軽量開発プロセスにおける Trac を用いたメトリクスの収集・蓄積・利用

Using Trac as a Platform for Empirical Data Collection in Lightweight Development

株式会社 NTT データ　技術開発本部
NTT DATA Corporation　Research and Development Headquarters

○大杉 直樹	伏田 享平 [1)]	井ノ口 伸人 [1)]	新井 広之 [1)]
丹羽 隆 [2)]	藤貫 美佐 [3)]	戸村 元久 [4)]	木谷 強 [5)]
○Naoki Ohsugi	Kyohei Fushida[1)]	Nobuto Inoguchi[1)]	Hiroyuki Arai[1)]
Takashi Niwa[2)]	Misa Fujinuki[3)]	Motohisa Tomura[4)]	Tsuyoshi Kitani[5)]

Abstract　This paper describes practical example of using *Trac* as a platform for empirical data collection in lightweight system development. Many project managers have used various metrics such as size, effort or number of bugs for software intensive system development. These metrics are vital for management although non-negligible cost is required for preparing effective combination of measurement tools, rules and continuous monitoring to collect reliable data; however, small and medium-sized projects generally have not enough cost to do them because of the budget limitation. This paper describes example of low cost data collection in two systems development. The examples are small (2 development personnel from 4 at a peak period) and middle (26 personnel from 80 at a peak period) sized projects to develop in-house enterprise systems. During 29 months, 10 basic metrics and 7 derived metrics regarding effort, size and quality were collected and used for progress management, estimation, and quality assurance.

1.　はじめに

開発規模、工数、レビューやテストの量、発見したバグ数など、システム開発の過程で得られる数値データ(メトリクス)を、システム開発の見積り・進捗管理・品質管理などに利用できる。例えば、開発対象の規模と工数のメトリクスを収集し、分析することで工数や開発期間の見積りに利用できる[5]。また、コーディングの過程で混入したバグの数やテスト・レビューの実施量[2][19]、あるいはソースコードの複雑さを数量化したメトリクス[12]を品質管理に利用できる。

信頼性の高いメトリクスを収集するためには、適切な計測・蓄積ツールとルールを準備し、ルールに従った計測活動とルール遵守を担保する監視活動が必要となる。これは、工数やバグ数など多くのメトリクスの計測に人間の判断が必要なためである。例えば、「結合テストで検出されたが、本来は単体テストで検出すべきだったバグの数」などである。実務でメトリクスを利用するためには、メトリクス収集に関わる全要員が計測ルールを理解し、ルール通り計測されていることを誰かが監視し、是正する体制が必要となる。

このためメトリクスの収集・蓄積・利用には、主に学習と運用に無視できない大きさのコストが必要となる。利用するツールの調達・立ち上げ・運用に加え、ルールの整備、開発要員へのルール周知と教育、要員

株式会社 NTT データ 技術開発本部 プロジェクトマネジメント・イノベーションセンタ シニアエキスパート
Senior Expert, Project Management Innovation Center, R&D Headquarters, NTT DATA Corporation
〒135-8671 東京都江東区豊洲 3-3-9　豊洲センタービルアネックス　Tel: 050-5546-2529
Toyosu Center Bldg. Annex, 3-9, Toyosu 3-chome, Koto-Ku, Tokyo 135-8671 Japan　Tel: +81 50 5546 2529

1) 株式会社 NTT データ 技術開発本部 プロジェクトマネジメント・イノベーションセンタ 主任
Assistant Manager, Project Management Innovation Center, R&D Headquarters, NTT DATA Corporation
2) 株式会社 NTT データ 技術開発本部 ALM ソリューションセンタ 部長
Senior Manager, ALM Solution Center, R&D Headquarters, NTT DATA Corporation
3) 株式会社 NTT データ 技術開発本部 プロジェクトマネジメント・イノベーションセンタ 部長
Senior Manager, Project Management Innovation Center, R&D Headquarters, NTT DATA Corporation
4) 株式会社 NTT データ技術開発本部 プロジェクトマネジメント・イノベーションセンタ センタ長
Head of Project Management Innovation Center, R&D Headquarters, NTT DATA Corporation
5) 株式会社 NTT データ 執行役員 技術開発本部 本部長
Senior Vice President, Head of R&D Headquarters, NTT DATA Corporation

によるデータ計測とツールへの入力、メトリクス計測結果を確実にレビュー・是正するためのプロセス整備とその運用にもコストが必要となる。メトリクスの分析に携わる担当者だけでなく、データを入力する開発専任要員も学習工数が必要となる。

一方、多くの中小規模開発ではプロジェクト管理活動に割ける要員や工数が限られており、メトリクスの収集や利用に大きなコストを確保し難い。例えば、筆者らが参画したピーク時要員 80 名の社内システム開発プロジェクトでは、進捗管理や品質管理などのルールと環境を整備し、運用する管理業務の専任担当者は 2 名であり、維持保守フェーズで開発要員が 26 名まで減少した際は、2 名が開発作業と兼務で管理業務を実施した。メトリクスの収集と利用に大きな稼働を捻出することは容易ではない。

本稿では、この課題へ対処した事例として、軽量開発プロセスにおける Trac を用いたメトリクスの収集・蓄積・利用について紹介する。紹介する事例は、一方が中規模（ピーク時開発要員数 80 名、サービス開始後 26 名）、もう一方が小規模（ピーク時 4 名、サービス開始後 2 名）であり、共に社内システムの新規開発および維持保守である。筆者らのうち 2 名がプロジェクトの管理業務担当者として参画し、同様の開発プロセス、計測ツール、ルールを、のべ 29 ヶ月間運用した。各工程の工数、規模、品質について 10 種類の基本メトリクスを測定し、7 種類の導出メトリクスと共に管理業務で利用した。

本稿では、報告する事例で定義・利用したプロセスを軽量開発プロセスと呼ぶ。基本とする開発手順はウォーターフォール開発にアジャイル開発の知見やプラクティスを有効に機能する形で取り入れ、成果物作成以外に必要な付帯的作業の軽量化と、管理工数の低減を目指したものである。当該プロセスは、あるリリースに向けた開発内容を、数日から 10 営業日程度で完了可能な小さい単位（案件と呼ぶ）に分割し、案件毎に設計、実装、テストを繰り返して反復型の開発とした。また、メトリクスの分析や多数参加者による集合型レビューなどの第 3 者的検証でなく、設計書、コード、テストコードなどの成果物レビューで品質を担保した。

プロジェクトの概要や体制などの背景、用いたツールと運用方法、収集したメトリクスの種類や利用方法を紹介することで、同様の課題を抱えるプロジェクトの管理業務担当者に課題に対処するための参考情報を提供すると期待できる。また、開発実務におけるメトリクスの利用事例と課題を共有することで、実務に則した研究を推進するための参考情報を提供できると期待できる。

以降、2 章でメトリクスと、その収集・利用について関連研究を紹介する。3 章で取り組みの背景として、プロジェクトの概要、開発プロセスと Trac の運用方法を説明する。4 章で収集したメトリクスと、その利用方法を紹介する。5 章で利用したメトリクスの有効性と課題を考察し、6 章でまとめと今後の課題を述べる。

2. 関連研究

2.1 メトリクスに関する研究

本稿の事例では、計測が比較的容易で、他のメトリクスと組み合わせて様々な値を導出できる基礎的なものを、規模、コスト、品質を表す基本メトリクスとして利用した。利用したメトリクスの定義や計測方法は、従来研究で提案され、一般に利用されているものと変わらない。ただし、全て管理上の開発単位（案件）毎に記録して利用した点が異なる。本稿で報告する開発においては、複数モジュールや機能を同時に修正する案件が大半であり、モジュール単位、機能単位の数値は管理に利用しにくいためである。

規模を表すメトリクスとして、ソースコードの命令行数を数える LOC（Lines of Code）[11]を利用した。LOC の計測方法は一般に用いられるコメント行や空行を除いた論理コード行数と同じであるが、既存コードに対する追加・修正行（コミットリビジョン間差分）を案件毎に記録した。これによって各案件の開発量を把握し、案件単位で計測した他のメトリクスの値を単位規模あたりの値に標準化するためにも利用した。他の規模メトリクスとして、ユーザから見た機能量を数値化するファンクションポイント[3]なども存在するが、計測に必要な工数が捻出できないため利用しなかった。

コストを表すメトリクスとして、工数と工期を利用した。案件毎に規模、工数、工期を収集し、開発開始前の計画段階や実行中の計画見直しで見積りに利用した。見積りでは工数を規模で除算した値（生産性係数と呼ぶ）や EVM（Earned Value Management）[15]を利用した。統計分析を利用すれば見積りの精度は向上するが[5]、分析に必要な工数が捻出できないため利用しなかった。

品質のメトリクスとして、検出したバグ数、レビューの時間、テストの量（テスト項目数とテストコード行数）を利用した。単位規模あたりのバグ数（バグ密度）やテスト量（テスト項目密度、テストコード密度）を案件

表 1. 報告事例プロジェクトの概要、開発スケジュール、報告対象

	A システム開発プロジェクト	B システム開発プロジェクト
開発対象システム	生産管理系社内システム	情報系社内システム
アーキテクチャ	Web 三層&クライアントサーバ	Web 三層
規模	中規模	小規模
要員数	ピーク時 80 名、サービス開始後 26 名	ピーク時 4 名、サービス開始後 2 名
開発期間	2008/8～2010/3（20 ヶ月間）	2012/4～2013/ 3（12 ヶ月間）
サービス開始	2010/4/1	2013/3/28
維持保守期間	2010/4～2013/9 現在稼働中	2013/3～2013/9 現在稼働中
本稿の報告対象	2009/11～2011/3	2012/4～2013/3
言語・FW など	Python, VB.NET, Trac, PostgreSQL	Java, JS, JSP, Struts2, iBatis, MySQL

毎に算出し、他と値が極端に乖離しているものを検出し、重点的にレビューするために利用した[19]。また、メトリクスとしてデータ収集はしていないが、C0/C1 テスト網羅率[14]を単体テストのレビュー観点として使用した。テストの進捗と検出したバグ数の関係に基づいてプロダクト内の残存バグ数を推測する方法[2]など、複雑な分析は工数が必要となるため利用しなかった。

2.2 メトリクス収集・利用に関する研究

開発実務を阻害することなくメトリクスを円滑に利用するためには、測定ツールの利用が望ましい [9]。例えば、Eclipse Metrics Plugin [18] のように、統合開発環境から簡単に利用できるツールを利用する必要がある。また、Empirical Project Monitor [7] のように、構成管理や変更管理ツールと統合することで、半自動的にメトリクスを測定し、蓄積や分析の機能も備えたツールの利用も有効である。

本稿の事例では、LOC の計測に様々な言語に対応している StepCounter [16] を利用し、リビジョン間差分の LOC を計測するために 80 行程度の自作バッチファイルを作成した。モジュール単位に LOC が出力された CSV 形式のファイルを Microsoft Office Excel [13] で読み込み、LOC とテストコード行数を算出した。これらツールを利用した理由は、長期間保守されており誤動作や故障が少ないこと、必要最低限の機能を備えたツールであるため学習コストが小さいこと、自作ツールとの連携が容易であることである。

また、本稿の事例では、データ収集・蓄積に Trac [8] を利用した。開発要員への作業アサイン時にチケットを発行することをルール化し、各チケットのカスタムフィールドとして各メトリクスを記録した。メトリクス利用時には、Trac のカスタムクエリ（チケット検索機能）で検索し、CSV ファイルとして出力して Microsoft Office Excel で読み込んで集計した。Trac を利用した理由は、Trac、Wiki [6]、Subversion [4] を連動させて変更管理や構成管理を実施しており、利用ルールをうまく定めることで開発実務を阻害せずにメトリクスを収集できること、カスタムクエリで分析に容易なデータを柔軟に素早く抽出できることである。

3. 取り組みの背景

3.1 プロジェクトの概要

本稿の報告事例プロジェクトの概要、および開発スケジュールと報告対象を表 1 に示す。報告対象のプロジェクトは共に社内システムの開発・維持保守であり、A システムについては維持保守体制移行から維持保守・機能追加フェーズ途中（2009/11～2011/3）までを、B システムについては開発開始から完了（2012/4～2013/3）を対象として報告する。この期間は著者らのうち 3 名がプロジェクト管理業務を実施しており、メトリクス収集・利用の状況を把握しているためである。報告対象期間の後も、おおよそ同じ方法でメトリクスの利用を継続しているが、本稿では報告対象としない。

図 1 に報告対象期間の主なプロジェクト体制と管理業務担当要員を示す。A システム開発プロジェクトは、システム全体を管理するグループリーダ（GL）と共にアーキテクチャに精通した有識者が全体の品質管理を担当した。アプリケーション部分を 3 つの業務に分類し、各サブグループ（SG）が各業務の開発を担当する体制とした。また、α 業務担当 SG は管理業務、γ 業務担当 SG はシステム基盤を担当する方式業務とオフショア要員の管理を兼務していた。B システム開発プロジェクトは規模が小さいため、1 人の開

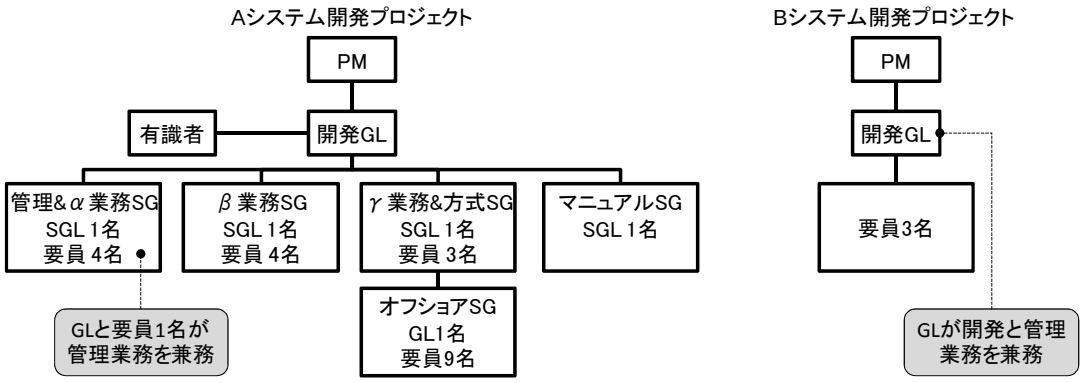

図 1. 報告対象期間のプロジェクト体制と管理業務担当要員

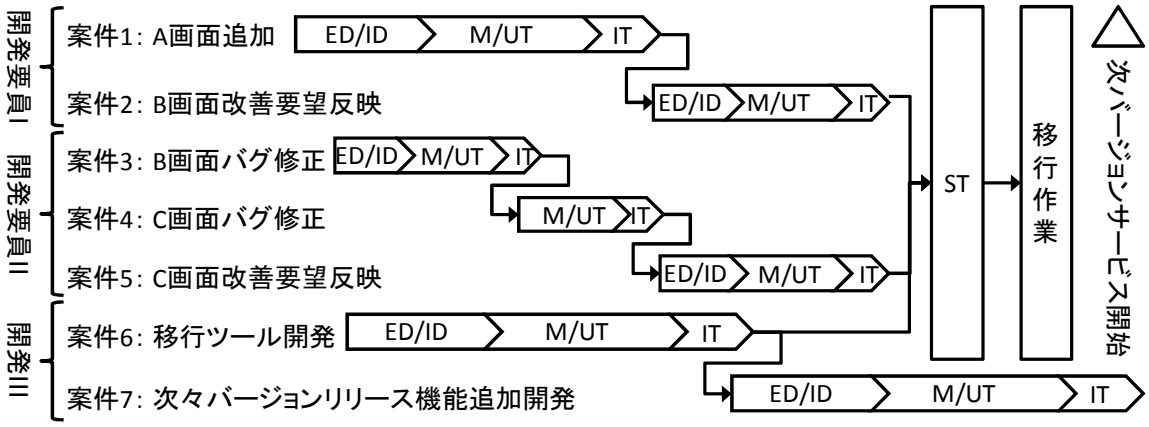

図 2. 報告対象プロジェクトにおける開発プロセスのイメージ

発 GL が全要員を管理した。開発 GL が全体の品質を管理すると共に、管理業務も兼務した。

メトリクスの収集・利用を含む管理業務は、いずれのプロジェクトでも開発業務と兼務であり、大きなコストを確保できる状態ではなかった。管理業務としては、開発プロセスと進捗管理・品質管理の手順・ルール整備、進捗管理・変更管理・構成管理ツールの準備と使用ルール整備、ルール遵守を担保する監視活動、各バージョンの開発計画策定、工程完了やリリースを伺う上位層への報告を継続的に実施した。メトリクスの収集と利用は、それら業務の一環として実施した。

3.2 開発プロセスと Trac の運用方法

図 2 に開発プロセスのイメージを示す。画面追加、バグ修正、改善要望反映など、責任分担と完了確認が明確にできる単位を「案件」とし、案件単位で担当者へ作業をアサインした。案件毎に外部設計/内部設計（ED/ID）、製造/単体テスト（M/UT）、結合テスト（IT）を実施し、複数案件をまとめたリリース単位で総合テスト（ST）を実施した。案件毎に 1 枚、ED/ID、M/UT、IT で各 1 枚ずつ（各案件について 4 枚）Trac チケットを発行し、開発 GL、サブグループリーダ（SGL）、各要員間でチケットを取り回して進捗と品質を管理した。ST は計画したシナリオに沿ってテストを実施し、シナリオ毎にチケットを発行して管理した。

図 3 にソースコードのライブラリ管理のイメージを示す。設計書とソースコードは別のディレクトリへコミットすることとし、ソースコードについてはプロジェクトを通して安定ラインとメインラインを維持保守した。各バージョンの開発開始時に、メインラインから分岐させた開発ラインを作成し、案件単位で M/UT の修正を開発ラインへコミットした。完了した M/UT については、案件単位で修正を IT コードと共にメインラインへマージした。IT 完了後に案件完了判定を行い、その確認をもって案件完了とした。リリースに先立ち、対象の全案件を安定ラインへマージし、ST を行った後、本番環境へデプロイした。B システム開発では、開発や管理のコストを捻出するため、デプロイ作業を自動化して ST 準備やリリースのコストを低減した。

各案件では、チケット駆動開発 [1] と同様の要領でチケットを取り回し、表 2 の作業とレビューを各自席で実施した。リーダは案件を自チームの要員にアサインし、案件に対応する ED/ID、M/UT、IT チケット

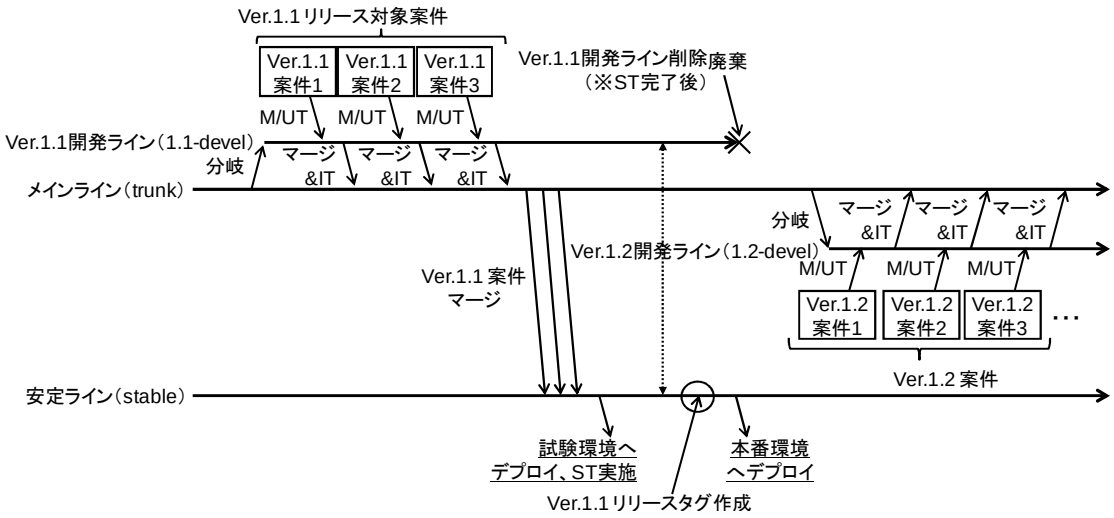

図 3. プロジェクトにおけるソースコードのライブラリ管理のイメージ

表 2. プロジェクトの品質管理方針

工程	作業内容	レビュー対象成果物	レビュー観点
設計（ED/ID）	・仕様検討 ・設計書作成修正	・設計書のチェンジセット	・要件を満たしているか、実現可能か ・影響範囲が考慮されているか
製造（M）	・ソースコード作成・修正	・ソースコードのチェンジセット	・設計に沿っているか ・アーキテクチャや規約に沿っているか ・性能や信頼性は考慮されているか
単体テスト（UT）	・処理のテスト（リクエストレスポンスでのブラックボックステスト） ・C1 網羅のためのホワイトボックステスト	・テスト項目表、テスト証跡※ ・UT コードのチェンジセット ・UT コード走行結果	・修正個所に対して C1 網羅率 100%を充足しているか ・テスト観点を全て考慮しているか ・無駄なテストコードがないか ・影響範囲の回帰テストが全て通るか。
結合テスト（IT）	・画面遷移確認 ・画面間データ連動確認	・テスト項目表、テスト証跡※ ・IT コードのチェンジセット ・IT コード走行結果	・設計書の修正個所を網羅しているか ・基底の試験観点を全て考慮しているか ・無駄なテストコードがないか
案件完了判定	全作業結果確認（主要部分のみ）	・全対象物（主要部分のみ）	・IT までの全観点（ただし、主要部分に絞って確認）
総合テスト（ST）	業務、移行、運用、環境、外部接続、マニュアルの連動、非機能要件の確認	・システム全体	・業務、移行、運用、環境、外部接続が正しく連動するか ・マニュアルが動作と整合しているか ・非機能要件が満たされているか

（※）A システム開発では手動テスト部分は画面キャプチャをテスト証跡とした。B システム開発ではテストを全て自動化し、テストコード中のコメントで試験項目表を代替した。自動化したテストについては、レビュアがテストコードを走行させるため、テスト証跡は不要とした。

を発行する。要員は各工程の作業を実施してレビュー対象成果物のチェンジセット（リビジョン）をチケットに記入し、verified 状態としてリーダにレビューを依頼する。リーダは観点を詳細化・列挙したチェックリスト（MS Excel 形式）に従って自席でレビューを行い、指摘を記入したチェックリストをチケットに添付して要員に返す。要員は指摘へ対応したチェンジセットをチケットに記入して修正確認を依頼する。リーダが全ての指摘対応を確認したら当該工程を完了し、チケットを閉じる。ED/ID、M/UT、IT を順に実施し、最後は案件チケットを GL に verified 状態で渡して案件完了判定を依頼する。

本稿の報告対象期間中（表 1）、A システム開発プロジェクトでは 567 案件に対応し 5 回の本番環境リリースを実施した。B システム開発プロジェクトでは 51 案件に対応し 6 回の本番環境リリースを実施した。

4. メトリクスの収集・蓄積・利用

表 3 に収集した基本メトリクスとチケットへの記入項目を示す。各レビューの際、ルール通りにメトリクスが計測・記入されていることをリーダが監視し、必要に応じて是正した。また、リーダのレビューと別に、管理業務担当者も適宜監視と是正を行った。業務時間中、全ての開発要員は同じ IRC チャネルへ接続しており、指示、チケットの受け渡し連絡、ファイルパスや Trac のカスタムクエリなどの URL は IRC を通じて自席を離れず低コストでやりとりできた。A、B いずれの開発でも、メトリクスの計測・入力、監視・是正活動に要した工数は各案件 1 人時未満であり、小さい工数で収集と蓄積を実施できた。

表 4 に基本メトリクスと共に利用した導出メトリクスを示す。導出したメトリクスの主な用途を下記に示す。いずれの場合も、Trac のカスタムクエリで蓄積したメトリクスを抽出した。抽出に要した工数は 1 時間未満であり、出力した CSV ファイルの MS Excel による集計に数時間必要だった。2 回目以降の集計では、最初に作成した MS Excel ファイルを再利用し、コストを小さく抑えられた。結果を集計した MS Excel は SVN サーバに格納し、プロジェクトの要員全てが閲覧可能な状態にしていた。

- **各バージョンの開発計画策定**

計画策定時には、生産性（KStep/人月）を利用して見積りを行った。リーダ（GL または SGL）が、完了済案件の規模を大、中、小の三段階で分類して生産性を集計する。開発予定案件も大、中、小で見積り、集計した生産性と規模を積算して各案件の見積り工数を算出した。同一箇所を見積り同時修正して衝突を発生させないなどの制約と共に、案件の難度を考慮して要員へ案件をアサインし、要員にも各自の担当案件の見積りを依頼した。リーダは上がってきた要員の見積りを相互比較し、見積りを精査した。大規模な機能追加や仕様が複雑化している部分の修正は、複数案件に分割して見積りと実行管理を行った。

- **スコープ調整・再計画における再見積り**

開発を進める中で進捗の遅れがリカバリできない状態となった場合にスコープ調整、もしくは再計画を実施した。スコープ調整では確保すべき工数と次回リリースに持ち越す案件を、再計画では延伸後のリリース日時を決定するため、再見積りを実施した。その時点での完了済案件の生産性や、1 日当たり稼働時間を用いた。当初 1 日 6 時間として見積っていたが、他業務と兼務の要員など、実質上の稼働が 1 日 3 時間程度のケースもあった。これらを根拠として再見積りを実施し、より正確な計画を再作成した。

- **完了案件に対する週次定量分析**

A システム開発プロジェクトでのみ、完了案件に対する週次定量分析を実施した。過去案件のデータと相互比較してメトリクスが大きく乖離した案件は、案件完了判定とは別に GL が問題の有無を再度ピアレビューで確認した。いくつかの問題が検出され、開発プロセスの改善やレビュー時に用いたチェックリストの改善を行った。B システム開発では体制が小さいため、GL が全案件をピアレビューした。このため週次定量分析の効果が小さいと判断し、実施しなかった。

- **工程完了やリリースを伺う報告**

全案件完了時の ST 開始前や ST 完了後のリリース前に上位マネジメント層への報告を行った。報告に際し、他とメトリクスが大きく乖離した案件について、作業や実施プロセスに問題がなかったかを再度確認した。確認の際は、管理担当者が成果物を直接レビューしたり、各要員にヒアリングしたりして、成果物やプロセス上の問題を検出した。検出した問題は、その重要度や対処工数を考慮して対応を決定した。この作業は、A システムサービス開始前（2010/4）のみ管理担当者がフルタイムで実施した。他の期間では、各バージョンについて 3 人日未満で報告書準備と報告を実施できた。

5. 考察

利用したメトリクスは全て基礎的なものであるが、工数、LOC、開発期間は有効利用できた。基礎的なメトリクスは計測と解釈が容易で、収集や要員の学習コストも小さい。また、それらの多くが他のメトリクスと組み合わせて分析できる。例えば、規模を開発期間で除算した「1 日当たり開発量」は見積りに利用できる。または、案件やモジュール毎に生産性を集計し、相対値が悪いものの抽出にも利用できる。メトリクス導入の際は、収集・学習コストが小さいこと、組み合わせ分析の可否が有効性判断の観点のひとつになる。

表 3. 収集した基本メトリクスとチケットへの記入項目

項目名	記入チケット	記入担当者	記入内容
作業工数(人時)	全チケット	要員	各チケットで実施する作業、およびレビュー指摘事項に対する対応に要した工数を人時で記入。
レビュー工数(人時)	全チケット	リーダ(GL か SGL)	各チケットで実施した作業のレビュー、およびレビュー指摘修正の確認に要した工数を人時で記入。
レビュー頁数(頁)	ED/ID	リーダ(GL か SGL)	レビューした設計書頁数。レビュー対象が MS Excel 形式の場合、レビューしたシート数を記入。
指摘件数(件)	ED/ID	リーダ(GL か SGL)	レビュー指摘件数を記入。レビュー結果は MS Excel 形式のレビュー結果票で 1 行に指摘 1 件を記入する。レビュー結果票の行数(=指摘数)を記入。
開発規模(Step)	M/UT	要員	M/UT 開始前～完了までの追加・変更行数の合計。削除行数は数えない。2.2 で説明したツールでリビジョン間差分の LOC を計測して記入。
テスト項目数(件) または テスト規模(Step)	M/UT、 IT、 ST	要員	テストの量。A システム開発プロジェクトではテスト項目数、B システム開発プロジェクトではテストコード(UT は JUnit、IT と ST は Selenium)の追加変更行数を記入。
バグ数(件)	IT、 ST	要員	IT や ST で検出されたバグの数を記入。 UT バグは原則として取り切ってから M/UT を完了する想定のため、計測しない。製造と UT コード記述を並行実施する場合も多いため、計測にかかるコストも大きすぎる。
すり抜けバグ数(件)	IT、 ST	要員	IT や ST で検出されたものの、完了済の前工程の観点で検出すべきだったバグの数を記入。要員が判断に迷ったバグは、レビュアと管理業務担当者が判断する。
開発期間(日)	案件	要員	案件開始～完了までの日数。開始日と終了日から算出。
案件完了判定指摘件数(件)	案件	GL	ED/ID～IT 完了後に実施する案件完了判定での指摘件数を記入。

表 4. 利用した導出メトリクスとその用途

メトリクス名	評価対象	導出方法※	主な用途
生産性 (KStep/人月)	案件	開発規模 ÷案件に要した全工数	計画時の見積り、および、スコープ調整や再計画で見積り根拠として利用。
1 日当たり稼働時間	案件	案件に要した全工数 ÷開発期間	要員にかかる負荷の調査、稼働時間が低い要員や案件の検出と問題改善。
レビュー密度(分/頁)	ED/ID	ED/ID レビュー工数 ÷レビュー頁数	レビューが不十分な可能性がある案件の検出。仕様が複雑なため、レビューが非常に長い要注意案件の抽出
エラー密度(件/百頁)	ED/ID	ED/ID 指摘件数 ÷レビュー頁数	レビュー不十分でエラーが少なすぎる、またはエラーが多すぎる案件の検出
テスト密度(件/KStep) または テストコード比率(%)	UT、 IT、 ST	テスト項目数÷開発規模 または テスト規模÷開発規模	テストが不十分な可能性がある案件の検出。仕様が複雑なため、テストが非常に多い要注意案件の検出。
バグ密度(件/KStep)	IT、 ST	バグ数 ÷開発規模	テスト不足でバグが少なすぎる、またはバグが多すぎる問題案件の検出。
すり抜けバグ密度 (件/KStep)	IT、 ST	すり抜けバグ数 ÷開発規模	前工程のテストの不備とレビュアのチェック漏れ検出。再発防止に繋げる。

※Step→KStep、時間→分、頁数→百頁など、基本測定量との単位差は適宜補正するとして割愛する。

一方、品質のメトリクスには課題がある。本稿の事例では案件単位で IT と ST のバグ数を計測したが、多くの案件ではそれら工程でバグは検出されなかった。このため、バグが検出された否かのみに着目すればよく、バグ密度を算出した意義は薄かった。案件単位に算出するのでなく、機能やモジュールを単位とし、ある期間に検出されたバグ数からバグ密度を算出すれば、当該部分の品質や、当該部分担当者・担当チームのスキル優劣を客観的に判断する参考情報などに使える可能性があると考える。

テスト量やテスト密度は、A システム開発プロジェクトのように体制が大きく、GL が全案件をピアレビューできない状況ではある程度有効だった。テスト密度が他から乖離している案件について原因を調査した結果、メトリクス計測の誤りなどプロセス上の問題を検出できた場合があった。一方、B システムのように GL がテストコードも含めて全案件をピアレビューする体制では、テスト密度は有効とは言い難い。GL はテスト密度を参照せずともレビューからより詳細な状況を把握しているためである。計測のためのコストを考慮する必要はあるが、C0/C1 網羅率 [14]や、コードクローン分析に基づくメトリクス[17]など、ピアレビューでは検出できない課題を、機械的に走査して検出するものが有効であると考えられる。

6. おわりに

本稿では、軽量開発プロセスにおいてメトリクスの収集・蓄積・利用に Trac を用いた事例を紹介した。実際の中小規模システム開発プロジェクトで紹介したツールとルールをのべ 29 ヶ月間運用し、小さいコストで信頼性の高いメトリクスを収集できた。特に、工数、規模、開発期間についてはメトリクスを有効利用できた。今後の課題として、特にテストコードも含めてピアレビューする中小規模プロジェクトにおいては、品質に関するメトリクスを工夫する必要があると考えられる。例えば、Jenkins [10] などの継続的インテグレーションツールと連動し、広範囲なソースコード分析やリポジトリマイニングを夜間に行うなど、人間に不可能な観点・方法で成果物をチェックするツールを導入し、低コストで運用できる方法を検討したい。

7. 参考文献

[1] 小川明彦, 阪井誠, “チケット駆動開発, ” 翔泳社, 2012.

[2] Akiyama, F., “An Example of Software System Debugging,” Info. Processing, 71: 353-379, 1971.

[3] Albrecht, A. J., “Measuring Application Development Productivity,” In *Proc. of the IBM Applications Development Joint SHARE/GUIDE Symposium*, Monterrey, California, USA, 83-92, 1979.

[4] Apache Software Fundation, The, “Apache Subversion,” 2000-2013; http://subversion.apache.org/

[5] Boehm, B. W., “Software Engineering Economics,” Prentice-Hall, New York, 1981.

[6] Cunningham, W., “WikiWikiWeb,” 1995; http://c2.com/cgi/wiki

[7] EASE Project, “EPM,” 2003-2008; http://www.empirical.jp/research/j_download.html

[8] Edgewall Software, “Trac Open Source Project,” 2003-2012; http://trac.edgewall.org/

[9] 阿萬裕久, 野中誠, 水野修, “ソフトウェアメトリクスとデータ分析の基礎,” コンピュータソフトウェア 28(3): 12-28, 2011.

[10] Jenkins CI, “Jenkins,” 2010-2013; http://jenkins-ci.org/

[11] Martin, J., “Programming Real-ime Computer Systems,” Englewood Cliffs, NJ: Prentice Hall, 1965.

[12] McCabe T., “A Software Complexity Measure,” IEEE Trans. on Soft. Eng., SE-2(4): 308-320, 1976.

[13] Microsoft Corporation, “Microsoft Office Excel,” 1985-2013; http://office.microsoft.com/excel/

[14] Miller, J. C., Maloney, C. J. “Systematic Mistake Analysis of Digital Computer Programs,” Communications of the ACM 6 (2): 58–63, 1963.

[15] Ostdiek, M. A., and Estes, R. T., “Cost Schedule Control System Criteria: An Analysis of Managerial Utility,” MS thesis, School of Systems and Logistics, Air Force Institute of Technology, 1975.

[16] Project Amateras, “StepCounter,” GitHub, 2002-2013; https://github.com/takezoe/stepcounter

[17] 中江大海,神谷年洋,門田暁人,加藤祐史,佐藤慎一,井上克郎, “レガシーソフトウェアを対象とするクローンコードの定量的分析,” 電子情報通信学会技術研究報告 SS200-49, 100(570): 57-64, 2001.

[18] Walton, L., “Eclipse Metrics Plugin,” SourceForge.net; http://eclipse-metrics.sourceforge.net/

[19] Weinberg, G. M., and Gressett, G. L., “An Experiment in Automatic Verification of Programs,” Communications of the ACM, 6(10): 610-613, 1963.