，32 番についてはテストの消化数が必要である．それらの入力があれば，自動計測が可能である．

4.6 実行例

本節ではシステムの実行例を挙げる．

図 19 がタグデータ定義の編集画面である．発注者，受注者間の協議のうえで決定した収集対象タグデータのタグデータ定義を編集する．タグデータ定義はプロジェクトの新規作成から終了までいつでも編集可能なので，プロジェクトの途中で収集するタグデータ項目が変更となったときは任意のタイミングで項目の追加，除去ができる．

タグデータ定義に従いデータ入力部が入力フォーマットを生成する．図 20 に示す入力画面はウィザード形式になっており，分類ごとにプロジェクトのデータを入力する．4.3 節で述べたように，プロジェクトの終了まで一定の期間でタグデータ入力ウィザードを起動し，タグデータの入力，自動収集を行う．

入力，収集されたデータはタグデータリポジトリに蓄積される．図 21 は蓄積されたデータをテーブルで表示したものである．

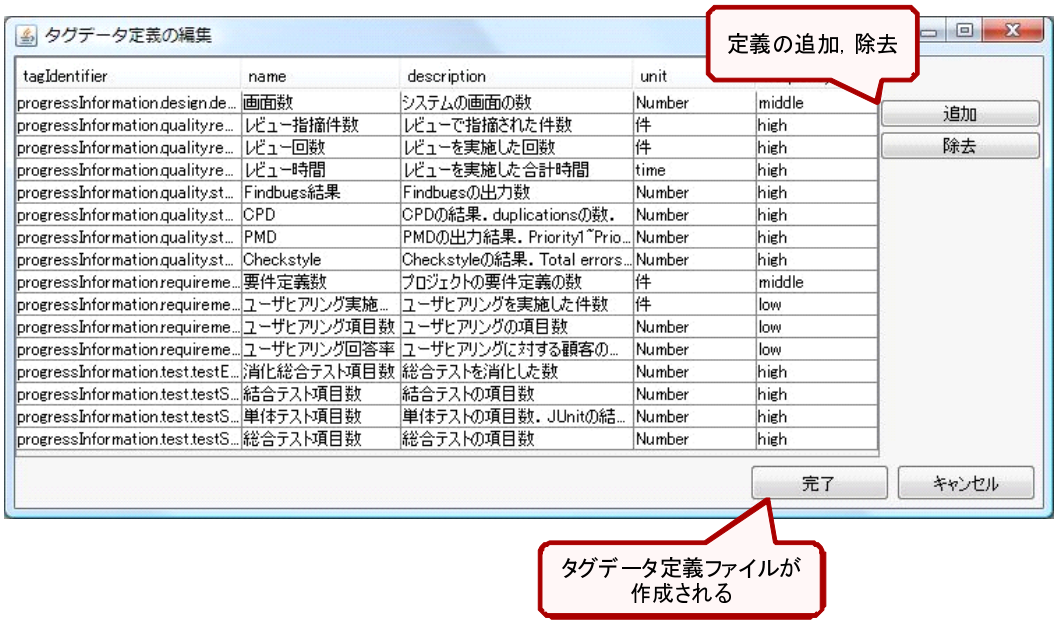

図 19: タグデータ定義編集画面

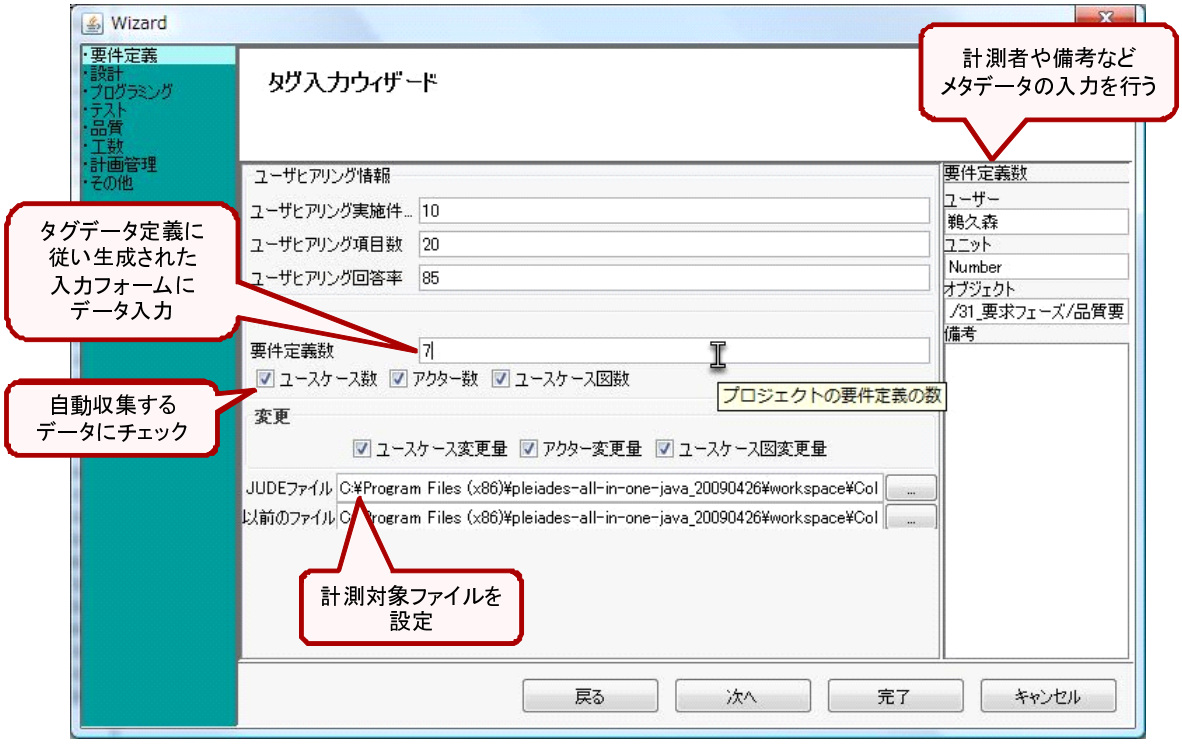

図 20: データ入力ウィザード

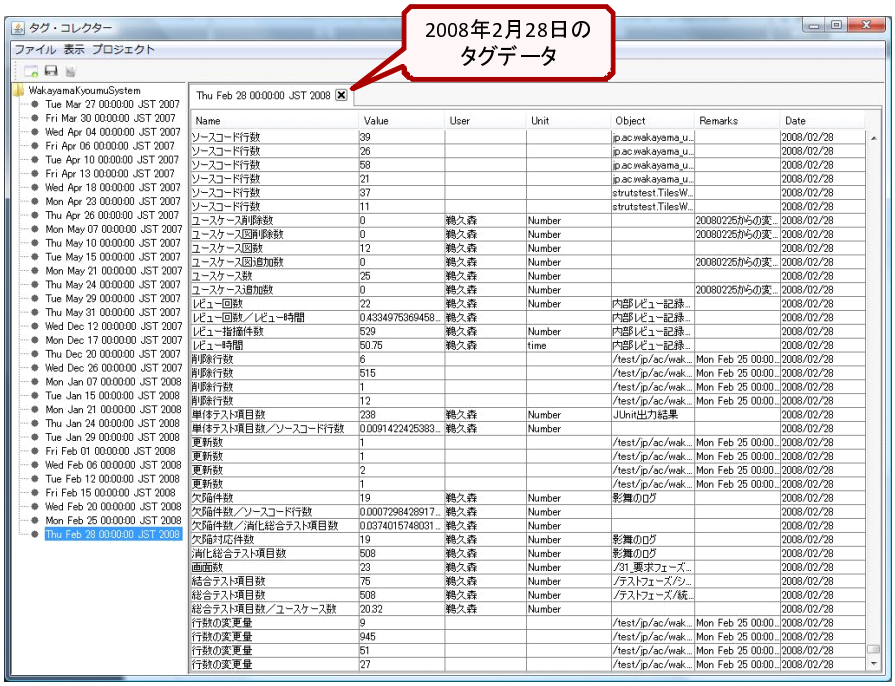

図 21: 蓄積されたデータの出力

5 実験

5.1 実験目的

本研究で開発したソフトウェアタグ生成システムを用いて，実際のプロジェクトからソフトウェアタグを生成する．ソフトウェア生成にかかるコストを計測し，本システムの有用性を検証する．

5.2 対象プロジェクト

ITSpiral[20] で作成された実プロジェクト教材に対して，本システムを利用してソフトウェアタグを生成した．これは大学の履修登録プログラムを開発した際に得られた開発データであり，要件定義，設計，実装，テスト等の開発プロセスでさまざまな実証データが存在する．プロジェクトはウォーターフォールモデルで開発され，設計プロセス終了に 2 ヶ月，実装からプロダクトの提出までに 3 ヶ月の時間を要している．総開発期間は 5ヶ月で，総データ量は 450MB である．このプロジェクトのエンピリカルデータを表 4 に示す．

なお，表 5 が示すように開発支援ツールを各開発プロセスで使用しており，収集データの一部としてツールの実行結果やプロジェクトデータが存在する．

表 4: 適用対象プロジェクトのエンピリカルデータ

プロセス名	エンピリカルデータ
要件定義	要求仕様書，業務フロー図，画面一覧 ユースケース図，アクター定義書，品質要求書
分析・設計	クラス図，コラボレーション図，シーケンス図 アーキテクチャ仕様書，エンティティクラス設計書 コントロールクラス設計書，バウンダリクラス設計書
実装	ソースコード，API ドキュメント，リポジトリ 静的チェック記録
テスト	テスト計画書・仕様書
プロジェクト管理	内部・外部レビュー記録，プロジェクト管理記録 変更経緯説明書

表 5: プロジェクトで使用された開発支援ツール

ツール名	説明	使用開発プロセス
JUDE	UML モデリングツール	要求・分析・設計
Eclipse	統合開発環境	実装
影舞	バグ管理ツール	テスト
Subversion	構成管理ツール	実装

5.3 収集タグデータ項目

表 6 に実験で収集したタグデータ項目を示す．収集タグデータ項目は 53 項目，そのうち自動収集したものは 40 項目，手動入力したものは 13 項目である．

5.4 実験手順

実験手順を以下に示す．

1. タグデータ定義編集部を使って収集するタグデータのタグデータ定義を追加する．
2. 表 4 のエンピリカルデータからデータを集める．
3. システムの入力ウィザードを使ってタグデータを入力，計測する．
4. ソフトウェアタグの出力．

タグデータ定義編集部で追加したタグデータ定義を表 7 に示す．表 7 で示しているデータ項目は，タグデータ入力部により提供された入力フィールドに入力した項目になる．それ以外の項目については自動収集を行っている．

5.5 評価方法

システムを利用したソフトウェアタグ生成のコストを検証する．評価基準は以下の項目である．

- タグデータ定義の追加に要した時間
- データ入力に要した時間
- 自動データ収集に要した時間

5.6 結果

表 7 に示す 13 項目のタグデータ定義を追加するのにかかった時間は 9 分 50 秒だった．表 8 に入力時間と自動収集時間の結果を示す．

表 6: 収集タグデータ項目

分類	項番	タグ項目	収集タグデータ項目	収集方法
要件定義	14	規模 [推移]	ユースケース図数	自動
			ユースケース数	自動
			アクター数	自動
			要件定義数	手動
	15	変更 [推移]	ユースケース図の追加数	自動
			ユースケースの追加数	自動
			アクターの追加数	自動
			ユースケース図の削除数	自動
			ユースケースの削除数	自動
			アクターの削除数	自動
設計	16	規模 [推移]	画面数	手動
			クラス図数	自動
			シーケンス図数	自動
			ステートマシン図数	自動
			アクティビティ図数	自動
	17	変更 [推移]	クラス図の追加数	自動
			シーケンス図の追加数	自動
			ステートマシン図の追加数	自動
			アクティビティ図の追加数	自動
			クラス図の削除数	自動
			シーケンス図の削除数	自動
			ステートマシン図の削除数	自動
			アクティビティ図の削除数	自動
プログラミング	19	規模 [推移]	ソースコード行数	自動
	20	変更 [推移]	行数の変更量	自動
			追加行数	自動
			削除行数	自動
			更新回数	自動
	21	複雑度	CK メトリクス.WMC	自動
			CK メトリクス.NOC	自動
			CK メトリクス.CBO	自動
			CK メトリクス.RFC	自動
			CK メトリクス.LCOM	自動
			CK メトリクス.DIT	自動
テスト	22	規模 [推移]	単体テスト項目数	手動
			結合テスト項目数	手動
			総合テスト項目数	手動
	24	密度	単体テスト項目数 / ソースコード	自動
			総合テスト項目数 / ユースケース数	自動
	25	消化	消化総合テスト項目数 / 総合テスト項目数	自動
			消化総合テスト項目数	手動
品質	26	レビュー状況	レビュー回数	手動
			レビュー時間	手動
	27	レビュー作業密度	レビュー回数 / レビュー時間	自動
	28	レビュー指摘率 [推移]	レビュー指摘件数	手動
	29	欠陥件数 [推移]	バグ報告数	自動
	30	欠陥対応件数 [推移]	バグクローズ数	自動
	31	欠陥密度	バグ報告数 / ソースコード行数	自動
	32	欠陥指摘率	バグ報告数 / 消化総合テスト項目数	自動
	33	静的チェックの結果	checkstyle の検出数	手動
			findbugs の検出数	手動
			pmd の検出数	手動
			pmd-cpd の検出数	手動

表 7: 追加したタグデータ定義

name	description
画面数	システムの画面の数．画面一覧.doc に記載.
レビュー指摘件数	レビューで指摘された件数. 議事録に記載.
レビュー回数	レビューを実施した回数．議事数でカウント.
レビュー時間	レビューを実施した合計時間．議事録に記載.
Findbugs	Findbugs の結果
CPD	CPD の結果
PMD	PMD の結果．Priority1 から Priority5 まで.
Checkstyle	Checkstyle の結果．トータル出力数.
要件定義数	品質要求.doc に記載されている件数.
単体テスト項目数	Junit で実行したテスト項目数
結合テスト項目数	システムテスト仕様書に記載.
総合テスト項目数	統合テスト仕様書に記載.
消化総合テスト項目数	総合テスト仕様書に記載.

6 考察

タグ定義を 13 項目追加するためにかかった時間は 9 分 50 秒だった．１件の追加におよそ 46 秒ほどかかる計算になるが，プロジェクトの初めに１度追加するだけなので，入力時間の合計１時間 10 分に比べればそれほど大きなコストとはならなかった.

2007 年 3 月 27 日から 5 月 31 日までの期間は要求，分析，設計のフェーズであったため，ソースコードが存在しない．また，欠陥件数に関する影舞のログもまだ存在していなかったため，この間の自動計測は JUDE ファイルからタグ項目要件定義，設計のデータを収集しているのみである．そのため，自動収集は１秒前後程度の非常に短い時間で終わっている.

2007 年 12 月 12 日から自動収集時間がそれまでに比べて大きくなっている．これはこの日からプログラミングが開始されたためである．リポジトリからのソースコードのエクスポート，MASU の実行，Diff の実行に時間がかかった結果と言える．同時に入力時間も増えているが，リポジトリの登録を含む自動収集の設定に時間がかかり，それまでと比べて 5 分以上余分に時間がかかった．リポジトリの登録などは一度行えば以降は行う必要がないため，後の計測は 2 分 30 秒ほどの入力時間ですんでいる．自動収集時間についてはそれ以降若干増加傾向にあるが，これはソースコードの規模が大きくなっているためと思われる.

31 日分のデータ入力時間の平均値は 2 分 17 秒である．ここで１ヶ月にかかる人的コスト

表 8: 実験結果

計測日	入力時間	自動収集時間
2007/03/27	0:01:54	0:00:01
2007/03/30	0:02:54	0:00:00
2007/04/04	0:02:27	0:00:00
2007/04/06	0:01:45	0:00:01
2007/04/10	0:01:43	0:00:02
2007/04/18	0:01:19	0:00:01
2007/04/23	0:01:42	0:00:01
2007/04/26	0:02:05	0:00:01
2007/05/07	0:01:13	0:00:01
2007/05/10	0:01:13	0:00:01
2007/05/15	0:01:16	0:00:01
2007/05/21	0:01:14	0:00:01
2007/05/24	0:01:29	0:00:01
2007/05/29	0:01:29	0:00:02
2007/05/31	0:01:09	0:00:02
2007/12/12	0:06:04	0:00:41
2007/12/17	0:03:07	0:00:55
2007/12/20	0:02:26	0:00:55
2007/12/26	0:02:20	0:01:02
2008/1/7	0:03:13	0:00:52
2008/1/15	0:02:55	0:01:18
2008/1/21	0:02:23	0:00:52
2008/1/24	0:03:12	0:00:49
2008/1/29	0:03:18	0:00:52
2008/2/1	0:02:25	0:00:51
2008/2/6	0:02:24	0:00:51
2008/2/12	0:02:46	0:00:48
2008/2/15	0:02:32	0:00:53
2008/2/20	0:02:34	0:00:52
2008/2/25	0:02:15	0:00:52
2008/2/28	0:02:01	0:00:49

	入力時間	自動収集時間
合計	1:10:47	0:14:28
最高値	0:06:04	0:01:18

を考える．

開発プロジェクトでは，プロジェクトメンバーに作業進捗を週報，日報といった形で提出させる．そこでタグデータの収集日を週報に合わせ，1 週間に 4 回と仮定すると，1ヶ月に 4 回データの収集を行うことになる．1 日分の入力時間を結果の平均値からとって 2 分 17 秒とすると 1ヶ月の入力時間は式 (1) の通り 10 分程度となる．

$$2 : 17 * 4(\text{日}) \approx 10(\text{分/月}) \approx 0.16667(\text{時間/月}) \tag{1}$$

1 人月を 8 時間，20 日で換算すると，１人月の時間は式 (2) となる．

$$1\ \text{人月} = 8(\text{時間}) * 20(\text{日}) = 160(\text{時間/月}) \tag{2}$$

式 (1) を式 (2) で割れば，１ヶ月あたりのタグ収集にかかるコストが導き出せる．

$$\frac{0.16667}{160} = 0.00104125 \approx 0.001(\text{人月}) \tag{3}$$

これは 1 人月を 100 万円とすると１ヶ月あたりにかかる費用は 1000 円程度となる．

ソフトウェア開発データ白書 2008[21]p.92 によると，ソフトウェアベンダー 21 社 1440 プロジェクトの工数の統計平均は 27485 人時である．これを式 (2) に従い，人月に直すと約 172 人月になる．172 人月に対してソフトウェアタグ生成に必要な人的コスト 0.001 人月は非常に小さい値である．したがって，本システムを用いたソフトウェアタグ生成はコストが低いと言える．

入力時間が短くなった要因として，4.4.4 項で述べた前回の入力データを入力フィールドに保持しておくサポートが挙げられる．user,unit,obj は，初めの入力から最終日の入力まで変更することがなかったため，入力には時間がかからなかった．また，remarks についても追加数や削除数などの項目については「2008 年 1 月 29 日からの追加数」などの入力を行ったものの，remarks に変更がない項目も存在したため，この場合は入力サポートの効果があったと言える．

7 Threats to Validity

今回追加したタグデータ定義は 13 項目であったため，追加にそれほど時間はかからなかった．しかし実際のソフトウェア開発では入力する項目はこれよりもっと多い．例えば 100 項目の追加を考えた場合，これをすべて人手でタグデータ定義を追加するのは手間がかかる．したがって，基本的なデータについてはあらかじめタグデータ定義をシステム側から提供する必要があるだろう．

同時に，手動での入力項目は 13 項目と少なかったため，項目が増えるとそれに比例してデータ入力の時間も必要となる．

また，実験では 4.4.1 節で述べたタグデータの remarks などの入力も行ったが，これらのデータ量が多い場合，入力にさらに手間がかかる．user に関しても今回はすべて被験者の名前を入力したが，これの変更が多ければ入力時間が延びるだろう．

結果には示していないが，自動計測時間はリポジトリからのエクスポート，MASU の実行時間が主に占めている．実験対象のプロジェクトはソースコード行数がおよそ 26000 行だったが，さらに巨大な規模のソフトウェアになると自動計測時間はさらに延びるだろう．参考までにソースコード２１万行のオープンソースプロジェクトに対して行数とＣＫメトリクスの計測を行ったところ，計測時間は 6 分 52 秒となった．ただし，ネットワーク上のリポジトリに対して計測を行ったので通信によるオーバーヘッドは生じている可能性がある．

UML 図やバグトラッキングシステムからの計測は１秒前後であったため，規模が大きくなってもオーバーヘッドは少ないと思われる．

今回の実験では被験者が開発者自身なので，システムに対する慣れが結果に影響を与えていると思われる．システムはソフトウェアタグやソフトウェア開発の定量的管理にそれなりに理解がある者を対象としているものの，初めてシステムを利用するものに対しては利用マニュアルなどを作成する必要がある．

システムの利用者がソフトウェアタグを生成するプロジェクトについて全く理解がない場合，つまり入力すべきデータの根拠となるファイルなどがどこにあるのか分からないような場合は結果よりさらに時間がかかる．今回，被験者はプロジェクトの要件定義書や議事録，静的チェックの結果ファイルなどがどこに存在するか理解していたので，データの入力にとまどうことはなかった．システムの利用者はプロジェクトについてある程度の理解をしているか，もしくは入力するプロジェクトデータを各担当者がまとめて提出するような体制が必要である．

8 関連研究

文献 [22] ではソフトウェアタグデータ収集システムのプロトタイプの開発を行っている．こちらの研究は自動データ収集についてのみを取り扱っており，本研究が提案した収集タグデータ項目をユーザが選択するといったことはできない．文献 [23] では本研究でも実験対象プロジェクトとして用いた ITSpiral 実プロジェクト教材のソフトウェアタグを作成し，ソフトウェアタグからのプロジェクトの分析を行っている．

StagE プロジェクトの参考文献を挙げる．文献 [24] では本論文の 2 章，3 章でも紹介した StagE プロジェクトとソフトウェアタグについて詳しく解説されている．文献 [25] ではソフトウェアタグ普及に向けた法的議論と利用技術基盤について，文献 [26] ではソフトウェアタグ支援ツールの開発について解説されている．文献 [27] はユーザ・ベンダ各々の作業の責任分担を調査し，各々の作業が十分に行われているかを定量的に判断する要求品質管理手法を提案している．文献 [5] では，発注者がプロジェクトの進捗状況を把握する際にソフトウェアタグを利用するという状況を想定した可視化ツールを提案している．従来の可視化ツールでは下流工程の可視化に重点を置いていたが，この研究では開発工程全体を網羅しているソフトウェアタグを用いるため，上流工程を含めた可視化を行うことが可能である．文献 [6] ではソフトウェアタグに含まれる設計文書メトリクスを用いて，低品質モジュールの予測を試みている．設計文書メトリクスを利用することで設計工程完了時に予測を行い，実装工程における品質改善活動に寄与することができると述べられている．

文献 [28] では，定量的管理を取り入れたソフトウェア開発プロセスのためのオーサリング・テーラリングフレームワークを提案している．オーサリングとはプロセスの構築作業，テーらリングとは作業内容の修整作業を指す．文献では提案フレームワークをもとに定量的管理計画の立案を支援するシステム「AQUAMarine」を開発し，その評価を行っている．

リポジトリの履歴データを利用した参考文献を挙げる．文献 [29] ではリポジトリの履歴データをもとにしてソースコード修正箇所のグルーピングを行ったデータベースを作成し，それをもとに開発者に修正箇所を提示するシステム ROSE を提案している．文献 [30] は CVS を用いた分析に必要な事前処理の収集，形式化を試みている．文献 [31] では Bugzilla のバグ情報とソースコードの import 情報マッピングし，バグ予測を行う手法を提案している．文献 [32] では 5 つのオープンソースソフトウェアと 1 つのクローズドソースソフトウェアのバグトラッキングシステムやバージョン管理システムからデータを収集し，プロジェクトごとにデータの比較を行っている．

ソースコードのメトリクス計測に関する参考文献を挙げる．文献 [33] では計測するメトリクスや入力が同じ場合，結果は異なるのかをテーマにソフトウェアメトリクス計測ツールの比較を行っている．実験では 9 つの計測ツールに対し，100 プロジェクトのソースコードを

入力として結果の比較を行っている．ソフトウェアタグ規格 [9] のタグ項目 3 番「静的チェックの結果」に関連して，静的バグ解析ツールに関する研究である文献 [34] を挙げる．この論文では IntelliJ IDEA, FindBugs, Jlint の 3 つの静的バグ解析ツールのパフォーマンスの比較を行っている．

9 あとがき

本論文では，ソフトウェアトレーサビリティの実現を目的としたソフトウェアタグ生成システムの提案を行った．2 章ではソフトウェアトレーサビリティの必要性について述べ，3 章ではトレーサビリティを実現するソフトウェアタグについて述べた．4 章ではソフトウェアタグの生成方法に対する要求について整理し，それらを解決するためのシステムの設計を行い，システムの構成について述べた．5 章では実プロジェクトに対して本システムを用いてソフトウェアタグの生成を行い，そのコストについて評価を行った．その結果，本システムによるソフトウェアタグの生成は非常に低コストで行えることを証明した．

今後の課題について述べる．

まず，標準タグデータ定義を作成する必要がある．7 章で述べたように入力するタグデータ項目が増加すると，それだけタグデータ定義を作成する必要があるため，高コストにつながる．そのためシステム側で基本的なデータについてはタグデータ定義を提供する予定である．

生成コストを下げるため，また自動収集の適応可能対象となるプロジェクトを増やすために，自動収集機能を拡張することも考えられる．UML モデリングツールからの計測は現状 JUDE[15] だけであるため，Microsoft Visio[17] や IBM Rational Software Modeler[16] などへの対応を検討する必要がある．バグトラッキングシステムにおいても，Bugzilla[35] など認知度の高いものについては対応していく必要がある．

また，発注者にわかりやすく開発状況や品質を示せるようなタグデータの表示形式を考える必要がある．プロジェクトの途中で，一般的なプロジェクトデータの統計値 [21] と比較できるような形式でタグデータを出力し，提示することで発注者は進捗を確認することができる．また，一般的なプロジェクトと比較することにより発注したソフトウェアのプロジェクトに対して安心感をもつことができる．また，データに疑問があった場合には受注者に確認をとることができるため，プロジェクトの問題早期発見につながる．

最後に，システムの現場評価が必要である．これについては協力企業にシステムの利用を依頼し，評価をお願いする．またネットワーク上での公開も予定している．現場での利用，公開に向けてアプリケーションとしての利便性，信頼性を上げ，マニュアルの整備を行う．さらに利用者からの評価を反映し，システムの改善を行っていく予定である．

謝辞

本研究の全過程を通して，常に適切な御指導および丁寧な御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 楠本真二教授に心から深く感謝申し上げます.

本研究に適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 岡野浩三准教授に心から感謝いたします.

本研究に対して，適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後芳樹助教に深く感謝いたします.

本研究に対して，適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 柿元健特任助教に深く感謝いたします.

システムの実装をお手伝い頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 1 年 山田慎也氏に感謝いたします.

その他の大阪大学大学院情報科学研究科コンピュータサイエンス専攻楠本研究室の皆様のご協力に感謝致します.

参考文献

[1] 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研究への取り組み”, 情報処理学会, Vol. 45, No. 7, pp. 722–728, 2004.

[2] 「動かないコンピュータ」の解決に裁判は役立つか: “http://itpro.nikkeibp.co.jp/free/ITPro/OPINION/20040609/145573/,”.

[3] StagE プロジェクト: “http://www.stage-project.jp/,”.

[4] E. R. Harold: “XML バイブル”, 日経 BP 社, 2001.

[5] 奥村哲也, 大蔵君治, 伏田享平, 川口真司, 名倉正剛, 飯田元. “ソフトウェアタグの運用を支援する開発履歴可視化ツール”,. 情報処理学会第 166 回ソフトウェア工学研究会, 情報処理学会研究報告, 第 2009-SE-166 巻, pp. 1–8, October 2009.

[6] 片山真一, 大蔵君治, 伏田享平, 川口真司, 名倉正剛, 門田暁人, 飯田元. “ソフトウェアタグを用いた設計文書メトリクスからの 低品質モジュールの予測”,. 電子情報通信学会技術研究報告, 第 109 巻, pp. 067–072, December 2009.

[7] 日経コンピュータ 特集 1：ユーザーの言い分ベンダーの言い分, pp. 40–55, 2008 年 10 月 1 日号.

[8] 開発のための CMMI.(CMMI-DEV) 1.2 版: “http://www.sei.cmu.edu/cmmi/translations/japanese/models/dev-v12-abstract-j.html,” 技術報告書, Vol. CMU/SEI-2006-TR-008, .

[9] ソフトウェアタグ規格第 1.0 版: “http://www.stage-project.jp/kanri/data/dl/20081029151602.pdf,”.

[10] 松本健一: “エンピリカルソフトウェア工学の現状と展望：SEL が残した 13 の教訓”, *SEC journal*, No. 2, pp. 6–13, 2005.

[11] subversion.tigris.org: “http://subversion.tigris.org/,”.

[12] C. C. V. System: “http://www.nongnu.org/cvs/,”.

[13] Gnats: “http://www.gnu.org/software/gnats/,”.

[14] バグトラッキングシステム影舞: “http://www.daifukuya.com/kagemai/,”.

[15] astah*(旧 JUDE)): “http://astah-users.change-vision.com/ja/,”.

[16] IBM Rational ソフトウェア *http://www-06.ibm.com/jp/software/rational/*.

[17] M. O. Visio: “http://office.microsoft.com/ja-jp/visio/default.aspx,”.

[18] 三宅達也, 肥後芳樹, 楠本真二, 井上克郎: “多言語対応メトリクス計測プラグイン開発基盤 MASU の開発”, 電子情報通信学会論文誌 D, Vol. J92-D, No. 9, pp. 1518–1531, 2009.

[19] S. R. Chidamber and C. F. Kemerer: “A Metrics Suite for Object-Oriented Design,” *IEEE Trans. Software Eng.*, Vol. 20, pp. 476–493, 1994.

[20] IT スパイラル: “http://it-spiral.ist.osaka-u.ac.jp/index.html,”.

[21] 独立行政法人情報処理推進機構（IPA）, ソフトウェア・エンジニアリング・センター（SEC）. ソフトウェア開発データ白書 2008. 日経 BP 社, 2008.

[22] 西川倫道, 鵜久森将隆, 山田慎也, 楠本真二: “ソフトウェアタグデータ収集システムの試作”, ソフトウェア信頼性研究会第 5 回ワークショップ論文集, pp. 57–65, 2009.

[23] 山田慎也, 西川倫道, 鵜久森将隆, 楠本真二: “実プロジェクトデータからのソフトウェアタグ作成とその分析”, 情報処理学会, 2009.

[24] 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－ StagE プロジェクトとソフトウェアタグ－”, *SEC journal*, Vol. 5, No. 3, pp. 198–203, 2009.

[25] 久保浩三, 小柴昌也, 角田雅照, 松村知子: “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－ソフトウェアタグ普及に向けた法的議論と利用技術基盤－”, *SEC journal*, Vol. 5, No. 6, pp. 368–376, 2009.

[26] 井上克郎, 楠本真二, 飯田元: “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－ソフトウェアタグ規格とソフトウェアタグ支援ツール－”, *SEC journal*, Vol. 5, No. 5, pp. 234–243, 2009.

[27] 松村知子, 松本健一: “ユーザとベンダ間の協調による要求品質確保のための定量化事例－”, 奈良先端科学技術大学院大学テクニカルレポート, NAIST-IS-TR2009006, 2009.

[28] 伏田享平, 高田純, 米光哲哉, 福地豊, 川口真司, 飯田元: “AQUAMarine: 定量的管理計画立案システム”, *SEC journal*, Vol. 5, No. 4, pp. 244–251, 2009.

[29] T. Zimmermann, P. Weisgerber, S. Dieh, and A. Zeller: "Mining Version Histories to Guide Software Changes," *International Conference on Software Engineering Proceedings of the 26th International Conference on Software Engineering*, pp. 563–572, 2004.

[30] T. Zimmermann and P. Weisgerbe: "Preprocessiong CVS Data for Fine-grained analysis," *1st International Workshop on Mining Software Repositories*, pp. 2–6, 2004.

[31] A. Schroter, T. Zimmermann, and A. Zeller: "Predicting Component Failures at Design Time," *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pp. 18–27, 2006.

[32] A. Bachmann and A. Bernstein: "Software process data quality and characteristics: a historical view on open and closed source projects," *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pp. 119–128, 2009.

[33] R. Lincke, J. Lundberg, and W. Löwe. "Comparing software metrics tools,". In *ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis*, pp. 131–142, New York, NY, USA, 2008. ACM.

[34] F. Wedyan, D. Alrmuny, and J. M. Bieman: "The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction," *ICST Proceedings of the 2009 International Conference on Software Testing Verification and Validation*, pp. 141–150, 2009.

[35] Bugzilla.org: "http://www.bugzilla.org/,".