ソフトウェア開発状況の把握を目的とした 変化点検出を用いたソフトウェアメトリクスの時系列データ分析

久木田　雄亮
和歌山大学 システム工学研究科
s161018@sys.wakayama-u.ac.jp

柏　祐太郎
和歌山大学 システム工学研究科
kashiwa.yutaro@g.wakayama-u.jp

大平　雅雄
和歌山大学 システム工学部
masao@sys.wakayama-u.ac.jp

要旨

大規模ソフトウェア開発では，進捗および品質の状況を随時把握する必要がある．開発状況を把握するためのモニタリングツールは多数存在しているが，異変が起こっているかの判断はプロジェクト管理者の勘と経験による部分が大きい．そこで本研究では，変化点検出アルゴリズムを用いて，リアルタイムに計測したソフトウェアメトリクスの値から開発状況の異変を機械的に検出するための手法を提案する．*Eclipse Platform* プロジェクトを対象としてケーススタディを行った結果，変化点前後で開発の活動内容が別の活動内容にシフトしていく様子を見てとることができた．また，トピック分析を併用した場合，プロジェクト管理者が変化の内容を理解しながら開発状況を把握できることが分かった．

1. はじめに

ソフトウェア製品の大規模化・複雑化に伴って，高品質なソフトウェアを安定して開発することが困難になってきている．ソフトウェアの品質を維持しつつ開発を行うためには，開発・品質状況を随時把握する必要がある．そのため，ソフトウェアメトリクスのリアルタイム収集および定量的な進捗・品質計測を支援するモニタリングツールが多数提案されてきた [9][3][5]．ソフトウェアメトリクスとは，ソフトウェア開発データの様々な特性から計測可能な定量的尺度 [11] のことであり，ソフトウェア開発における進捗・品質管理にとって必要不可欠な存在である．

既存のモニタリングツールを用いた品質管理では，プロジェクトが計画通りに進んでいるか否か，品質が一定の水準に保たれるかどうかの判断は，未だにプロジェクト管理者の勘と経験による部分が大きいとされる．そのため，開発の遅延やソフトウェアの障害につながる可能性のある異変をいち早く検知するための支援が求められている．

例えば，コード行数（LOC） を計測してソースコードの規模推移をリアルタイムにモニタリングする状況を考える．実装開始直後は通常，LOC の変化（増加量）は日々比較的大きな値をとり，出荷に向けて LOC の変化は小さな値をとる．テスト工程での欠陥修正による LOC の変化は，実装工程での LOC の変化に比べ小さなものであることが多いため，テスト工程でのコード修正にまつわるなんらかの異常を LOC の変化から見て取ることは困難であると考えられる．すなわち，ソースコードの規模推移がすべて見渡せる環境があるが故に，分析者（プロジェクト管理者）は変化の大きさを相対的に認知してしまい，工程や期間毎に意味の異なる変化を見落としてしまう可能性が高い．プロジェクトの異変を，相対的な変化ではなく工程毎のコンテキストに沿った意味的な変化として機械的に検知することができれば，早くに対処することができる可能性がある．

本研究の目的は，変化点検出アルゴリズム [10] を用いて，プロジェクトの進捗・品質状況の悪化などといった

変化を，リアルタイムに計測したソフトウェアメトリクスの値から早い段階で捉えるための手法の構築することである．本論文では，オープンソースプロジェクトから取得したデータに対して変化点検出を行うとともに，トピック分析 [2] を用いて変化の要因を理解することが可能かどうかを確かめる．

2. モニタリングツールによるソフトウェア管理

大規模かつ複雑な作業の組み合わせを経て開発される近年のソフトウェアを，定められた納期に定められた品質で開発するのは容易ではない．ソフトウェアをより安定して開発するためには，開発者の能力や開発対象によって品質が左右されないようにリソース割当計画や開発プロセスを策定し，進捗・品質を管理することが必要となる．プロジェクトの状況を把握し問題点を発見・対処するために，プロジェクトの状況を可視化するモニタリングツールが多数存在している．本章では，既存のモニタリングツールをいくつか紹介し，本研究の立場の違いを説明する．

2.1. プロジェクト管理を目的としたモニタリングツール

プロジェクトモニタリングツールのひとつとして，大平らの EPM(Empirical Projet Monitor) [9][4] がある（図 1）．EPM は，リアルタイムのプロジェクトモニタリングを目的としたツールであり，より安定したソフトウェア開発のために開発状況を随時把握することを支援する．EPM は，バージョン管理システム，メーリングリスト管理システム，障害管理システムのリポジトリに保存された履歴データを収集して利用する．収集したメトリクスデータから，ソースコードの規模推移やチェックインとチェックアウトの関連など，グラフィカルにプロジェクトの状況を表現することができる．

Teamscale[3] は，リアルタイムにソフトウェアの品質を管理することを目的として開発されたツールである（図 2）．ソフトウェアの品質状況を示す様々な指標を可視化する．バージョン管理システムに開発者が差分ソースコードをコミット（登録）した直後に，分析結果をフィードバックすることができる．開発者が普段使う開発ツール（Eclipse, や Visual Studio など）に組み込むことができる．

Cisco の Software Quality Dashboard(SWQD)[5] は，顧客に関するメトリクスを取得しモニタリングする（図 3）．例えば，取得しているメトリクスの 1 つに CFD (Customer-Found Defect) があり，顧客が発見した欠陥の数を取得して表示していることが見て取れる．

Hackystat[1] は，ソフトウェアのプロダクトデータとプロセスデータを収集・分析し，結果を可視化するツールである（図 4）．Hackystat は，プロジェクトの開発者や管理者がプロジェクトの開発状況を把握することに用いられる以外に，教育分野でも用いられている．

2.2 既存のモニタリングツールの問題点

既存のモニタリングツールは基本的に，取得・計測した値のみを可視化する．現在プロジェクトが順調に進んでいるのか停滞しているのかといった判断は，管理者や開発者の経験や勘に頼る部分が大きい．特に，問題発見は管理者および開発者の判断で行われるため，プロジェクトの遅延につながる問題や，後に大きな障害となりそうな可能性のある変更（code change）が混入したかどうか，などを時系列のグラフや値から読み取ることは難しい．プロジェクト遅延や品質低下を招く原因となりうる変化を機械的にできるだけ早く見つけることができれば，手戻りコストを減らすことができると言える．

本研究では，プロジェクトの進捗・品質管理において今後大きな問題につながる変化を，開発者や管理者の勘や経験に頼らず，自動的にリアルタイムで抽出する手法を提案する．

3. 変化点検出を用いたソフトウェアメトリクスの時系列データ分析手法

本章では，変化点検出を用いたソフトウェアメトリクスの時系列データ分析手法を提案する．提案手法は，大きく分けて，変化点検出アルゴリズム [10] をソフトウェアメトリクスに適用するための処理と，検出した変化点の前後でどのような話題が議論されていたかを分析するためのトピック分析 [2] からなる．

3.1 変化点検出

3.1.1 データ収集とメトリクス集計

版管理システム，不具合管理システム，メーリングリスト管理システムなどのリポジトリからメトリクスデー

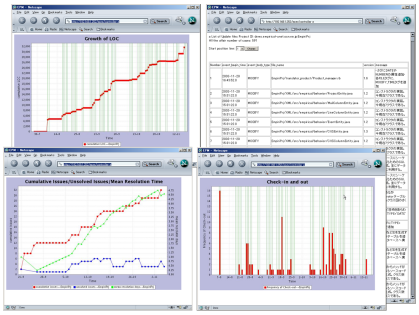


図 1. EPM[9] の使用例

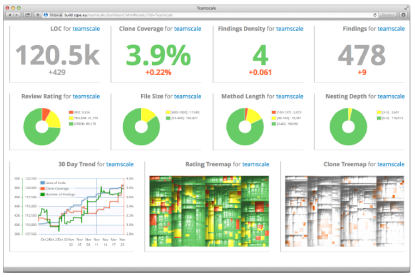


図 2. Teamscale[3] の使用例

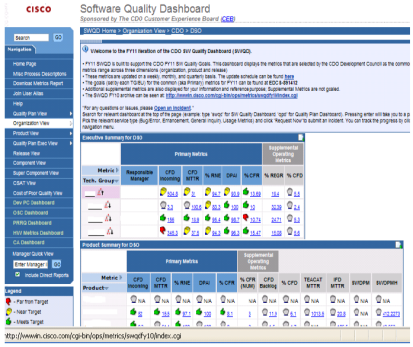


図 3. SWQD[5] の使用例

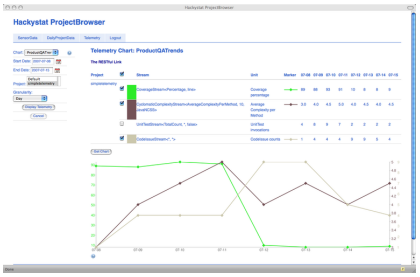


図 4. Hackystat[1] の使用例

タを時系列データとして取得する．リポジトリの種類によって，時系列データの整形方法は異なる．

版管理システム：コミットログに従ってリビジョン毎にソースコードからメトリクスを抽出する．さらに，各リビジョンのタイムスタンプに従って，データを 1 日単位にまとめて集計する．例えば，1 日に 2 回のコミットがあり，それぞれ異なるファイルにコードを追加している場合は，二つのファイルの追加行数の合計がその日のソースコード増加行数として集計される．

不具合管理システム：不具合管理システムに登録された不具合票には，報告日時，割当日時，修正完了日時などの異なるタイムスタンプが記録されているため，取得するメトリクス毎にタイムスタンプを区別して集計する．例えば，不具合報告数は，すべての不具合報告の報告日時を計測し，1 日毎に集計して算出する．

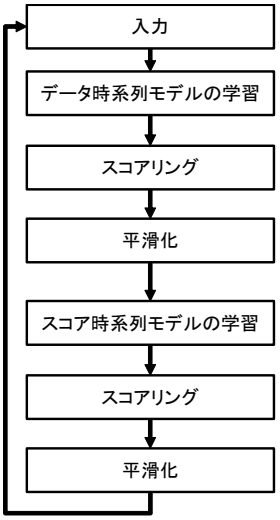
メーリングリスト管理システム:メーリングリストデータからは送信（受信）日時を用いて，メール送信数や送信者数を 1 日毎に集計する．

なお，本稿のケーススタディでは時間の最小単位を 1 日としたが，上述のデータをまとめることで 1 週間単位や 1ヶ月単位での時系列データを作成することもできる．

3.1.2 変化点検出

集計したメトリクスの時系列データに対して変化点検出アルゴリズム [10] を適用する．変化点検出とは，データマイニングの分野で用いられている異常検知手法の 1 つである．変化点とは，時系列データのモデル（トレンド）の変わり目を指す．一時的なデータの大きな変動（外れ値等）は変化点とはみなさず，あるデータの大きな変動がある程度継続して続く場合，本質的なデータの変動を変化点としてみなす．

本研究では，ChangeFinder[6] を用いた変化点検出を行う．ChangeFinder とは時系列モデルの 2 段階学習に基づいている．時系列モデルとは，過去の時系列データをもとにして将来の時系列データを予測するために定式化されたものである．時系列モデルには様々な種類のモデルが存在しており，自己回帰モデル（Auto Regressive Model），移動平均モデル（Moving Average Model），自己回帰移動平均モデル（Auto Regressive Moving Average Model）などがある．ChangeFinder では，時系列データに対して自己回帰モデル（AR モデル）を用いてモデル化し，SDAR アルゴリズムを適用し，学習する．AR モデルは，時系列データの定常性を仮定した下でし



$$x_t \quad (1)$$

$$P_{t-1}(x) \quad (2)$$

$$Score(x_t) = -\ln P_{t-1}(x_t) \quad (3)$$

$$y_t = \frac{1}{T}\sum_{i=t-T+1}^{t}(-\ln P_{t-1}(x_i)) \quad (4)$$

$$Q_{t-1}(y) \quad (5)$$

$$-\ln Q_{t-1}(y_t) \quad (6)$$

$$Score(t) = \frac{1}{T}\sum_{i=t-T+1}^{t}(-\ln Q_{t-1}(y_i)) \quad (7)$$

図 5. 時系列モデルの 2 段階学習

か扱うことができないが，SDAR アルゴリズムを適用することにより 非定常な時系列データに対応することができる．SDAR アルゴリズムとは AR モデルの忘却学習アルゴリズムのことである．忘却学習とは，過去のデータが現時点からさかのぼればさかのぼるほどモデルの学習に与える影響が減少する学習のことである．SDAR アルゴリズムでは忘却パラメータを設定することにより忘却型学習を行っている．この忘却型学習によって，AR モデルを用いて非定常な時系列データを学習することができる．

図 5 に，時系列モデルの 2 段階学習のアルゴリズムを示す．式 (1) で時点 t におけるデータ x_t を入力として受ける．式 (2) の $P_{t-1}(x)$ は $x^{t-1} = x_1, ..., x_{t-1}$ から SDAR を用いて学習 (第 1 段階の学習) をした確率密度関数を示している．式 (3) の $Score(x_t)$ は対数損失関数である．つまり，入力されたデータ x_t が P_{t-1} からどの程度外れているかを表している．式 (4) では，式 (3) で求めた $Score(x_t)$ の移動平均を求めることで平滑化し，y_t とする．平滑化することにより，一時的なデータの変動に対して反応した外れ値を除去する．T は移動平均のウインドウサイズで，与えられた正数である．式 (5) の Q_{t-1} は新しく得られた時系列データ $\{y_t : t = 1, 2, ...\}$ の確率密度関数で，式 (2) と同様に $y^{t-1} = y_1, ..., y_{t-1}$ から SDAR を用いて学習 (第 2 段階の学習) をして得る．式 (6) は式 (3) と同様に，y_t が Q_{t-1} からどの程度外れているかを表している．式 (7) で得られる $Score(t)$ が変化点スコアである．第 1 段階の学習の時点では，外れ値と変化点が両方がスコアとして高い値が出力されるが，式 (4) で平滑化を行うことで，外れ値を除去できる．そして，第 2 段階の学習によって本質的な変動のみが $Score(t)$(変化点スコア) の値が高くなる．

3.2 トピック分析

変化点検出によって検出された変化点の前後でどのような議論がなされていたのかを確認できれば，変化点の意味を理解するために役立つと思われる．提案手法では，変化点として観察される事象が発生した要因を特定するための方法としてトピック分析を用いる．

3.2.1 トピック分析

提案手法では，後述する LDA に基づくトピック分析を，メーリングリストや不具合報告のコメントデータに適用する．トピック分析を用いることで，それぞれのコミュニケーションメディアで行われた議論の要点をトピック（話題）の分布として捉えることができる．提案手法では，変化点が検出された前後のトピックを，設定した期間（1 日や 1 週間など）毎に集計する．トピックの移り変わりを見ることにより，変化点の前後でプロジェクトに何が起きていたのか理解することを支援する．

3.2.2 潜在的ディリクレ配分法（LDA）

潜在的ディリクレ配分法は，文書の生成を確率的にモデル化したトピックモデルである．トピックモデルとは，文書内に潜在しているトピック（話題）を推定するモデルである．本手法でのトピック分析において潜在的ディリクレ配分法（Latent Dirichlet Allocation）[2] を用いる．ひとつの文書には複数のトピックが存在し，それぞれのトピックがある確率によって文書内に生起するという考えのもと，それぞれのトピックの確率分布を求める手法である．文書に含まれる可能性の高いトピックを抽出することで，文書内に潜在しているトピックを推定できる．

近年では，LDA はリポジトリマイニングの分野で使われることが多い [7][8]．主にリポジトリのデータに対する開発者の理解を手助けする目的などで用いられている．

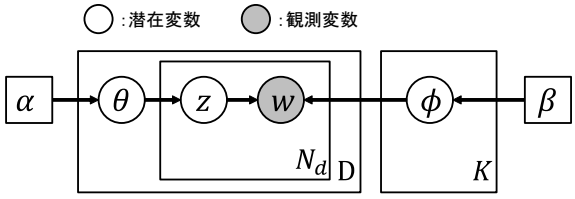


図 6. LDA のグラフィックモデル

図 6 に，潜在的ディリクレ配分法のグラフィックモデルを示す．文書毎にトピック分布 θ を持ち，文書上の各単語について，θ によってトピック z が選ばれ，トピック z の単語分布 ϕ に従って，単語 w が生成される．K はトピック数，D は文書数，N_d は文書 d 上の単語の出現回数を表している．また，α は θ が従うディリクレ分布のパラメータで，β は ϕ が従うディリクレ分布のパラメータである．

4. ケーススタディ

提案手法の有用性を確認するために，本論文ではオープンソースプロジェクトの 1 つである Eclipse Platform プロジェクトを対象としたケーススタディを行う．

4.1 分析対象データ

分析対象とするデータは，Eclipse Platform プロジェクトのソースコード，ソースコードの変更履歴，不具合報告，メーリングリストである．対象とする期間は 2003 年 7 月から 2008 年 7 月までの 5 年間とする．

4.2 分析方法

本ケーススタディにおける分析の手順を説明する．

1. **メトリクスの抽出**：まず，Eclipse Platform プロジェクトが利用している版管理システムから，ソースコードおよびソースコードの変更履歴（コミットログ）をすべて取得する．次に，コミットログに従い，リビジョン毎にソースコードを取り出す．その後，ソースコード解析ツール Understand を用いてコードメトリクスを抽出する．本ケーススタディでは，コード行数（LOC）およびサイクロマティック数をリビジョン毎に抽出した．また同様に，不具合管理システムから不具合報告に関するメトリクスを抽出する．本ケーススタディでは，不具合報告数，不具合解決数，修正待ち不具合数を 1 日毎に抽出した．
2. **コードメトリクスの時系列データの整形**：コードメトリクスデータを時系列データにまとめる．各リビジョンのタイムスタンプに従い，メトリクスデータを 1 日単位の時系列データに整形する．不具合メトリクスは，抽出時点で時系列データとなっているため，本ステップでは何も行わない．
3. **時系列データを用いた変化点検出**：コードメトリクスおよび不具合メトリクスの時系列データを入力として，変化点検出を行う．ChangeFinder を使う際に設定するパラメータには，AR モデルの次数，平滑化のウィンドウサイズ，忘却パラメータの 3 つがある．本ケーススタディでは，それぞれ 4，7，0.01 として変化点検出を行った．
4. **トピック分析**：開発者のメーリングリストおよび不具合報告に含まれるのコメント（テキストデータ）に対して，1 週間毎に LDA を適用する．設定するトピック数が多すぎると分析が困難になるため，本ケーススタディでは，トピック数を 10 個に設定する．
5. **変化点の要因特定**：高い変化点スコアが現れた時点の前後 1 週間のトピック分析の結果を抽出し，各トピックに含まれる単語を確認する．変化点前後におけるプロジェクト内での議論内容を確認できるため，大きな変化点が計測された要因が特定しやすくなる．本ケーススタディでは，変化点スコア 5 以上の変化点の前後 1 週間のトピックを抽出することとした．変化点スコアを 5 に設定した理由は，5 以上の変化点スコア計測されることは本データセットでは稀であったことから，コード行数およびサイクロマティック数に大きな変化をもたらすイベントが生じた可能性が高く，それらに関連する議論がなされている筈と考えたためである．

4.3 分析結果

4.3.1 ソースコードメトリクスでの変化点検出結果

コード行数（LOC）およびサイクロマティック数に対して変化点検出を行った結果を次の図 7，図 8 に示す．横軸は 1 日単位の時系列を表し，左縦軸はそれぞれコー

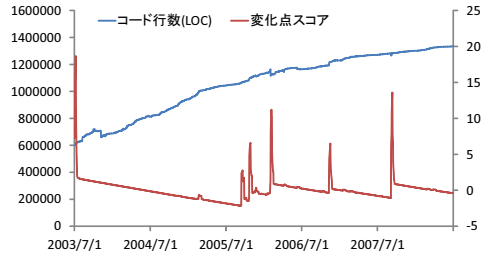


図 7. コード行数に対する変化点検出の結果

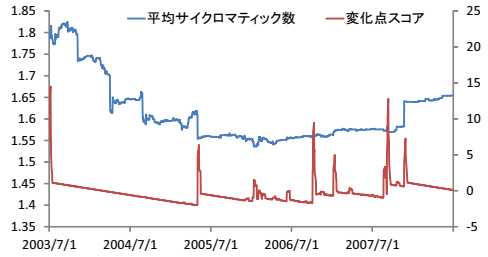


図 8. サイクロマティック数に対する変化点検出の結果

ド行数，サイクロマティック数の平均値を表し，右縦軸は変化点スコアを表している．

コード行数では，変化点スコアが 5 以上であったものは，2005 年 10 月，2006 年 11 月，2006 年 2 月，2007 年 9 月の計 4 箇所であった．コード行数が急増あるいは急減したことを示している．

サイクロマティック数では，2005 年 5 月，2006 年 10 月，2007 年 9 月，2007 年 11 月の計 4 箇所で 5 以上の変化点スコアが計測されており，サイクロマチック数の大きな増加あるいは減少があったことが見て取れる．

4.3.2　不具合メトリクスでの変化点検出結果

不具合管理システムから抽出したメトリクスである不具合報告数，不具合解決数，修正待ち不具合数に対して変化点検出を行った結果を図 9，図 10，図 11 に示す．横軸は 1 日単位の時系列を表し，左縦軸はそれぞれ累計不具合報告数，累計不具合解決数，修正待ち不具合数を表し，右縦軸は変化点スコアを表している．

不具合報告数に関しては，5 以上の高い変化点スコアは見ることができなかった．

不具合解決数と修正待ち不具合数では変化点スコアが

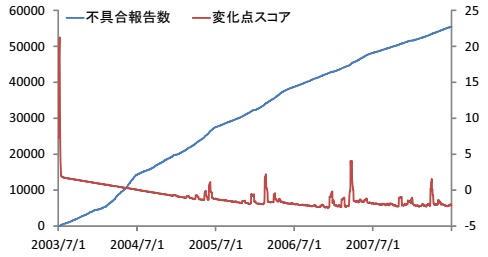


図 9. 不具合報告数に対する変化点検出の結果

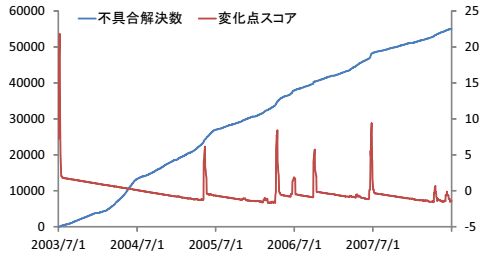


図 10. 不具合解決数に対する変化点検出の結果

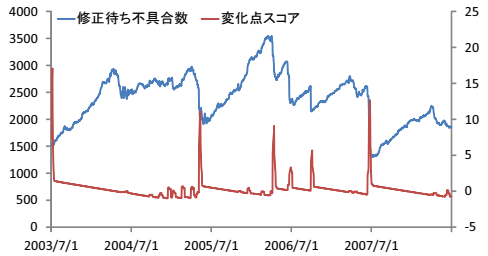


図 11. 修正待ち不具合数に対する変化点検出の結果

5 以上の時点は共通しており，2005 年 5 月，2006 年 4 月，2006 年 10 月，2007 年 6 月 2 の計 4 箇所でそれぞれ計測された．共通する 4 時点は，不具合解決数の急増と修正待ち不具合数の急減に対応しており，妥当な結果といえる．

4.3.3　トピック分析結果

設定した期間のメーリングリストと不具合報告のコメントに対してトピック分析を行った結果，それぞれ表 1，表 2 のように 10 種類のトピックが分類された．各トピックに含まれる単語からトピック（議論の内容）を推測するために利用できる．

表 1. メールのトピック分類

トピック番号	トピックに含まれる単語
M1	table tree swt event public column object viewer item tom selection text string int jface return tableviewer void method
M2	eclipse wrote problem code don application cvs swt java work ve windows time user project make find rcp bug
M3	view menu eclipse ui action org extension editor id add perspective views plugin class actions page point rcp shellbtn
M4	eclipse plugin file project jar build plugins xml plug ant org application wrote java run rcp directory class files
M5	java org eclipse internal core ui main osgi run framework runtime lang swt workbench invoke reflect sun launcher method
M6	backslash nbsp font br face size courier serif sans div wb ah wc ep wd mj ad yt ai
M7	eclipse org http news wrote message www bugs html swt bug platform cgi index browser show id https wiki
M8	editor file class method code wrote null line thread public string eclipse return object text resource plugin run problem
M9	swt shell display composite org eclipse import public text image button widgets void layout griddata label parent settext int
M10	backslash eclipse org lib jar java plugins ui usr dll core bundle message swt system subentry missing required home

表 2. 不具合コメントのトピック分類

トピック番号	トピックに含まれる単語
B1	lib eclipse usr jar plugins xp java home dll org system opt rw aab jre gtk rwxp javac bin
B2	eclipse org ui core plugins file jar update platform xml http java plugin jdt id plug osgi version feature
B3	swt shell display line int public string event void import class org eclipse null table os text item return
B4	file project cvs workspace eclipse build files problem error run ant log user set case projects dialog view laun
B5	eclipse swt problem gtk java version windows linux running build browser time vm run comment work code system os
B6	java eclipse org internal core ui swt run widgets main lang workbench jface runtime method display invoke object reflect
B7	bug duplicate marked fixed fix verified build problem head released bugs patch comment verify issue closing marking rc report
B8	api code comment change don method make case class work content add support point ui implementation part extension type
B9	created details attachment patch test screenshot fix tests file log screen attached updated problem added applied error shot image
B10	view editor text menu dialog window open problem key button selection tab windows perspective ctrl show don click set

5. 考察

本稿では紙面の都合上，サイクロマティック数の推移（図 8）に対して大きな増減を捉えた 4 時点の変化点の内 2 時点（2005 年 5 月と 2007 年 11 月），および，不具合解決数と修正待ち不具合数の推移（図 10 と図 11）対して大きな増減を捉えた 4 時点の変化点の内 2 時点（2005 年 5 月と 2007 年 6 月）のみに着目する．それぞれの変化点の前後のトピックの変化を確認することにより，大きな変化点の原因となった要因を理解できるかどうか，すなわち，提案手法がプロジェクト内に発生した何らかの変化の原因を理解するために有用かどうかを議論する．

5.1 コードメトリクスに対する変化点検出

サイクロマティック数の減少に対して大きな変化点が計測された 2005 年 5 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 12 および図 13 に示す．同様に，サイクロマティック数の増加に対して大きな変化点が計測された 2007 年 11 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 14 および図 15 に示す．

2005 年 5 月におけるメールトピックでは，トピック M2 が増加している．一方，不具合コメントトピックでは，トピック B7 が増加していることが見て取れる．メールトピック M2 には，“eclipse”, “wrote”, “problem”, “code”, “bug” などの単語が含まれており，不具合に関する議論が行われていると推察できる．また，不具合コメントトピック B7 には，“bug”, “duplicate”, “marked”, “fixed”, “fix” などの単語が含まれており，リリースに向けた活動に関する議論がなされていたものと思われる．これらのことから，サイクロマティック数の大幅な減少に対する理由としては，以下のようなことが考えられる．2005 年 5 月に大規模なリファクタリングが行われた結果，不具合が増加したと考えられる．そして，不具合が増加したため，リリースに向けた修正活動が活発になされていたものと推測される．

一方，2007 年 11 月におけるメールトピックは，トピック M3 が増加している．不具合コメントトピックでは，トピック B5 が減少していることが見て取れる．メールトピック M3 には，“jar”, “view”, “menu”, “ui”, “action” といった単語が含まれており，UI に関する議論が行われていると推察できる．また，不具合コメントトピック B5 には，“problem”, “version”, “windows”, “linux”,

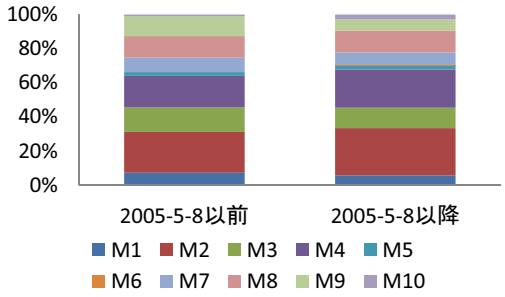


図 12. 2005-5-8 の前後 1 週間のメールトピックの確率分布

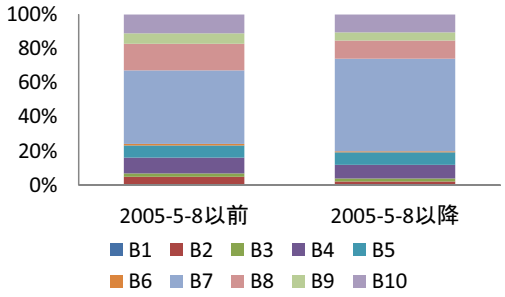


図 13. 2005-5-8 の前後 1 週間の不具合コメントトピックの確率分布

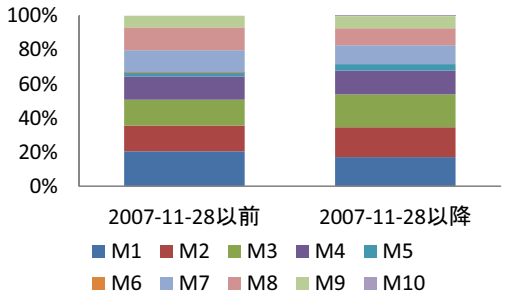


図 14. 2007-11-28 の前後 1 週間のメールトピックの確率分布

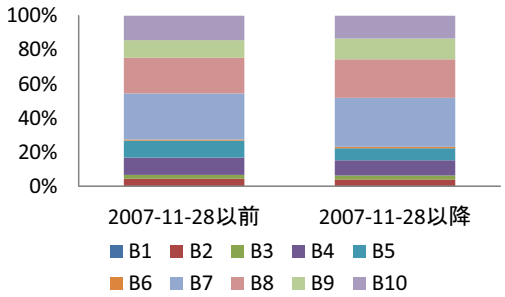


図 15. 2007-11-28 の前後 1 週間の不具合コメントトピックの確率分布

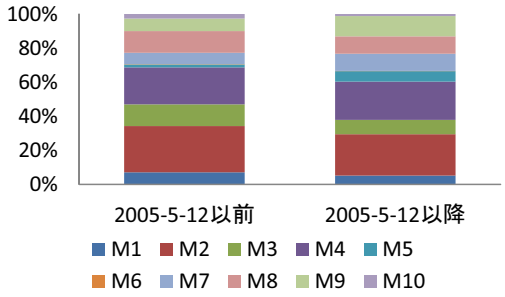


図 16. 2005-5-12 の前後 1 週間のメールトピックの確率分布

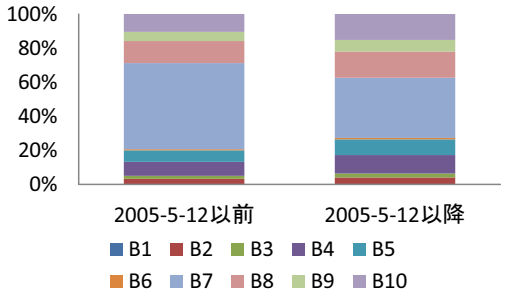


図 17. 2005-5-12 の前後 1 週間の不具合コメントトピックの確率分布

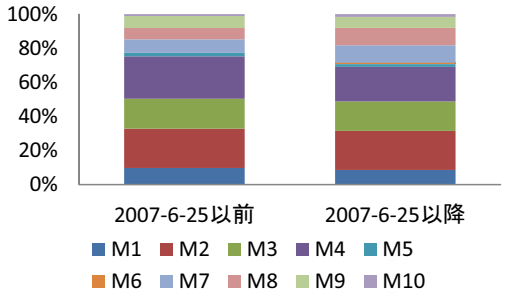


図 18. 2007-6-25 の前後 1 週間のメールトピックの確率分布

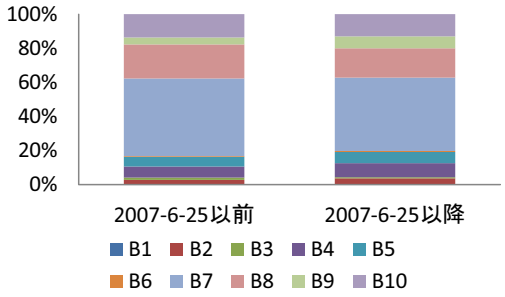


図 19. 2007-6-25 の前後 1 週間の不具合コメントトピックの確率分布

“running” などの単語が含まれており，テスト活動に関する議論がなされていたものと思われる．これらのことから，サイクロマティック数の大幅な増加に対する理由としては，以下のようなことが考えられる．2007 年 11 月に UI に関する大きな修正が必要であったとテスト活動によって判明した思われる．その結果，サイクロマティック数が増加したと推測される．

5.2 不具合メトリクスに対する変化点検出

不具合解決数の増加と修正待ち不具合数の減少に対して大きな変化点が計測された 2005 年 5 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 16 および図 17 に示す．同様に，不具合解決数の増加と修正待ち不具合数の減少に対して大きな変化点が計測された 2007 年 6 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 18 および図 19 に示す．

2005 年 5 月におけるメールトピックでは，トピック M3 が減少している．一方，不具合コメントトピックでは，トピック B7 が減少していることが見て取れる．メールトピック M3 には，“jar”, “view”, “menu”, “ui”, “action” といった単語が含まれており，UI に関する議論が行われていると推察できる．また，トピック B7 には，“bug”, “duplicate”, “marked”, “fixed”, “fix” などの単語が含まれており，リリースに向けた活動に関する議論がされていると思われる．このことから，修正待ち不具合数の大幅な減少，不具合解決数の大幅な増加に対する理由としては，以下のようなことが考えられる．UI に関する不具合を解決する方法が見つかった結果，多数の不具合修正が行われたと推測される．

一方，2007 年 6 月におけるメールトピックでは，トピック M4 が減少している．不具合コメントトピックでは，トピック B7 が減少していることが見て取れる．メールトピック M4 には “jar”, “build”, “plugins”, “plug”, “ant” といった単語が含まれており，プラグインに関する議論が行われていると推察できる．また，不具合コメントトピック B7 には，“bug”, “duplicate”, “marked”, “fixed”, “fix” などの単語が含まれており，リリースに向けた活動に関する議論がされていると思われる．このことから，修正待ち不具合数の大幅な減少，不具合解決数の大幅な増加に対する理由としては，以下のようなことが考えられる．プラグインに関する不具合を解決する方法が見つかった結果，多数の不具合修正が行われたと推測される．

5.3 本論文の制約

本論文では変化点のスコアが高くなっている要因を特定するために LDA を用いて前後 1 週間のトピック分析を行った．トピックモデルの LDA を適用して得られた結果として，不具合コメントとメールで 10 個づつのトピックがある．それぞれのトピックの議論の内容をトピックに含まれる単語から推測していた．そのため，今後は分類されたテキスト（メールや不具合コメント）も含めて確認し，総合的に議論の内容を推測する必要があると考えられる．

本論文で用いた変化点検出には事前に設定するパラメータ，AR モデルの次数，平滑化のウィンドウサイズ，忘却パラメータの 3 つがある．AR モデルの次数については参考文献で使われていた 4 次に設定した [6, pp.57-58]．平滑化ウインドウサイズは小さすぎると移動平均の値の動きが大きすぎるため毎日変化点が検出されることになるので，平滑化ウィンドウサイズについては 1 週間つまり 7 日毎の移動平均を取るために 7 に設定した．忘却パラメータについては，0.01 に設定した．これ以上小さくすると変化点を計測することができなくなるため，変化点検出可能な忘却パラメータのうち最も小さい値を設定した．

6. まとめ

本論文では，プロジェクトにおけるソフトウェア開発状況の異変を機械的に早い段階で検知することを目的としたトピック分析を併用した変化点検出手法を提案した．提案手法は，変化点検出アルゴリズムをソフトウェアメトリクスの時系列データに適用し変化点を検出する．検出された時点のメールや不具合コメントにトピックモデルの LDA を用いてトピック分析を行うことで，変化点が検出された要因を特定する．

提案手法の有用性を確かめるために，オープンソースプロジェクトの Eclipse Platform プロジェクトを対象とするケーススタディを行った．ケーススタディではソースコード，不具合管理システムの 2 つのデータソースから取得できるメトリクス（コード行数，サイクロマティッ

ク数の平均値，不具合報告数，不具合解決数，修正待ち不具合数）の時系列データを用いて変化点検出を行った．その結果，不具合報告数の時系列データ以外で高い変化点スコアの持つ変化点を検出することができた．また，不具合管理システムから取得できる不具合コメントと開発者メーリングリストから取得できるメールに対してトピック分析を行った．そして，変化点が検出された前後1週間の不具合コメントとメールトピックの割合を比較した．

ケーススタディを行った結果，ソースコードから取得したメトリクスを使った変化点検出と不具合管理システムから取得したメトリクスを使った変化点検出において変化点を検出することができた．そして，変化点の前後1週間のメールトピックと不具合コメントトピックの比較をすることによって，変化点の前後で，開発者間で議論がシフトしている様子を確認することができた．したがって，管理者が開発活動の状況を把握することに役立つと言える．

今後は複数のプロジェクトに対して同じ分析手法を適用し，より高い一般性のある結果を得る必要がある．変化点検出に用いたメトリクスについてもソースコードと不具合管理システムから取得したメトリクスだけでなく，別の視点でのメトリクスを対象とすることで，プロジェクトにおける開発状況の異変を別の観点から検知できると考えている．

謝辞

本研究の一部は，文部科学省科学研究補助金（基盤(C): 15K00101）による助成を受けた．

参考文献

[1] Hackystat. https://code.google.com/p/hackystat.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research (JMLR2003)*, Vol. 3, pp. 993–1022, 5 2003.

[3] Lars Heinemann, Benjamin Hummel, and Daniela Steidl. Teamscale: Software quality control in real-time. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE2014)*, pp. 592–595, 7 2014.

[4] Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, and Koji Torii. Empirical project monitor: a tool for mining multiple project data. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR2004)*, pp. 42–46, 5 2004.

[5] Pete Rotella and Sunita Chulani. Implementing quality metrics and goals at the corporate level. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR2011)*, pp. 113–122, 5 2011.

[6] Jun-ichi Takeuchi and Kenji Yamanishi. A unifying framework for detecting outliers and change points from time series. *Transactions on Knowledge and Data Engineering (TKDE2006)*, Vol. 18, No. 4, pp. 482–492, 4 2006.

[7] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR2011)*, pp. 173–182, 5 2011.

[8] Shamima Yeasmin, Chanchal K. Roy, and Kevin A. Schneider. Interactive visualization of bug reports using topic evolution and extractive summaries. In *Proceedings of International Conference on Software Maintenance and Evolution (ICSME2014)*, pp. 421–425, 9 2014.

[9] 大平雅雄, 横森励士, 阪井誠, 岩村聡, 小野英治, 新海平, 横川智教. ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム. 電子情報通信学会論文誌 D-I, Vol. J88-D-I, No. 2, pp. 228–239, 2 2005.

[10] 山西健司. データマイニングによる異常検知. 共立出版, 東京, 2005.

[11] 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, Vol. 28, No. 3, pp. 12–28, 8 2011.